Mining Simple and Complex Patterns Efficiently Using Binary Decision Diagrams

Elsa Loekito

Submitted in total fulfilment of the requirements
of the degree of Doctor of Philosophy

June 2009

Department of Computer Science and Software Engineering
The University of Melbourne
Victoria, Australia

Abstract

Pattern mining is a knowledge discovery task which is useful for finding interesting data characteristics. Existing mining techniques sometimes suffer from limited performance in challenging situations, such as when finding patterns in high-dimensional datasets. Binary Decision Diagrams and their variants are a compact and efficient graph data structure for representing and manipulating boolean functions and they are potentially attractive for solving many problems in pattern mining. This thesis explores techniques for the use of binary decision diagrams for mining both simple and complex types of patterns.

Firstly, we investigate the use of Binary Decision Diagrams for mining the fundamental types of patterns. These include frequent patterns, also known as frequent itemsets. We introduce a structure called the Weighted Zero-suppressed Binary Decision Diagram and evaluate its use on high dimensional data. This type of Decision Diagram is extremely useful for re-using intermediate patterns during computation.

Secondly, we study the problem of mining patterns in sequential databases. Here, we introduce a new structure called the Sequence Binary Decision Diagram, which can be used for mining frequent subsequences. We show that our technique is competitive with the state of the art and identify situations where it is superior.

Thirdly, we show how Weighted Zero-suppressed Binary Decision Diagrams can be used for discovering new and complex types of patterns. We introduce new types of highly expressive patterns for capturing contrasts, which express disjunctions of attribute values. Moreover, to investigate the usefulness of disjunctive patterns for knowledge discovery, we employ a statistical methodology for testing their significance, and study their use for solving classification problems. Our findings show that classifiers based on significant disjunctive patterns can be more robust than those which are only based on simple patterns.

Finally, we introduce patterns for capturing second-order differences between two groups of classes, which can provide useful insights for human experts. Again, we show how binary decision diagrams can be deployed for efficiently discovering this type of knowledge.

In summary, we demonstrate that Binary Decision Diagrams, are a powerful and scalable tool in pattern mining. We believe their use is very promising for a range of current and future tasks in the data mining context.

Dedicated to my parents

Declaration

This is to certify that

i the thesis comprises only my original work towards the PhD except where indicated in the Preface;

ii due acknowledgement has been made in the text to all other material used;

iii the thesis is less than 100,000 words in length, exclusive of tables, figures, bibliographies, and appendices.

Elsa Loekito
Department of Computer Science and Software Engineering
University of Melbourne
June, 2009

Preface

This dissertation contains nine chapters. The first three chapters provide an introductory description of the background and related work. The last chapter summarises the thesis and proposes some future research issues. The remaining chapters cover the core research topics. No part of the thesis has been submitted for any degree; or ever been conducted while under employment.

Part of Chapter 4 was published in the Proceedings of the Sixth Australian Data Mining Conference (AusDM), in December 2007:

> Elsa Loekito and James Bailey. Are Zero-suppressed Binary Decision Diagrams Good for Mining Frequent Patterns in High Dimensional Datasets?. In *Proceedings of the Sixth Australian Data Mining Conference (AusDM)*, pages 139-150, Gold Coast, Queensland, Australia, in December 3-4, 2007.

A part of Chapter 6 was published in the Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), in August 2006:

> Elsa Loekito and James Bailey. Fast Mining of High Dimensional Expressive Contrast Patterns Using Zero-suppressed Binary Decision Diagrams. In *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 307-316, Philadelphia, PA, USA, in August 20-23, 2006.

A part of Chapter 7 was published in the Proceedings of the Thirteenth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), in April 2009:

> Elsa Loekito and James Bailey. Using Highly Expressive Contrast Patterns for Classification - Is It Worthwhile? In *Proceedings of the Thir-*

teenth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), pages 483-490, Bangkok, Thailand, in April 27-30, 2009.

A part of Chapter 8 was published in the Proceedings of the Seventeenth International Conference on Information and Knowledge Management (CIKM), in October 2008:

> Elsa Loekito and James Bailey. Mining Influential Attributes That Capture Class and Group Contrast Behaviour. In *Proceedings of the Seventeenth International Conference on Information and Knowledge Management (CIKM)*, pages 971-980, Napa Valley, California, USA, in October 26-30, 2008.

A part of Chapter 5 is to appear in the Knowledge and Information Systems Journal (KAIS):

> Elsa Loekito, James Bailey, and Jian Pei. A Binary Decision Diagram Based Approach for Mining Frequent Subsequences. To appear in *Knowledge and Information Systems Journal (KAIS).*

This dissertation was prepared by LaTeX. The algorithms included were writted in C/C++, compiled by GNU gcc compiler and run on the Solaris 9/x86.

Acknowledgements

There are no words to express my gratitude to my supervisor, Dr. James Bailey, who has selflessly taught me how to think 'outside the box' in the past three and a half years. The study presented in this dissertation would not have happened without his support, guidance, and encouragement. It is my privilege to have a highly dedicated supervisor like him. He has put much effort to help refine my skills in solving research problems, and communicating research ideas to a wide range of audience. He is also an excellent writer, and a meticulous reader. His constructive comments have made me mature in my research career.

It is my honor to have Professor Ramamohanarao Kotagiri, Professor Peter Stuckey and Associate Professor Chris Leckie as my committee members. I am very thankful to all of them who have provided insightful comments in regards to my research.

I must express my appreciation to Associate Professor Jian Pei from Simon Fraser University, Canada. His work in the area of pattern mining has been a great inspiration to my research. I can still remember the excitement when we were having effective discussions with my supervisor during his visit to Melbourne in 2006.

It is also a life time opportunity to be part of the machine learning research group in the Computer Science and Software Engineering Department, where I have met aspiring colleagues and friends: Hongjian Fan, Ce (William) Dong, Zhou (Joanne) Zhu, Roger Ting, Xiaonan Ji, Eric Bae, Bader Aljaber, Shaoyi Zhang, Geoff MacIntyre, and Elizabeth Antoine, just to name a few. This research is financially supported by the University of Melbourne and National ICT Australia (NICTA).

I whole-heartedly thank my parents, my sister, and my close friends, for their support. Without them, I would not have been able to persevere during my research and complete this dissertation. *Deo Gratias (thanks be to God). He exists before all things, and in Him all things hold together (Colossians 1:17)*

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Recent advances in information technology have made possible the collection of enormous amounts of data. Manual analysis of such data is impractical. Data mining technology enables the extraction of patterns from data, and presentation of knowledge for users. The usefulness of a data mining technique is determined by a number of factors, including the expressiveness and interpretability of the patterns. Expressive patterns are more complex, harder to mine than the simpler patterns, and they can also be harder to interpret. Mining expressive patterns is desirable for extracting useful knowledge, but they have not been able to be found using previous techniques. The increasing growth of database sizes, moreover, urgently calls for new mining methods which are efficient and highly scalable. Binary decision diagrams have proven to be a compact and efficient graph data structure for manipulating large scale boolean functions. Having been widely used in the area of VLSI/CAD, they are attractive for solving pattern mining problems. This thesis explores techniques for mining simple as well as complex patterns, based on the use of binary decision diagrams and their variants.

1.1 What is Data Mining?

Data mining [56] is a component of knowledge discovery, which is a process of extracting information from databases, defined as follows.

Definition 1. *[56, 53]* **Knowledge Discovery in Databases** *is the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data.*

Knowledge discovery can be divided into seven steps: data cleaning, data integration, data selection, data transformation, data mining, pattern evaluation, and knowledge presentation. Data mining can be defined as follows.

Definition 2. *[67]* **Data Mining** *is the process where intelligent methods are applied in order to extract patterns or information that satisfy a certain objective or interestingness measure.*

In technical terms, intelligent methods mean efficient algorithms, and the objective or interestingness measure determines the quality of the patterns. The patterns can either be predictive or descriptive, which are distinguished as follows: predictive patterns predict the unknown value of some data items; descriptive patterns extract salient features from the data. The typical predictive methods include classification (i.e. predicting the class label of unseen data) and regression (i.e. predicting the value of an unknown part of the data). Descriptive methods include association rule mining (i.e. identifying associations between data items), and pattern discovery (i.e. discovering patterns or trends within the data).

To be interesting and useful, it is of high importance that patterns satisfy the following criteria: interpretability, non redundancy, expressiveness, and tractability of computation. The quality of the patterns is also important, which can be measured by statistical significance. In our research, we focus on efficient methods for pattern discovery, and study expressive, interpretable, and statistically significant patterns.

1.2 Different Data Types

There exist different types of data, and the definition of data mining assumes the data to be of a certain type.

A first categorisation of data types can be made according to the nature of the database, which may be *transactional*, *sequential*, or *relational*. A transactional database consists of transactions where each transaction is described by a set of data items. A relational database also consists of transactions, but they are defined over a set of data attributes, and each attribute is defined over a set of domain values. Such transactions, in either a transactional or relational database, are referred to as *itemsets*. An itemset contains distinct and unordered items. On the other hand, in a sequential database, a *sequence* may contain duplicate items, and the items are ordered.

Example 1. $\{a, b, c, d\}$ *is an itemset. It may be a set of purchased items in a transaction in market basket data. Alternatively, each item may also represent an attribute value in a relational database. Itemsets* $\{a, b, c, d\}$ *and* $\{b, c, d, a\}$ *are equivalent, since an itemset contains unordered items.*

$\{a, a, c, a, b, b, d\}$ *is a sequence. In a DNA or protein sequence, each item may represent a nucleotide or an amino acid.* $\{a, a, c, a, b, b, d\}$ *and* $\{a, c, a, a, d, b, b\}$ *are two different sequences, since the items are ordered differently.*

In a relational database, the domain values of an attribute can be categorised into: *continuous*, and *discrete or categorical.* Continuous data can take values from an infinite set, such as the set of real numbers. Discrete data, on the other hand, can only take values from a finite set. Many data mining methods, such as those for mining patterns, assume discrete data. In the presence of continuous valued data, such methods can be used by applying a data discretisation method, which transforms continuous data to discrete data. For some methods, the data discretisation can have a great influence on the existence of the patterns, since through this data transformation, some information may be lost [52].

In our research, we study pattern mining techniques in the context of itemsets as well as sequences. We also address some of the problems which occur due to data discretisation.

1.3 Pattern-based Data Mining Methods and Their Applications

The most fundamental type of patterns are called *frequent itemsets* [4, 5], which capture data characteristics within a set of objects, e.g. transactions. Frequent patterns for sequence databases are called *frequent subsequences* [6, 3].

A frequent itemset is a combination of items which appears with frequency greater than or equal to a user-specified threshold. For example, the itemset $\{cereal, milk\}$, which is purchased in 90% of the transactions, is a frequent itemset for any frequency threshold of 90% or less. Frequent itemset mining is a well studied problem and very useful for other data mining tasks. It was first introduced for finding associations between items, called *association rules* [4].

There exist other types of frequent patterns, such as frequent subgraphs in a graph database, or frequent subtrees in a tree database, but in this thesis we will focus on the fundamental types, which include frequent itemsets and frequent subsequences.

Various extensions of frequent patterns have also considered a variety of constraints, other than minimum frequency, such as total average, minimum/maximum total value, or minimum length of patterns, which are useful for different domain applications. The use of such constraint-specific techniques is often called *constrained pattern mining* [68, 69, 28, 21].

Contrast patterns [37, 17, 139] are a useful class of constrained patterns for capturing contrasts, or differences, between two sets of data. The differentiation may be made between data across classes, or across time periods. A simple type of contrast patterns is called *emerging patterns* [37], and a variety of highly accurate classifiers based on emerging patterns have been proposed [42, 82, 50, 51]. Applications of contrast mining have been studied in many domains, such as the bioinformatics domain (a survey is presented in [12]), which employs efficient techniques to discover patterns in high dimensional biological data.

Trees are one of the most popular data structures used in existing pattern mining techniques [70, 61, 158, 21, 14]. Moreover, graph data structures, such as Zero-suppressed Binary Decision Diagrams (ZBDDs) [111], have also been used by several techniques, particularly for mining frequent itemsets [108, 109, 80, 112, 117]. Binary decision diagrams highly promote node re-use, which is not possible using trees. This sophistication motivates our research to investigate the advantages and disadvantages of using binary decision diagrams, instead of trees, for pattern mining.

1.4 Simple and Complex Patterns

Existing types of patterns, such as frequent patterns, frequent subsequences, and contrast patterns, may be limited in terms of the kind of knowledge they can discover and express.

Our research addresses two problems in contrast mining: i) finding expressive contrasts, ii) capturing contrasts between classes and across groups of classes. These types of contrasts have a higher complexity to compute than the existing patterns, which are considered *simple patterns*. An example of each of these patterns, which we refer to as *complex patterns*, follows shortly.

Example 2. *An expressive contrast may occur in a census data [71], where the com-*

bination ['aged between 40 and 60' and 'worked in a telecommunication industry or a transportation industry'] *strongly differentiates males from females. Current definition of contrast patterns can only express conjunctions, such as* ['aged between 40 and 60' and 'worked in a telecommunication industry']*, but cannot express a disjunction between the telecommunication and transportation industries.*

Revisiting the previous contrast example from the census data, an expert may be interested to compare differences between males and females across two categories of individuals. None of the existing contrast pattern definitions is able to identify such high-level contrasts, which we call *second-order contrasts.* These complex patterns are novel types of patterns. No techniques have been proposed for mining them.

In this thesis, we will propose new definitions of these complex patterns, and show that binary decision diagrams can serve as a powerful tool for mining them efficiently.

1.5 Research Aim and Challenges

To summarise, the aim of our research is to study the mining of simple, as well as complex types of patterns, using graph-based data structures based on Binary Decision Diagrams [25]. We address the following questions:

- *To what extent can existing pattern mining techniques, such as frequent itemset mining and frequent subsequence mining, be improved by using graph-based data structures?*
- *How can these new data structures allow data mining techniques for the discovery of complex knowledge, such as highly expressive contrasts and second-order contrasts, which existing techniques have not been able to discover?*

When mining simple patterns, we focus on the challenging high dimensional data, in which a large volume of patterns exist. By allowing disjunctions, expressive contrasts are computationally expensive to mine, since more combinations need to be explored. Furthermore, their use for improving classification accuracy remains an open question, which we will investigate in our research. In second-order contrast mining, existing types of contrast patterns do not readily allow second order differentiation, and an overwhelming volume of output patterns is often returned to the user.

Binary Decision Diagrams are attractive for solving large-scale data mining problems. They have been used in many VLSI/CAD [103] applications, but their effectiveness for data mining purposes remains an open question. Despite having attractive properties which are potentially useful for performing efficient pattern mining, not all of the existing optimisations in current pattern mining techniques are suitable for being adopted into a graph data structure. Moreover, different techniques may be more suitable for different circumstances, which may be influenced by the characteristics of the patterns and the input data.

1.6 Research Contributions and Organisation of the Thesis

Listed below are our contributions and the organisation of this thesis:

- Chapter 2 provides background on pattern mining and gives a review on the existing pattern mining techniques.
- Chapter 3 provides background on Binary Decision Diagrams (BDDs), as well as some of the variants which are related to the data structures used in our work.
- Our first contribution is to investigate whether employing Binary Decision Diagrams can improve existing pattern mining techniques. We study the problem of mining high-dimensional patterns in different types of databases, transactional, relational, and sequential, which includes frequent itemset mining and frequent subsequence mining.

  In Chapter 4, we propose a new and original variant of BDDs, called *Weighted Zero-suppressed Binary Decision Diagrams (Weighted ZBDD)*, which are powerful for mining frequent patterns with high scalability. In Chapter 5, we also propose a new data structure for mining sequential patterns, called *Sequence Binary Decision Diagrams.*

  Our results show that the BDD-based techniques can outperform the tree-based techniques, particularly when a large volume of patterns exists. The BDD's ability to re-use intermediate computation results helps to avoid redundant computation, which is a novel feature in pattern mining.
- In Chapter 6, we study the use of Zero-suppressed Binary Decision Diagrams (ZBDDs) for mining contrasts. We define novel types of expressive contrasts, called

disjunctive emerging patterns. These patterns generalise the previous definition of contrast patterns, such as emerging patterns, which can only express simple combinations (i.e. conjunctions) of data items.

Being more expressive, many more disjunctive patterns may exist than the simple emerging patterns. We propose algorithms for mining both the simple emerging patterns as well as expressive patterns, based on the use of ZBDDs or weighted ZBDDs. Our results show that ZBDDs allow the large search space of disjunctive patterns to be explored efficiently, and are able to substantially improve the scalability and time efficiency of previous techniques.

- In Chapter 7, we study the use of disjunctive emerging patterns for classification. The disjunctive characteristics of these patterns enable contrasts to be discovered in circumstances where simple emerging patterns rarely exist. To improve the robustness of the classifier, we propose a methodology for testing the significance of the disjunctions. Previous work on statistical significance, such as for finding significant association rules, considers pure conjunctions, but in our work, we consider disjunctions.

  We conduct experiments to investigate whether the use of disjunctive emerging patterns for classification is worthwhile. Our results indicate that the significant disjunctive patterns do allow more robust classifiers to be built, obtaining higher accuracies over previous classifiers based on emerging patterns, especially when the data is sparse.

- In Chapter 8, we study the problem of mining second order contrasts between multiple groups of classes. These contrasts allow two levels of analysis: within group contrasts and across groups contrasts. We call such patterns *group discriminative contrasts*, and these are a new and complex type of contrast pattern. To address the issue of pattern interpretability, we propose a method for finding the attributes which have the most impact in group discriminative contrasts.

  We evaluate our technique using a real census [71] data set, as well as a few other data sets. From the census data, we are able to identify attributes which otherwise cannot be identified by other techniques, which have strong influence in discriminating males from females, conditional to their race background.

  We propose an algorithm for mining these second order contrasts, based on weighted Zero-suppressed Binary Decision Diagrams. It allows both the patterns and the influential attributes to be found simultaneously, by processing all classes at once and pushing multiple constraints deep inside the mining routine. This level of mining complexity has not been considered in previous techniques.

- Finally, Chapter 9 presents the conclusions of our study, and proposes future research directions.

Chapter 2

Overview of Pattern Mining Techniques

There exist numerous pattern mining techniques. This chapter presents an overview of some of the influential techniques for mining various types of patterns, which include frequent itemsets, frequent subsequences, constrained patterns, and contrast patterns. Firstly, we give some preliminary definitions.

2.1 Preliminaries

This preliminary section provides terminologies which will be used in the remainder of this chapter and the remainder of this thesis. Assume we have a dataset D defined upon a set of k attributes (also referred as dimensions) $\{A_1, A_2, \ldots, A_k\}$.

An *item* is an element of the domain values of an attribute. For every attribute A_i, the domain of its values (or items) is denoted by $dom(A_i)$. Let I be the aggregate of the domains across all the attributes, i.e. $I = \bigcup_{i=1}^{k} dom(A_i)$. An *itemset* is a subset of I. Let P and Q be two itemsets. P *contains* Q if Q is a subset of P, i.e. $Q \subseteq P$. Alternatively, P is a *superset* of Q.

Example 3. *Suppose* $I = \{a, b, c, d, e, f, g\}$*. An itemset* $P = \{a, b, e\}$ *is a subset of* $Q = \{a, b, d, e, f\}$*, and* Q *is a superset of* P*.*

A *dataset* is a collection of transactions, where each *transaction* is an itemset. The *size* of a dataset D is the number of transactions in D, denoted by $|D|$. The *frequency*

id	Itemset
t_1	$\{a, b, e\}$
t_2	$\{b, c, e\}$
t_3	$\{a, b, c, d\}$
t_4	$\{a, b, c, e\}$

Figure 2.1: A dataset

of an itemset P in D, denoted by $count(P, D)$, is the number of transactions in D which contain P. The *support* of P, denoted by $support(P, D)$, is the frequency divided by the size of D, i.e. $\frac{count(P,D)}{|D|}$, where $0 \leq support(P, D) \leq 1$.

Example 4. *Figure 2.1 shows an example of a dataset D. The size of D is 4, i.e. $|D| = 4$. The frequency of itemset $\{a, b\}$ in D is 3, since it is contained by transactions t_1, t_3, and t_4. Its support is $\frac{3}{4} = 0.75$.*

2.2 Frequent Itemsets and Their Applications

A frequent itemset is defined as follows.

Definition 3. *Given a minimum support threshold α, a* **frequent itemset** *[4] in a dataset D is an itemset P such that $support(P, D) \geq \alpha$.*

Frequent itemsets exhibit an anti-monotonic APRIORI property, that is, all subsets of a frequent itemset are also frequent itemsets.

Theorem 1. APRIORI property *[4]: for any two itemsets P and Q, such that $Q \subseteq P$, if P is a frequent itemset, then Q is also a frequent itemset.*

Proof. Suppose P is a frequent itemset, given a minimum support threshold α. Let Q be a subset of P, i.e. $Q \subseteq P$. For every transaction T which contains P, i.e. $P \subseteq T$, T also contains Q, i.e. $Q \subseteq T$. However, for every transaction T which contains Q, i.e. $Q \subseteq T$, T may not contain P. Therefore, $support(Q, D) \geq support(P, D)$. If P is a frequent itemset, i.e. $support(P, D) \geq \alpha$, then Q is also a frequent itemset, i.e. $support(Q, D) \geq \alpha$. □

The number of frequent itemsets can be huge, especially for low α and in circumstances where many items occur frequently. Mining frequent itemsets then can be very time consuming [5]. The number of itemsets to be checked is $O(2^N)$, where N is the number of items. To address this problem, concise representations of frequent itemsets have been defined, such as closed frequent itemsets and maximal frequent itemsets.

Definition 4. *A* **closed frequent itemset** *[128] is a frequent itemset which is not a subset of another frequent itemset with the same frequency.*

Lemma 1. *For any two closed frequent itemsets P and Q, such that Q is a subset of P, the support of P is less than the support of Q [128], i.e. $support(P, D) < support(Q, D)$.*

Maximal frequent itemsets are the most concise representation which removes frequent itemsets which are subsets of another frequent itemset. Maximal frequent itemsets, however, do not allow support information to be deduced for their subsets.

Definition 5. *A* **maximal frequent itemset** *[18] is a frequent itemset which is not a subset of another frequent itemset.*

Lemma 2. *For any two maximal frequent itemsets P and Q, Q is not a subset of P, and P is not a subset of Q [18].*

Example 5. *Consider an example dataset shown in Figure 2.1. Given a minimum support threshold $\alpha = 0.5$. The frequent itemsets (with their frequencies) are $\{a\} : 3$, $\{b\} : 4$, $\{c\} : 3$, $\{e\} : 3$, $\{a, b\} : 3$, $\{a, c\} : 2$, $\{a, e\} : 2$, $\{b, c\} : 3$, $\{b, e\} : 3$, $\{c, e\} : 2$, $\{a, b, c\} : 2$, $\{a, b, e\} : 2$, $\{b, c, e\} : 2$.*

The closed frequent itemsets are $\{b\}$, $\{a, b\}$, $\{b, c\}$, $\{b, e\}$, $\{a, b, c\}$, $\{a, b, e\}$, $\{b, c, e\}$. Itemset $\{a\}$ is not a closed frequent itemset because its superset $\{a, b\}$ has the same support of 3. For the same reason, itemsets $\{c\}$, $\{e\}$, $\{a, c\}$, $\{a, e\}$, and $\{c, e\}$ are not closed frequent itemsets.

Itemsets $\{a, b, c\}$, $\{a, b, e\}$ and $\{b, c, e\}$, are maximal frequent itemsets because none of their supersets are frequent itemsets.

The sets of frequent itemsets (**FI**), closed frequent itemsets (**CFI**), and maximal frequent itemsets (**MFI**), follow the following relationship [29]: **MFI** $\subseteq$ **CFI** $\subseteq$ **FI**. The set **MFI** can be orders of magnitude smaller than **CFI**, and the set **CFI** can be smaller than **FI**. The **CFI** is useful, since it is smaller than **FI**, yet still contains enough information about the support of the itemsets [127]. When there exist very long patterns (i.e. patterns containing many items), however, **FI** and **CFI** may become very large, thus, finding **MFI** is more practical [18]. The set of maximal itemsets have been shown to be adequate in high dimensional databases, such as biological databases [135].

Frequent itemsets have been shown to be useful for finding association rules and for building classifiers. Association rules [4], are typically found using two constraints: a *minimum support*, and a *minimum confidence* of each rule. The confidence measures the association between items in an itemset. Those rules can be found in two phases: i) find the frequent itemsets with respect to the minimum support, ii) find the highly confident frequent itemsets. Work in [92] studied class-association rules, also known as classification rules, which find strong associations between itemsets and the classes in the dataset, which are particularly useful for building a classifier.

There exist various types of association rule based classifiers [163, 91, 152], which may employ additional constraints to improve classification performance. Sequential association rules have also been studied in [161], and these are useful for finding associations between sequences of events in a sequence database.

2.3 Review of Frequent Itemset Mining

The first technique for mining frequent itemsets was introduced for mining association rules [4, 104]. Association rules are useful for both knowledge discovery and classification. Numerous association rule mining techniques have been proposed, varying in the data representations, the candidate generation techniques, and the candidate pruning techniques. Each of them may be more suitable for different characteristics of the output patterns, or the input data, e.g. data sparsity/density. There exists an association rule based classifier called ARC-AC, which has also been studied for various domains, such as medical images [11], and text documents [164]. A detailed review of the state-of-the-art of frequent pattern mining techniques and their future directions can be found in [66].

Frequent itemset mining techniques can be classified into two categories of approaches: the generate-and-test approach [4], and the pattern growth approach [70]. The first category of approach generates candidate patterns by exploring the itemset lattice to search for candidate patterns, while pushing the frequency constraint to prune its search space. The second approach avoids the generation of infrequent candidate patterns by using the input database as a guide for pruning. Each of them has advantages and disadvantages, which we will describe shortly.

2.3.1 Generate-and-test Frequent Itemset Mining Approach

The APRIORI [4] algorithm was one of the first generations of frequent itemset mining algorithms. It generates pattern candidates in a breadth-first, bottom-up fashion. It finds length $(k+1)$ candidates from the frequent itemsets of length k $(k \geq 1)$, and counts the frequencies of the candidates as they are generated.

Example 6. *Recall the dataset used in Example 5. Given a minimum support threshold* $\alpha = 0.5$*, the APRIORI algorithm starts with each item being a pattern candidate, tests its frequency, and finds that item 'd' is not frequent. Thus, it finds the length-*1 *frequent itemsets:* $\{a\}, \{b\}, \{c\}, \{e\}$.

Then, length-2 *candidates are generated from the length-*1 *frequent itemsets, by finding their pair-wise cross-products (e.g.* $a \times b = ab$*):* $\{a, b\}$, $\{a, c\}$, $\{a, e\}$, $\{b, c\}$, $\{b, e\}$, $\{c, e\}$. *From these candidates,* $\{a, b\}$, $\{a, e\}$, $\{b, e\}$, *and* $\{c, e\}$ *are frequent itemsets, since their support values exceed* 0.5. *Length-*3 *candidates are generated from the length-*2 *frequent itemsets, and so on, using the same procedure. The iteration terminates when only one length-*k *frequent itemset is found, or none of the candidates is frequent.*

Frequency counting is the core computation in frequent itemset mining. The naive approach for calculating the frequency requires one database scan for each pattern candidate. This, however, is inefficient given the exponential search space. For a database of N items, the number of pattern candidates is $O(2^N)$. Work in [99] proposed a counting inference method, which allows more efficient frequency counting. They introduced the notion of *key patterns*, which are minimal itemsets that have the same support and occur in the same transactions. Recall that closed itemsets are the maximal itemsets of the same pattern space as key patterns.

Key-patterns have the useful property that all k-itemset candidates which are supersets of a key-pattern inherit the frequency of the key-pattern. Once an infrequent key-pattern has been identified, none of its supersets needs to be generated. On the other hand, all supersets of a frequent key-pattern that occur in the database are also frequent. This technique is most effective when the number of key-patterns is significantly smaller than the number of the actual patterns, which often occurs in a dense dataset. The results in [99, 100] indicate that the algorithm is superior than the APRIORI technique for mining the complete set of frequent itemsets.

Efficient successors of APRIORI for finding maximal frequent itemsets include

tid	*a*	*b*	*c*	*d*	*e*
t_1	1	1	0	0	1
t_2	0	1	1	0	1
t_3	1	1	1	1	0
t_4	1	1	1	0	1

(a) Vertical layout

tid	*a*	*b*	*c*	*d*	*e*
t_1	1	1	0	0	1
t_2	0	1	1	0	1
t_3	1	1	1	1	0
t_4	1	1	1	0	1

(b) Horizontal layout

Figure 2.2: Vertical and horizontal layout of dataset D

MAFIA [29], DepthProject [1], and GenMax [63]. Differing from the Apriori-based approach, which performs a breadth-first search through the itemset-lattice, these techniques perform a depth-first search that finds maximal frequent itemsets early, by integrating strong search pruning strategies. The following discussion compares the performance between those algorithms.

MAFIA and DepthProject similarly perform depth-first-search when exploring the itemset lattice, except they use different database representations. MAFIA uses a vertical bitmap representation where each item is associated with a list of zeros and ones. The 'ones' in the bitmap represent the occurrences of that item in the database. Instead of using a vertical database layout, DepthProject uses a horizontal database representation, which allows more efficient support counting when the dataset is sparse, since the bitmap contains many zeros. Figure 2.2 shows the vertical and horizontal layout of the example dataset D shown in Figure 2.1.

GenMax aims to improve the MAFIA algorithm by adopting compact database representations. It associates each transaction with a transaction identifier, or *tid*, and represents the occurrence of each item by a list of *tids* from the transactions which contain that item. The frequency is calculated based on the *tid*-lists, instead of bitsets which are used by both MAFIA and DepthProject.

A performance study of MAFIA, in comparison to the other algorithms in the same category is presented in [61]. It is shown that MAFIA works best when mining long itemsets in a dense dataset(contain highly similar transactions). In sparse datasets, DepthProject is faster than MAFIA for high support thresholds, but MAFIA performs well for low support thresholds. Compared to both DepthProject and MAFIA, GenMax is superior for higher supports where the bitsets contain many more *zeros* than *ones.*

2.3.2 Pattern Growth Mining Approach

The second category of approaches, called the *pattern growth* [70], avoids generating infrequent candidates. It employs a partitioning-based, divide-and-conquer strategy, rather than the Apriori bottom-up generation of frequent itemsets. The pattern-growth decomposes the mining task into a set of smaller tasks, and incrementally grows patterns using smaller database projections. A partially grown pattern is called a *prefix*. For an item x, the algorithm projects an *x-conditional database*, which excludes transactions which do not contain that item. The frequent patterns are found recursively by projecting subsequent conditional databases.

The first pattern-growth algorithm, *FP-growth* [70], is based on a prefix tree data structure, called the *FP-Tree (Frequent Pattern Tree)*. An FP-tree is a compact representation of a database. It contains the frequent items, and every branch represents an itemset, allowing common prefixes to be shared between itemsets. Each node in the prefix tree represents a prefix and its frequency. An FP-tree is accompanied by a header table, which stores the total frequency of each item in the database. Nodes with same label (i.e. item) are linked, and the head of the link lists for each item is stored in a header table.

Figure 2.3 shows an example of an FP-tree for the example dataset given in Figure 2.1. Since an FP-tree stores information about the frequency of the contained itemsets, frequency calculation be performed efficiently, especially when many of the itemsets share similar prefixes. Such a circumstance occurs in a database that contains many frequent items.

Compared to the generate-and-test approach which requires multiple database scans, the FP-growth algorithm only needs two database scans. The first scan counts the frequency of each item, and the second one builds the FP-tree representation of the database.

Variants of FP-trees have also been shown to be useful for finding the other types of patterns, such as sequential patterns [48], constrained patterns [21], and contrast patterns [14]. We will provide more details about these other types of patterns later.

Dynamic Item Ordering

The technique proposed in FP-growth [70] uses a dynamic item ordering, which results in smaller conditional databases and, in turn, more efficient overall mining time.

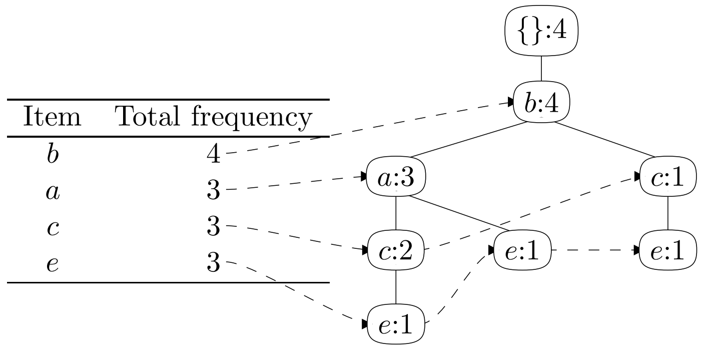


Figure 2.3: An FP-tree

The FP-growth algorithm orders the items in each FP-tree by decreasing frequency, putting the most frequent item at the top of the tree. This item ordering aims to find long frequent itemsets early, and reduce the width of the conditional FP-trees. When performing a conditional database projection, the least frequent item in the FP-tree is chosen when growing the first prefix. This ordering produces tall-narrow FP-trees for low frequency items and short-wide trees for high frequency items.

Example 7. *Consider the FP-tree in Figure 2.3. Suppose the frequency constraint requires the patterns to have frequency at least 2. The tree is traversed starting with the lowest item, i.e. e, projecting a conditional database* $\{\{b, a, c\}\}$*. Since the conditional database contains only one transaction, it returns all subsets of* $\{b, a, c\}$ *as frequent patterns. Returning to the initial FP-tree, it then visits the second-lowest item, i.e. e, and projects a conditional database* $\{\{a, b\}, \{a, b\}\}$*. Items c and d are omitted from the e-conditional database, since they are infrequent in this conditional database.*

Optimised Implementations of FP-growth

Prior to the construction of every conditional FP-tree, two scans of the current FP-tree database are required. The ordering of the items is important for efficiency of the algorithm. The first scan finds the frequent items, and the second scan constructs the new tree. In a sparse dataset, the FP-tree does not allow many prefixes to share nodes, resulting in numerous large conditional FP-trees. An improved FP-growth method, namely FP-growth* was proposed in [64], which uses an additional *array* for each FP-tree, and allows early pruning and faster construction of the conditional FP-trees. The array stores the frequency of 2-itemsets. As a result, the frequent items in the subsequent conditional databases can be found from this array, saving one database scan.

Work in [64] proposed extensions of the FP-growth* algorithm for mining maximal frequent itemsets and closed frequent itemsets, called as FPmax* and FPclose*, respectively. Compared with the other algorithms, FP-max* [64] is faster than MAFIA and GenMax for dense datasets, where many long patterns exist. For sparse datasets, on the other hand, which contain short patterns, it is less efficient due to the overhead cost in constructing many conditional FP-trees, especially when the support threshold is low. Similar trends are found in the comparison between FP-close* and the extensions of MAFIA for finding closed frequent itemsets.

Work in [167] introduced a pattern-growth algorithm, called CHARM, for mining closed frequent itemsets, which explores both the itemset space and the *tid*-set space. It uses an Itemset-tidset-tree (or IT-tree for short), and grows prefixes from the IT-tree. An optimised FP-growth algorithm, called CLOSET+, for mining frequent closed itemsets was proposed in [158].

There exist numerous alternative implementations [62, 78] of pattern growth which mainly differ in their database representations. Another tree based algorithm called *Leap* [165] exists, which is optimised for finding maximal frequent itemsets. It uses a variant of FP-trees called Headerless Frequent Pattern trees (HFP-trees). The HFP-tree does not have a header table. In addition to support, each node has a second variable, called *participation*, which represents the number of times the node has participated in the already-counted patterns at a given time in the mining process. In general, the algorithm avoids generating non-maximal candidates, and the HFP-trees are used for efficiently calculating the support of the maximal frequent itemsets.

Different data structures for storing the conditional databases are more suitable for different situations. Tree data structures, such as the FP-tree, or other tree-based data structures [64, 94, 132, 45, 165], are suitable for dense data. Tree-based techniques can benefit from the sharing of common prefixes, but they are not very suitable for sparse data, due to poor compactness of the tree representations.

On the other hand, an array structure is favorable when the conditional databases are sparse, i.e. the frequent patterns are short. The LCM algorithm [151] uses a combination of prefix-trees, bitmaps, and array. Their selection of data representation type is based on heuristics regarding the data density.

There exist two graph-based algorithms for mining frequent patterns. A recent implementation of LCM [117] uses a Zero-suppressed Binary Decision Diagram (ZBDD) for compactly storing the output patterns, and it performs well for mining patterns in sparse datasets. Another algorithm that uses a ZBDD was proposed in [113], called *ZBDD-growth*, which employs the FP-growth framework. It has a limited scalability, though, when mining a large number of patterns. More details about this algorithm will be given in Chapter 3.

2.3.3 Row-wise Pattern Mining In High Dimensional Datasets

The row-wise mining approach [125] explores row combinations instead of item combinations. Such a technique is suitable for mining patterns in the high dimensional biological datasets, which typically contain only a few rows, but a large number of dimensions or data attributes.

Based on the relationship between closed frequent itemsets and their transaction-id-sets [167], several row-wise mining algorithms [125, 124, 136, 34, 96] have been proposed for finding closed patterns in sparse high-dimensional microarray datasets. This row-wise mining approach has two advantages: it does not have support counting overhead, and all of the generated candidates are closed itemsets. The following are some terminologies used in the row-wise mining framework.

Definition 6. *Given an itemset P, bit_support(P) denotes the set of row-ids in which P occurs Let $\mathcal{R}$ be the set of row-ids. A* **rowset** *is a subset of $\mathcal{R}$ [136]. Given a rowset R, row_support_set(R) is the maximal itemset which occurs in all rows in R.*

In this thesis, we use the following support definition for rowsets.

Definition 7. *The support of row_support_set(R), denoted rsupport(R), is the number of row-ids in R relative to the number of transactions in D, i.e. rsupport$(R) = \frac{|R|}{|D|}$.*

The row-support-set of a given rowset R is a closed frequent itemset if the row-support, i.e. $rsupport(R)$, is at least equal to the minimum support threshold. The following theorem holds between a closed frequent itemset and the row-support-set of its corresponding row-set R.

Theorem 2. *A* **closed frequent itemset** *P is a a row_support_set(R), such that rsupport$(R) \geq \alpha$, and rowset R is the bit-support of P, i.e. $R =$ bit_support(P).*

There exist other works on finding patterns in microarray datasets which study different type of constraints, such as those in [24, 40]. Many variants of the row-wise mining techniques are based on the closure property of itemsets [33, 157], and *representative sets* [126]. In such a representative data description method, some of the patterns may be underrepresented. Due to this reason, we prefer the complete pattern representation.

2.4 Frequent Patterns in Sequential Databases

In this section, we will consider a type of frequent pattern which is suitable for sequential data, known as a *frequent subsequence*. The fundamental difference between sequences and itemsets is that the data values or items in a sequence are ordered and may be duplicated, which is not the case in an itemset. We will give the definitions and terminologies, and will review the techniques for mining frequent subsequences, shortly.

2.4.1 Definition of Frequent Subsequences

Let I be the set of items. A *sequence* S over set I is an ordered list of items, $e_1e_2 \ldots e_m$ where $e_j \in I$, for $1 \leq j \leq m$.

Each item in a sequence S is called an *element* of S. The j-th element that occurs in S is denoted by $S[j]$. The number of elements in a sequence, m, is referred as the *length* of S, and it is denoted by $|S|$. Each number between 1 and m is a *position* in S. Set I is also called as the *alphabet domain* of S. For instance, in a DNA data set, the set I contains alphabet letters A, C, G, T. An item from the alphabet domain can occur multiple times as different elements of a sequence. A sequential dataset D is a collection of sequences, defined upon a domain set of items, I. The number of sequences in D is denoted by $|D|$.

A sequence $P = a_1a_2 \ldots a_m$ is a *supersequence* of another sequence $Q = b_1b_2 \ldots b_n$ ($n \leq m$), and q is a *subsequence* of p, if there exist integers $1 \leq i_1 < i_2 < \ldots < i_n \leq m$ such that $Q[1] = P[i_1]$, $Q[2] = P[i_2]$, $\ldots$, $Q[n] = P[i_n]$. We say that P *contains* the subsequence Q. The frequency of a sequence P in D, denoted $count(P, D)$, is the number of sequences in D which contain P. The *support* of P in D, denoted $support(P, D)$, is the relative frequency of P with respect to the number of sequences in D, i.e. $support(P, D) = \frac{count(P)}{|D|}$.

Sequence Q is a *prefix* of P if Q is a subsequence of O, and $Q[i]$ is equal to $P[i]$

for all $1 \leq i \leq |Q|$. Sequence Q is a *suffix* of P if Q is a subsequence of P, and $Q[i]$ is equal to $P[j]$ for all $1 \leq i \leq |Q|$, $j = (|P| - |Q| + i)$. An *x-suffix* of a sequence P is a suffix of P which begins with item x. Moreover, it is a *longest x-suffix* in P if it begins with the first occurrence of x in sequence P.

The task of mining frequent subsequences from a dataset D is defined as finding all subsequences from D whose supports are at least *min_support*, formally defined as follows.

Definition 8. *Given a positive minimum support threshold α, P is a* **frequent subsequence** *if the support of P in dataset D is at least α i.e. support$(P, D) \geq \alpha$, where $0 \leq \alpha \leq 1$.*

Many of the techniques for finding frequent subsequences [6, 3, 105, 166, 31, 48] are developed based on frequent itemset mining, since frequent subsequences also exhibit the anti-monotonic *Apriori* property [6]. That is, for every sequence P and its subsequence Q, *support*$(P) \leq$ *support*(Q).

Theorem 3. APRIORI property *[6]: For any two sequences P and Q such that $Q \subseteq P$, if P is a frequent subsequence in a dataset D, then Q is also a frequent subsequence in D.*

Example 8. *Consider the sequential dataset example in Figure 2.4. Subsequence 'aba' has a frequency of 3, since it occurs in sequences s_1, s_4, s_5. Based on the anti-monotonic property, the frequency of each of its subsequences, i.e. a, b, aa, ab, ba, is no less than 3.*

Since every prefix of a sequence P is a subsequence of P, frequent subsequences also have a prefix anti-monotonic property [131]. That is, every sequence P and its prefix Q, *support*$(P) \leq$ *support*(Q). According to Theorem 3, therefore, if P is a frequent subsequence, then all of its prefixes are also frequent subsequences. This prefix anti-monotonic property is heavily used for pruning infrequent candidates when mining frequent subsequences [6, 105, 3, 129, 47].

2.4.2 Review of Frequent Subsequence Mining Techniques

Similar to the categorisation of approaches for finding frequent itemests, techniques for finding frequent subsequences can also be categorised into two approaches: i) candidate

id	Sequence
s_1	*abbac*
s_2	*bcabc*
s_3	*bbabc*
s_4	*aacba*
s_5	*abbba*

Figure 2.4: A sequence dataset

generation and test approach, and ii) prefix growth approach. However, mining subsequences is more challenging, since items in a frequent subsequence must follow the same ordering as they appear in the input sequences. Moreover, an item may occur multiple times in a sequence, but can only occur once in an itemset. Due to these differences, the dynamic item ordering technique used by frequent itemset mining algorithms [4, 70], is not applicable for frequent subsequence mining.

Apriori, AprioriAll, AprioriSome [6] were the first techniques in sequential pattern mining, which are based on the Apriori property. They scan the database once to find the frequent single items, e.g. x and y, and then combine the pair of frequent items to get length-2 candidates, e.g. xy and yx. Length-3 candidates are found by combining the frequent subsequences of length two, and so on. Subsequently, more efficient techniques were proposed, such as *GSP* [3], *PSP* [105] and *SPADE* [166].

GSP (Generalized Sequential Pattern) [3] follows the Apriori, candidate generation-and-test, framework. Support counting is the major cost in the GSP algorithm, which requires one database scan for each pattern candidate. *PSP* (Prefix Sequential Pattern) [105] is similar to GSP, except that PSP introduces the use of a prefix-tree to perform the mining procedure. *SPADE*, proposed in [166], is based on decomposing the pattern lattice into smaller sub-lattices, and decomposing the mining task into mining in those smaller sub-lattices. For counting support, SPADE uses a *vertical*, instead of a horizontal, data representation .

PrefixSpan [129] adopts the framework of FP-growth [70], by growing prefixes of the frequent subsequences and projecting conditional databases. It prunes the infrequent candidates based on the prefix anti-monotonic property, and it does not use a tree structure. Optimisation techniques used by PrefixSpan include pseudo-projection and bi-level database projection. The pseudo-projection technique uses *pointer-offset* pairs for finding the conditional databases, instead of physically creating them. However, it is only possible when the database fits in the main memory. The bi-level projection allows fewer and smaller databases being projected, by inducing a conditional database for each length-2 prefix. Work in [129] shows that PrefixSpan is the most efficient and scalable for mining long subsequences.

WAP-mine [130] uses a WAP-tree (Web Access Pattern tree) as database representation, which is similar to the FP-tree [70] in frequent itemset mining. Each node in a WAP tree is labeled, and nodes with the same label are linked together. Each branch represents a sequence. Unlike the other algorithms, WAP-mine grows suffixes of frequent subsequences. The use of a prefix tree aims to achieve data compression, and quick identification of frequent prefixes. Similar to FP-growth for mining frequent itemsets, WAP-mine also uses the least-frequent-item ordering which aims to generate smaller and fewer conditional databases. However, the item ordering does not allow much optimisation due to the possible multiple occurrences of an item in a sequence, and the number of conditional databases can be enormous.

PLWAP [48] improves WAP-mine by avoiding physical construction of the conditional databases. PLWAP uses a PLWAP-tree (Pre-order Linked Web Access Pattern tree) as data structure with a special code annotating each node which allows the initial database to be re-used for the conditional databases. The algorithm follows the prefix-growth framework, but it can be computationally expensive when long subsequences exist.

2.5 Mining Frequent Patterns With Additional Constraints

Frequent pattern mining is based only on a single frequency constraint, which is anti-monotonic. Work in [68, 69] has explored various constraints which can be pushed into frequent pattern mining. A class of *monotone* constraints, namely convertible monotone constraints, were also studied. A monotone constraint specifies that if an itemset satisfies the constraint, then all of its supersets also satisfy the constraint. Various interesting applications exist where both anti-monotone and monotone constraints are used. An example is given shortly.

Example 9. *In market basket data analysis, a marketing analyst might be interested in finding combinations of items which occur frequently in the database, for which the sum of their price is higher than a certain threshold. The minimum frequency is an anti-monotone constraint, whereas the minimum total price is a monotone constraint.*

Unlike frequent patterns, patterns which satisfy both monotonic and anti-monotonic

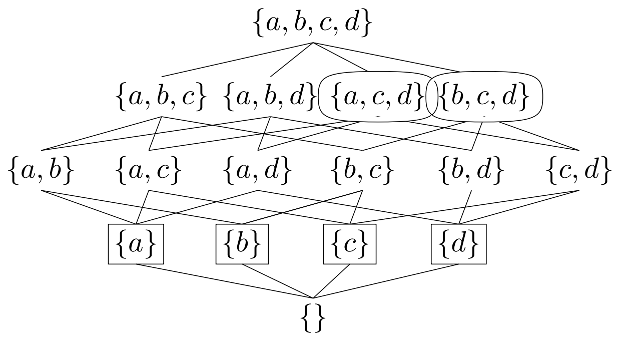


Figure 2.5: An itemset lattice with a subalgebra $(\{\{a\},\{b\},\{c\},\{d\}\}, \{\{a,c,d\},\{b,c,d\}\})$

constraints cannot be fully represented using only the maximal itemsets. Work in [69, 28, 20, 21, 46] proposed efficient techniques for pushing both constraints into pattern mining. The techniques for finding these types of patterns can be categorised into: i) an itemset lattice traversal, and ii) an FP-tree based approach. We will shortly give an overview of *DualMiner* [28] and *FP-Bonsai* [21], which represent the two respective approaches.

DualMiner [28] uses the concept of subalgebras for finding the set of patterns that satisfy the given constraints. A subalgebra is a top-bottom pair of itemsets, (T,B), which represents the set of itemsets which are supersets of B and also subsets of T. T and B respectively correspond to the maximal and the minimal itemsets in the set. When mining the patterns, DualMiner explores the subalgebra space starting with the largest subalgebra $(\emptyset, I)$, where I is the set of all items. It explores subsets of the subalgebra in a depth-first manner, while checking the constraints. Exploring subsets of the subalgebra confines the itemset lattice from both directions, from the top as well as the bottom. For example, Figure 2.5 shows an example of subalgebra $(\{a\},\{b\},\{c\},\{d\}, \{\{a,c,d\},\{b,c,d\}\})$, given $I = \{a,b,c,d\}$.

Due to its similarity to MAFIA in the depth-first search strategy, optimisation techniques in MAFIA can be adopted for efficient support counting and for pruning non-maximal good subalgebras. DualMiner performs well when the anti-monotone constraint is selective, but its performance decreases for less selective anti-monotone constraints.

The FP-Bonsai [21] algorithm considers a class of monotone constraints, namely the "local" monotone constraints, which depend solely on the patterns and not on the transaction database. Examples of such constraints in the market basket data domain include cardinality ($|X| >= n$), and sum of prices ($sum(X.price) >= n$). Conceptually, the FP-Bonsai algorithm incorporates the FP-growth algorithm with a data reduction technique [20], based on the local nature of the monotone constraint, which allows the FP-tree to be pruned. However, such a technique can not be applied for mining other types of monotone constraints which are dependent on the underlying database, such as the infrequency constraint which is considered in contrast mining.

The pruning strategy of FP-Bonsai has been adopted by another algorithm, namely *BifoldLeap* [46], which pushes both the monotone and anti-monotone constraints into the leap-traversal algorithm [165], based on the use of FP-trees and COFI-trees [45] for efficient support counting at the expense of memory usage.

2.6 Contrast Pattern Definitions

Contrast patterns capture differences between two sets of objects, or changes in a set of objects between different time periods. For example, when comparing the data between diabetic and non-diabetic patients, contrast characteristics, such as '*aged between 65-85 years old and do not have high blood pressure*' may occur in 60% of the diabetic patients, but only in 1% of the non-diabetic patients.

Emerging patterns [37] are a simple type of contrast pattern. They are item combinations which have sharp differences of frequency between two classes. These patterns are a class of constrained patterns, where the constraints depend on two classes of data.

Contrast mining has been used for a number of bioinformatics applications, such as classifying gene expressions [40], and understanding leukaemia diseases [84]. Work in [87] proposed an efficient technique for mining emerging patterns, by introducing a concept for describing geography of differences between datasets [37]. Other mining techniques, and the useful applications of emerging patterns can be found in [38, 51, 16, 14]. Emerging patterns are defined as follows.

Definition 9. *Consider two classes in a dataset D, namely D_p (the positive class) and D_n (the negative class). The* **growth rate** *[37] of an itemset P is the support ratio between the two classes, i.e. $\frac{support(P,D_p)}{support(P,D_n)}$. The* **discriminating power***, or* **strength** *[42] of P is the ability of P to distinguish the instances in D_p from the instances in D_n, which is a function of its support and its growth rate:*

$$strength(P, D_p, D_n) = support(P, D_p) * \frac{growthrate(P, D_p, D_n)}{1 + growthrate(P, D_p, D_n)}$$

Definition 10. *[37] Given a minimum growth rate ρ, A ρ* **Emerging Pattern** *favouring D_p (i.e. positive class) is an itemset which has a growth rate greater than or equal to ρ.*

The growth rate may be finite or infinite. In the case that the growth rate ρ is infinite, the pattern is called a **Jumping Emerging Pattern (JEP)** [37]. Jumping emerging patterns present only in the positive class but absent in the negative class.

The following definition of emerging pattern further specifies the threshold of the support of the pattern for each class.

Definition 11. *[37] Given two support thresholds α and β, an* **Emerging Pattern (EP)** *for the positive class D_p is an itemset P which satisfies two constraints: i) $support(P, D_p) \geq \alpha$ (i.e. frequent in D_p), and ii) $support(P, D_n) \leq \beta$ (i.e. infrequent in D_n).*

By having a non-zero value of β, greater robustness to noise can be achieved, but the discriminating power of the emerging patterns is sacrificed. By setting $\beta = 0$, constrained emerging patterns reduce to jumping emerging patterns. Moreover, P is a **minimal jumping emerging patterns** if P does not contain other emerging patterns. Minimal jumping emerging patterns are considered the most expressive patterns because their supports are no smaller than their supersets. Due to their high discriminating power, jumping emerging patterns have been successfully applied for building a highly accurate classifier, called the JEP-Classifier [82].

Example 10. *Consider the example data set shown in Figure 2.6. Given $\alpha = 0.25$ and $\beta = 0$, the minimal jumping emerging patterns are ae, ad, i, ch, eh. All their supersets which occur in the positive class are also jumping emerging patterns.*

Emerging patterns are related to class-association rules [91]. An emerging pattern P may correspond to an association rule: $P \rightarrow D_p$, where D_p is the positive class. The rule's confidence can be measured by dividing the total frequency of the emerging pattern in both classes, with the frequency of the pattern in the positive class. That is, confidence$(P \rightarrow D_p) = \frac{count(P,D_p)+count(P,D_n)}{count(P,D_p)}$. The rule's confidence is equal to 1 if $count(P, D_n) = 0$, as the case for all jumping emerging patterns. Therefore, jumping

id	Itemset
t_1	$\{a, e, g\}$
t_2	$\{a, d, i\}$
t_3	$\{b, f, h\}$
t_4	$\{c, e, h\}$

(a) Positive class

id	Itemset
t_1	$\{a, f, g\}$
t_2	$\{b, d, h\}$
t_3	$\{b, f, h\}$
t_4	$\{c, e, g\}$

(b) Negative class

Figure 2.6: A dataset with a positive class and a negative class

emerging patterns are class-association rules which have 100% confidence in the positive class.

A condensed representation of contrast patterns was proposed in [146], by adopting the concept of *closed frequent patterns* to emerging patterns, it finds emerging patterns that do not have same growth rate as their subsets. There is another type of emerging pattern defined using a statistical significance χ^2-test, called the **Chi Emerging Patterns (Chi-EP)** [50]. These patterns can find statistically significant contrasts, and are useful for improving the robustness of an emerging pattern-based classifier, in the presence of noise.

2.7 Review of Contrast Mining Techniques

Work in [37] introduced the concept of *border* for concisely representing emerging patterns, which relies on the interval-closure property.

Definition 12. *A* **border** *is a an ordered pair $< L, R >$ that represents a collection of itemsets. L is called the* left border*, which is the collection of minimal itemsets in S, and R is the* right border*, which is the collection of maximal itemsets in S.*

Definition 13. *A set of itemsets S is* **interval closed** *if there exists L and R such that L and R are in S and for every $X \in S$, $L \subseteq X \subseteq R$ holds.*

A closed interval also corresponds to a subalgebra in constrained pattern mining [28], where the left border corresponds to the bottom itemsets, and the right border corresponds to the upper itemsets. The input data set and the set of emerging patterns are interval-closed. Hence, the set of emerging patterns can be found by manipulating the border of the positive class and the border of the negative class, according to the given support constraints. The borders of the contrast patterns are then found by differentiating the borders between the two datasets, which is called the *Border-Diff* operation[37]. The algorithm is called *Horizon-Miner*, which we outline below.

Given a minimum support α for the positive class, and a maximum support β for the negative class. The right border for the positive class can be found using any of the maximal frequent itemset mining techniques (e.g. [29, 70]) with a minimum support threshold α, and the right border for the negative class contains the maximal frequent itemsets given a minimum support threshold β. The contrast patterns are found by finding itemsets which occur within the positive class border representations, and do not occur within the negative class border representations.

A semi-naive border differential algorithm was introduced in [41] by employing a set-enumeration tree to enumerate subsets of the right border of the positive class, while checking the constraint. It allows different constraints, i.e. growth rate constraint, or minimum/maximum support constraint, to be applied on the emerging patterns. This algorithm performs well on small databases, but it does not scale well for large databases.

A more efficient algorithm for computing Border-Diff was proposed in [16], which is based on the relationship between jumping emerging patterns and minimal hypergraph transversals. The following are formal definitions of hypergraph and its minimal transversal.

Definition 14. *A hypergraph is defined by a set of vertices $V = \{v_1, v_2, \dots v_n\}$ and a set of edges E, where each edge is some subset of V. A* transversal *of a hypergraph is any set of vertices that contains at least one element of every edge. A* minimal transversal *is a transversal such that no proper subset is also a transversal.*

Each transaction t in the positive class D_p can be thought of as inducing a hypergraph with respect to all the transactions in the negative class D_n. The vertex set V corresponds to the elements of t. Hypergraph edges are individually defined by subtracting a transaction in D_n from V. The minimal emerging patterns are the minimal transversals of this set of hypergraphs.

The border-differential technique can be extended for finding constrained emerging patterns, with α and β support thresholds. The hypergraph edges are induced from the right border representations of D_p and D_n, respectively, and the minimal transversals of the resulting hypergraph are guaranteed to have support at least α in D_p and less than β in D_n.

For efficient minimal hypergraph transversals, the work in [16] proposed a recursive-partitioning approach, which grows the emerging patterns incrementally in a similar manner to growing patterns in the pattern-growth approach. Further optimisations to this partitioning approach were introduced in [14]. They use a prefix tree for compactly storing the transactions in D_p and D_n, which allows fewer hypergraph problems to be induced by transactions that share some common prefix.

2.8 Statistically Class-Discriminative Contrast Patterns

There is another approach [160, 139] for finding contrasts that uses a statistical method for exploring the pattern space and for measuring the discriminative power of the candidates, which are called *contrast sets*. This statistical method differs from emerging patterns whose discriminative power is measured based on their support values and the relative support ratio between the classes. A statistical test, such as the χ^2-approximation or the Fisher's exact test, is used for testing whether the association between items within a pattern is statistically significant. Moreover, work in [160] identified the multiple tests problem when numerous rules exist. They proposed a solution by adjusting the critical significance level as the number of rules grows.

Work in [51] also proposed a technique for finding statistically interesting contrasts, based on adopting the χ^2-test for emerging patterns. The difference between the statistically significant emerging patterns and statistically significant contrast sets lies in the function for measuring their discriminative powers. Work on statistical significant rules, however, has a slightly different aim from classification, since significant rules aim to reduce the false-positives rather than the false-negatives [160].

Another class of statistically important contrasts was introduced in [83, 81]. In [81], they studied patterns which correspond to relative risks and odds ratio in a cohort study for studying the risk factors of a disease. In the contrast mining context, a disease is interpreted as a positive class, the risk factors as patterns, and an exposure of an individual to a factor as the occurrence of the pattern in a transaction. Relative risks (RR) are the ratio of the proportions between exposed (to specific a factor) and unexposed individuals. This measurement is similar to the growth rate of an itemset. If an exposed factor is represented by an itemset, the relative risk of factor P is:

$$\begin{aligned} RR &= \frac{count(P,D_p)/|D_p|}{count(P,D_n)/|D_n|} \\ &= \frac{support(P,D_p)}{support(P,D_n)} \\ &= growthrate(P, D_p, D_n) \end{aligned}$$

The odds ratio (OR), for a factor P is the ratio between the odds that a diseased individual has been exposed to the factor and the odds that a healthy individual has been exposed to the factor.

$$OR = \frac{support(P, D_p)/(1 - support(P, D_p))}{support(P, D_n)/(1 - support(P, D_n))}$$

Based on the concept of itemset closure [128], an equivalence class represents a set of itemsets which have the same frequency in the database. Similar to the concept of a border, an equivalence class EC is represented by two sets of itemsets: i) the closed patterns, which correspond to the maximal itemsets in EC, ii) the generators which correspond to the minimal itemsets in EC. All itemsets within an equivalence class share the same level of statistical significance. This technique is useful when a large number of contrast patterns exists, and the closed patterns are much fewer.

The proposed algorithm in [83] finds only the generators and the closed patterns, which are sufficient to represent the equivalence classes. Their algorithm uses hash tables, or FP-trees, for storing the closed patterns and the generators. The FP-trees contain the closed frequent itemsets, and their class frequencies. Once the closed patterns are found, their statistical significance can be measured based on their class frequencies. Additionally, the class-discriminative behaviour of the patterns is constrained by a δ threshold, which specifies the maximum frequency of a pattern in the negative class. When multiple classes exist, one class is chosen as the positive class, and the δ threshold specifies the maximum frequency of the pattern in each of the negative classes.

2.9 Classification Based on Contrast Patterns

The first emerging pattern (EP) based classifier is called Classification by Aggregating Emerging Patterns (CAEP) [42]. Based on their distinguishing class frequencies, emerging patterns favor D_p over D_n, for a chosen positive class D_p. Thus, given a test instance T, if an emerging pattern occurs in T, that pattern makes a *contribution* to classify T as an instance of D_p [42]. To make a classification, all patterns which contain T are found from each class, and their *contribution* are aggregated. Finally, class C will be chosen as the label of T if the aggregated score of patterns which favor C are stronger than those which favor the other class. CAEP chooses the class whose total contrast strength of the contributing patterns is maximum: $class(T) = \max_{score(T,C)} C$,

$$score(T, C) = \sum strength(Q, C, \neg C) \tag{2.1}$$

where Q is an element of $E(C)$, $E(C)$ is the set of emerging patterns of class C, $\neg C$ is the other class, and Q contains T, i.e. $Q \in E(C)$, $T \subseteq Q$.

Apart from support or contrast strength aggregation, other scoring functions may be used by the EP-based classifier, such as a transaction coverage function which is used by association rule classifiers [92]. Emerging patterns can achieve very high accuracy for classifying a two-class data set, especially when there are enough training transactions.

The Jumping Emerging Pattern Classifier (JEP-Classifier) [39] aggregates the contributing minimal JEPs by the sum of their *support* in D_p, instead of their *strength.* It was shown to be highly accurate for classifying large and dense data sets, i.e. when there are many individually frequent items, due to the sharp class-distinguishing characteristics of JEPs. However, the JEP-Classifier may overfit the training data and become noise-intolerant. It may not perform well for classifying extremely imbalanced data sets, since the occurrence of JEPs may be rare.

In the presence of multiple classes, emerging pattern-based classifiers follow a pairwise technique [15]. Given n classes in the data set, they perform $\frac{n(n-1)}{2}$ mining operations, for which each operation finds the emerging patterns for one of the classes and merges the other remaining classes into one negative class. Similar to the CAEP model, for classifying a test instance T, they find the emerging patterns from each class-pair, and then associate each class with a tally of wins. The class that has the most wins is chosen as the label for T.

2.9.1 Noise-tolerant Emerging Pattern Classifier

On noisy data sets, JEP-classifier does not perform well. In such circumstances, there can exist JEPs which occur very rarely in the positive class, and they may correspond to noise. Moreover, the number of JEPs may be very small, due to the strict requirement that the patterns must have zero support in the negative class. To overcome those limitations, Chi Emerging Patterns (Chi EP) [50] were introduced to find large-growth-rate emerging patterns, and eliminate the noisy patterns using a statistical significance test.

Definition 15. *An itemset X is a Chi Emerging Pattern (Chi EP) if all of the following conditions are true:*

1. $support(X) \geq \xi$, *where* ξ *is a minimum support threshold*
2. $growthrate(X) \geq \rho$, *where* ρ *is a minimum growth rate threshold*
3. $\neg\exists Y(Y \subset X) \wedge (support(Y) \geq \xi) \wedge (growthrate(Y) \geq \rho) \wedge (strength(Y) \geq strength(X))$
4. $|X| = 1 \vee |X| > 1 \wedge (\forall Y(Y \subset X \wedge |Y| = |X| - 1) \Rightarrow chi(X,Y) \geq \eta)$, *where* $\eta = 3.84$ *is a minimum chi-value threshold and* $chi(X,Y)$ *is computed using the following contingency table*

	X	Y	$\sum row$
D_1	$count_{D_1}(X)$	$count_{D_1}(Y)$	$count_{D_1}(X) + count_{D_1}(Y)$
D_2	$count_{D_2}(X)$	$count_{D_2}(Y)$	$count_{D_2}(X) + count_{D_2}(Y)$
$\sum column$	$count_{D_2+D_2}(X)$	$count_{D_1+D_2}(Y)$	$count_{D_1+D_2}(X) + count_{D_1+D_2}(Y)$

The first condition ensures a chi-EP is not a noise by imposing a minimum support on the training dataset. The second condition ensures that the pattern has a strong discriminating power. The third condition prefers short patterns which have large strength. And the fourth condition ensures that every item in every chi-EP contributes significantly to the discriminating power of the pattern.

Bayesian Classification by Emerging Patterns (BCEP) [49], was proposed to handle noise in the input data set, based on the Bayes theorem and emerging patterns. Such a Bayesian classifier is inherently noise tolerant due to its collection of class and conditional probabilities. The chosen class C for a given transaction T the BCEP classifier should maximise the following probability function:

$$P(C|T) = \frac{P(T,C)}{P(T)} = P(C)\frac{P(T|C)}{P(T)}$$

where $P(Y|X)$ denotes the conditional probability of Y given X, and the probabilities are estimated from the training data set.

The main weakness of a Bayesian classifier is the assumption that all attributes are independent given the class. This assumption is relaxed in the BCEP classifier by using emerging patterns, since an emerging pattern is a combination of attribute values. The use of Chi Emerging Patterns in the BCEP classifier [50] combines the noise tolerance of the Bayesian approach, and the high quality of the Chi-EP patterns. An empirical study showed that BCEP has a higher accuracy than the other EP-based classifiers, being able to handle different types of noise, such as attribute noise, label noise, and a combination between the two.

2.9.2 Lazy Classification

The classifiers that we have so far discussed perform the mining of emerging patterns from the training data set, and then use those patterns repeatedly to new instances (from the testing data set) for classification. There is a lazy approach, called the DeEPs classifier [82], which uses the new (test) instance as a constraint to extract knowledge useful only for the classification of this instance. The classifier is also termed as *instance-based classifier*.

The strength of the DeEPs classifier is its effective data reduction technique. It uses the new instance as a filter to remove irrelevant training values. It excludes training instances which do not contain any of the items which appear in the test instance, and excludes items which do not appear in the test instance. This results in a sparser training data, which makes the mining of relevant patterns easier. This lazy classification approach mines patterns for each instance in the testing set. It is useful when there is an excessive number of patterns, but it may increase redundancy when the test instances are similar. According to [82], the DeEPs classifier is useful for practical applications, where the data is frequently updated or changing.

2.9.3 PCL Classifier

Work in [84] proposed a pattern-based classifier for studying leukemia disease, based on the *collective likelihood* of emerging patterns. The intuition behind this classifier, namely Prediction by Collective Likelihood (PCL), is that a given test instance should contain strong emerging patterns from its own class and weak emerging patterns from the other class (or classes). This classifier has shown to be useful for classifying gene expression profiles, in which many high dimensional emerging patterns exist. We will give an overview of the classifier shortly.

In many cases, a testing sample contains several emerging patterns from each class. PCL orders the patterns by decreasing frequency. Given a test instance T, it makes its prediction by finding how many top-k emerging patterns from each class are contained in T, and measuring how far away are the contained patterns from the top-k patterns in the entire class. The score for each class is computed as the following function:

$$score(T)_D = \sum_{m=1}^{k} \frac{support(EP_{i_m})}{support(EP_m)}$$

where D may be D_p or D_n, EP_m denote the m-th emerging pattern in D, and EP_{i_m} is the m-th emerging pattern which is contained in T. The maximum score value is 1, for which the most common property of class D is present in T. The score gives the collective likelihood of the top-k emerging patterns which are contained in T to describe the property of class D.

This technique has been shown to be able to outperform the other supervised classification techniques, due to its ability to provide new insight into the correlation between genes, which are discovered by the emerging patterns. However, PCL still has some limitations. Due to the high dimensionality of the data set, PCL performs a feature selection technique prior to mining the emerging patterns, which may hide some dependency between genes, and in turn influence the prediction ability. Moreover, mining emerging patterns requires discrete data, which means, the performance of such a classifier is also influenced by the data discretisation method.

2.10 Summary

In this chapter, we have given a review of the existing techniques in pattern mining. They include the techniques for mining frequent itemsets, frequent subsequences, and mining contrast patterns. The pattern growth is considered as the winning approach for mining a large number of patterns. Tree data structures, or combinations between trees and arrays, are popularly used in many implementations of the pattern growth algorithm. Only two implementations which are based on a graph data structure, and they are so far only useful for mining frequent itemsets. These graph-based algorithms, however, are not competitive compared to the tree-based algorithms.

Chapter 3

Binary Decision Diagrams and Related Applications

This chapter provides related background on Binary Decision Diagrams (BDDs) [7, 25, 26], which are the data structure that underpins the data mining techniques presented in this thesis.

3.1 Overview of Binary Decision Diagrams and Applications

Binary Decision Diagrams (BDDs) [7, 25] were firstly introduced as a directed acyclic graph (DAG) data structure for representing switching functions $f : \mathbf{B}^n \mapsto \mathbf{B}$, which map bit vectors to single bits. Informally, a Binary Decision Diagram is similar to a binary decision tree, except that identical sub-trees are merged, and node fan-in is allowed as well as fan-out. Binary Decision Diagrams (and their variants) have been widely used in the field of VLSI/CAD, such as for logic synthesis [103] and formal verification of digital systems [27, 154], and in the field of reliability engineering for fault-tree analysis [144]. A summary of the various types of the data structures can be found in [148]. We will list several of the variants which are useful for data mining.

Formally, a BDD [25] is a canonical directed acyclic graph (DAG), consisting of one source node, multiple internal nodes, and two sink nodes (also called as terminal nodes) which are labeled as 0 (referring to the 0-terminal node) and 1 (referring to the 1-terminal node). Each internal node may have multiple parent nodes, but it has only two child nodes. The nodes are labeled and ordered such that the parent has a lower index than its children.

In BDD semantics, an internal node N with label x, denoted $node(x, N_1, N_0)$, encodes the boolean formula $N = (x \wedge N_1) \vee (\overline{x} \wedge N_0)$, which follows the *Shannon decomposition* [26]. Given a Boolean function f, the Shannon decomposition is based on the two partial evaluations of f with respect to one of its variables:

$$f = xf_x + \overline{x}f_{\overline{x}} \tag{3.1}$$

where x is a variable, f_x is a partial evaluation of f when $x = 1$, and $f_{\overline{x}}$ is a partial evaluation of f when $x = 0$. N_1 and N_0 are referred to as the *true*-child and *false*-child of N. The edge connecting node N to N_1 (resp. N_0) is also called the *true*-edge (resp. *false*-edge) [1]. Each path from the root to sink-1 (resp. sink-0) gives a true (resp. false) output of the represented function.

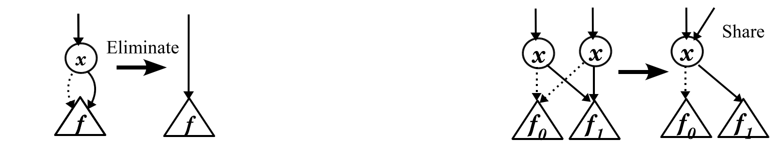
A class of BDDs, called Ordered Binary Decision Diagrams (OBDDs) [26], are canonical graph representations of Boolean functions. Suppose there is a variable indexing function π which determines the variable ordering in the OBDD. If an internal node N_y is a child of N (either be a *true*-child or a *false* child), and node N_y is labeled with variable y, then $\pi(x) < \pi(y)$. Moreover, an OBDD is called a Reduced Ordered Binary Decision Diagram (ROBDD) [26] if it employs the following two reduction rules:

- **Elimination rule**: If both outgoing edges of a node v point to the same node u, then eliminate v and redirect all of its incoming edges to u (Figure 3.1(a)).
- **Merging rule**: If nodes u and v are identical, then eliminate one of the two nodes, and redirect all incoming edges of the deleted node to the remaining one (Figure 3.1(b)).

ROBDDs are useful for compactly representing Boolean functions. Efficient manipulations of ROBDDs are obtained based on the following two important properties:

- **Canonical**: Equivalent subtrees are shared and redundant nodes are eliminated.
- **Caching of computation results:** Intermediate computation results are stored for future re-use.

[1] In their illustrations, the true-edges are shown as solid lines, the false edges are shown as dotted lines



(a) Elimination rule

(b) Merging rule

Figure 3.1: ROBDD Reduction Rules ($f_0 = f_{\overline{x}}$; $f_1 = f_x$)

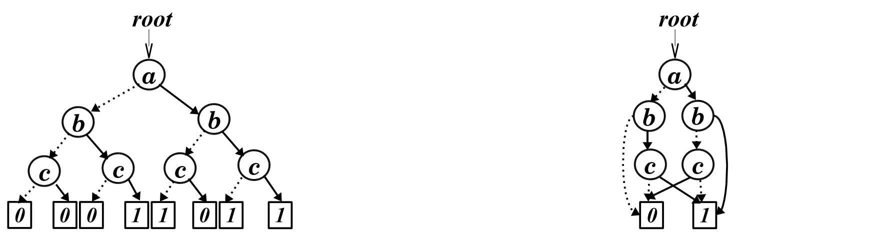
The effect of these principles is that a boolean formula can be represented with high compression, i.e. for an n variable formula, the possible space of truth values is 2^n, however the corresponding BDD can have exponentially fewer nodes. An optimal utilisation of BDD can be achieved by having as much node sharing as possible in the BDD.

ROBDDs have been shown to be able to achieve high data compression and efficient computation time. Their canonical property allows logical operations such as AND, OR, XOR, etc. to be performed in polynomial time with respect to the number of nodes. In recent ROBDD application, ROBDDs are often referred to as BDDs (a survey can be found in [107]). Consider an example of a BDD in Example 11.

Example 11. *Let F be a boolean formula such that $F = (a \wedge b \wedge \overline{c}) \vee (a \wedge b \wedge c) \vee (a \wedge \overline{b} \wedge \overline{c}) \vee (\overline{a} \wedge b \wedge c)$. Figure 3.2(a) shows an example of a binary decision tree for F, where each branch represents an assignment for its variables. A solid line represents a TRUE value of the variable, and a dotted line represents a FALSE value of the variable. Each node at the lowest level represents the output of the function,* 1 *means TRUE,* 0 *means FALSE. An example of a BDD representation for F, with the same variable ordering as the binary decision tree, is shown in Figure 3.2(b). It shows that the BDD representation is 8 nodes smaller than the binary decision tree.*

Due to the vast number of BDD applications, there exist many variants of OBDDs, which employ different rules from the reduction rules for ROBDDs. Other BDD-inspired data structures have been introduced for data mining applications, such as Reduced Ordered Decision Graphs (RODGs) [122], which are compact representations of decision trees, and useful for inducing classification rules. For text mining applications, work in [36] introduced Directed Acyclic Word Graphs (DAWGs), which are DAG database representations for mining substrings.

A number of efficient BDD packages have been implemented with the following useful properties:



(a) Binary Decision Tree (b) Binary Decision Diagram

Figure 3.2: Example of a Binary Decision Tree and a Binary Decision Diagram for boolean formula $F = (a \wedge b \wedge \overline{c}) \vee (a \wedge b \wedge c) \vee (a \wedge \overline{b} \wedge \overline{c}) \vee (\overline{a} \wedge b \wedge c)$

- Able to generate BDDs for large-scale functions
- Able to check the equivalence of two functions, i.e. that two BDDs are identical, in constant time
- Able to carry out logic operations within a time that is almost proportional to the size of the BDDs

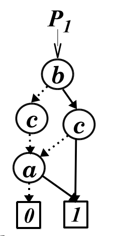
In order to achieve their efficiency, though, the variable ordering plays an important role. We will discuss how it affects the compactness of BDDs shortly.

3.1.1 Effect of Variable Ordering on the Compactness of a BDD

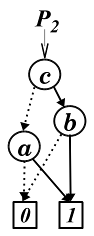
Depending on the function being represented, the number of nodes in a BDD may be highly sensitive to its variable ordering. A good variable ordering for a compact BDD has two properties [58]:

- Groups of inputs that are closely related should be kept near to each other
- Inputs that greatly affect the function should be located at higher positions in the structure

The problem of finding the optimal variable ordering for BDDs is NP-complete [149]. There exist many works that investigate the variable ordering. One approach is to find the appropriate ordering before generating a BDD, based on heuristics [57, 8, 141]. Another approach is to start with an initial ordering and permute the variables as the BDD is constructed [138]. The latter approach is usually more effective than the former but it consumes much more of the computation time.



(a) Var.ordering $\pi_1 = \{b, c, a\}$



(b) Var.ordering $\pi_2 = \{c, b, a\}$

Figure 3.3: Binary decision diagrams for a boolean formula $F = (a \wedge b \wedge \overline{c}) \vee (a \wedge b \wedge c) \vee (a \wedge \overline{b} \wedge \overline{c}) \vee (\overline{a} \wedge b \wedge c)$ with alternative variable orderings

Example 12. *Recall the binary decision diagram for the boolean formula* $F = (a \wedge b \wedge \overline{c}) \vee (a \wedge b \wedge c) \vee (a \wedge \overline{b} \wedge \overline{c}) \vee (\overline{a} \wedge b \wedge c)$ *in Figure 3.2b. Suppose two alternative variable orderings are given, labeled* π_1*, and* π_2*, such that* $\pi_1 = \{b, c, a\}$*,* $\pi_2 = \{c, b, a\}$*. The binary decision diagrams under the two variable orderings are shown in Figure 3.3, which are labeled* P_1 *and* P_2*, respectively. It shows that* P_2 *is one node smaller than* P_1*, showing that variable ordering* π_2 *is more suitable for this boolean formula.*

3.1.2 Canonical Property of BDDs

Since multiple identical nodes are not allowed, BDDs are canonical. Moreover, a set of BDDs for representing multiple functions may be united into a single graph, called Shared Binary Decision Diagrams [115], by merging identical nodes across the BDDs.

The canonical property of BDDs is maintained by storing the unique nodes in a hash table called the *uniquetable.* Each entry in this table is a pair <*key,node*>. *node* is a BDD node, and the *key* is an integer value, which is a function of the node's label and the *keys* of its child nodes. Prior to creating a new node, the uniquetable is checked whether the node exists. If it does not exist, then the node is inserted into the table. Otherwise, a pointer to the pre-existing node is returned instead. The time complexity for creating a BDD node, therefore, depends on the access time of the *uniquetable.*

There exist a number of BDD packages that have been developed for various applications. CUDD [145] is one of the first packages that is widely used for developing many VLSI/CAD applications. The more recently developed package, JINC [145, 123], is object-oriented and has a library for symbolic methods. They allow fast uniquetable access, with $O(1)$ time complexity. JINC uses a hash table with open chaining. That is, nodes with the same hash key are stored as a sorted linked list. The lookup operation requires a single traversal of that list, which has an average size of $\frac{n}{m}$, where n is the number of nodes and m is the size of the table. Their implementation keeps $\frac{n}{m}$ to a small value, and allows $O(1)$ time complexity for an insertion or lookup operation.

3.1.3 Caching Principle of BDD Operations

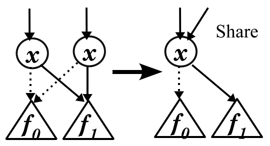
Each BDD primitive operation is associated with a cache, which is also called as *computation table*, that maps the input parameters to the output of each computation. The cached output may be re-used if the same intermediate operation is re-visited, avoiding redundant computations. This caching ability makes the complexity of most BDD perations polynomial with respect to the number of nodes. This principle is particularly effective if many subtrees are being shared within the input BDD, since the same subtree may be encountered multiple times throughout the intermediate computations.

The caching library in JINC [123] implements the cache as a hash table. Each entry in this table is a pair of $\langle input, output \rangle$, where the *input* uniquely identifies the input parameters of the corresponding operation. If two operations have the same key for their input parameters, then the pre-existed cache entry is replaced by the newer entry.

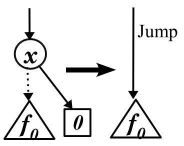
3.2 Zero-suppressed Binary Decision Diagrams for Representing Sets of Itemsets

This section shows that BDDs can be used for efficiently representing sets of itemsets, using their Zero-suppressed Binary Decision Diagrams (ZBDDs) variant [106]. ZBDDs are suitable for representing a large-scale itemset data [111], which makes them potentially useful for our data mining tasks. Different from ROBDDs, however, different reduction rules are employed by ZBDDs. We will firstly describe the Boolean function representation of a set of itemsets, followed by the formal definition of ZBDDs.

A collection of itemsets can be mapped into a Boolean space [106]. Given a domain of n items, a set of itemsets can be represented as a Boolean function by using n input boolean variables for each bit in the itemset. The output value, 1 or 0, expresses whether each item-combination specified by the input variables are included in the set or not. Formally, an itemset p is represented by a n-bit binary vector $\mathcal{X} = (x_1, x_2, \ldots, x_n)$, here $x_i = 1$ if item i is contained in p. The characteristic function for a set S is the function $\mathcal{X}_S : \{0,1\}^n \mapsto \{0,1\}$, where $\mathcal{X}_S(p) = 1$ if $p \in S$, and $\mathcal{X}_S(p) = 0$ otherwise.



(a) Merging rule



(b) Zero-suppression rule

Figure 3.4: ZBDD Reduction Rules

In ZBDD semantics, an internal node $N = (x, N_1, N_0)$ represents a set S of itemsets such that $S = (\{x\} \times S_1) \cup S_0$, where S_1 and S_0 are the sets of itemsets encoded by N_1 and N_0, respectively. An itemset p in S is interpreted as a conjunction of the items contained in p, and yields a true assignment for the boolean formula encoded by N. Thus, each path from the root node to the sink-1 node represents an itemset in S. A ZBDD consisting of only the sink-0 node encodes the empty set ($\emptyset$), and a ZBDD consisting of only the sink-1 node encodes the set of empty itemsets ($\{\emptyset\}$).

Similar to the Ordered Binary Decision Diagrams, Zero-suppressed Binary Decision Diagrams are canonical and their variables are ordered. Different from ROBDDs, however, ZBDDs employ the following two rules (see their illustrations in Figure 3.4):

1. **Merging rule**: share all equivalent subtrees (to obtain canonicity)
2. **Zero-suppression rule**: eliminate all nodes whose true-edge points to the sink-0 node, and bypass the incoming links to the node's 0-child

The zero-suppression rule is effective, since the characteristic functions represented by ZBDDs are monotonic boolean functions, i.e. do not contain negated variables or terms. Thus, negative variables are not necessary and their corresponding nodes may be eliminated.

Basic set operations for ZBDDs which will be used in our algorithm are listed in Table 3.1. These include *set-union ($A \cup B$), set-subtraction or set-difference ($A \backslash B$), and set-intersection ($A \cap B$)*, which have been defined in [106, 118]. They have polynomial time complexity in the number of nodes in the input ZBDD(s).

Example:

1. Given $P = \{\{a, b, d\}, \{b, c\}\},\ Q = \{\{b, c, d\}, \{a, c, d\}\}$
   $P \cap Q = \{\{\}\}$
2. Given $P = \{\{a, b, d\}, \{b, c\}\},\ Q = \{\{b, c, d\}, \{a, c, d\}\}$
   $P \cup Q = \{\{a, b, d\}, \{b, c\}, \{b, c, d\}, \{a, c, d\}\}$

Table 3.1: Primitive ZBDD operations, P and Q are two ZBDDs

0	The 0-terminal node; an empty set, i.e. $\emptyset$
1	The 1-terminal node; a set of an empty itemset, i.e. $\{\emptyset\}$
P.change(v)	Invert all occurrences of item v in P
$P \cap Q$	Set-intersection: itemsets which are elements of both P and Q
$P \cup Q$	Set-union: itemsets which are elements of either P and Q
$P - Q$	Set-subtraction: itemsets which are elements of P but not Q

3. Given $P = \{\{b,c\},\{a,c,d\},\{a,b,d\}\}$, $Q = \{\{a,b,d\},\{a,c\}\}$
   $P \setminus Q = \{\{b,c\},\{a,c,d\}\}$

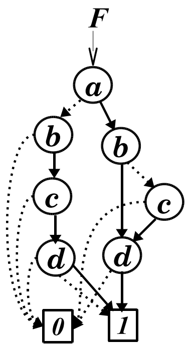
Similar to BDD library routines, ZBDD library routines also perform recursive task decompositions. For example, the traversal procedure for counting the number of nodes in a ZBDD P, i.e. count(P), recursively decomposes the operation into the child-nodes of P. The recursion terminates when it finds a sink node. The procedure is shown below.

```
count($P$){
    if ($P == 1$): return 1
    if ($P == 0$): return 0
    if ($P == (x, P_1, P_0)$): return count($P_1$) + count($P_0$)))
}
```

With the use of ZBDD's cache, the procedure count(P) visits each node in P only once, even though a node may be part of multiple branches. Should the same sub-graph be visited from different branches, the pre-computed result that exists in the computation table may be re-used. The time complexity of this traversal operation is $O(|P|)$, assuming the cache lookup operation takes a constant time, where $|P|$ denotes the number of nodes in P.

Example 13. *The boolean function representation, and the ZBDD encoding for a set of itemsets:* $\{\{a,b,d\},\{b,c\},\{b,c,d\},\{a,c,d\}\}$ *are shown in Figure 3.5 (assume lexicographic variable ordering). This set can also be expressed as a DNF formula:* $F = (a \wedge b \wedge d) \vee (b \wedge c) \vee (b \wedge c \wedge d) \vee (a \wedge c \wedge d)$

Due to its ability to efficiently manipulate sparse combinations [111], ZBDDs are potentially useful for mining patterns in sparse high dimensional datasets. ZBDDs are popularly used for solving boolean satisfiability problems [30, 9], and in the field of



a	b	c	d	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Figure 3.5: A ZBDD representation for a set of itemsets $\{\{a,b,d\}, \{b,c\}, \{b,c,d\}, \{a,c,d\}\}$, and the truth table for its characteristic function $F = (a \wedge b \wedge d) \vee (b \wedge c) \vee (b \wedge c \wedge d) \vee (a \wedge c \wedge d)$

reliability engineering for fault-tree analysis [134]. A survey on their applications can be found in [108]. However, they have received relatively little attention thus far in data mining. We will give an overview of their data mining applications in the following section.

3.3 Zero-suppressed Binary Decision Diagram Applications in Data Mining

In existing data mining applications [109, 112, 80], ZBDDs have been shown to be useful for representing and manipulating the input databases, and for storing the output frequent patterns. Work in [117] shows that using ZBDDs for maintaining the output patterns can improve the LCM [151] algorithm for mining frequent itemsets. Work in [116] shows symmetric itemset mining using ZBDDs. Furthermore, work in [109, 112] shows that ZBDDs are useful for post-processing operations on the patterns, such as pattern matching, and extracting length-k patterns.

3.3.1 ZBDD-growth

The technique proposed in [112], called ZBDD-growth, is based on the FP-growth framework [70], which uses a vector of ZBDDs called *ZBDD-Histogram*, instead of an FP-tree database representation. The algorithm can be used for finding frequent itemsets, as well as their maximal and closed variants. We will give an overview of the ZBDD-growth algorithm.

When representing a set of itemsets as a ZBDD using its characteristic function, which maps each itemset to a binary value of 1 (if it exists in the database) or 0 (otherwise). Thus, the frequency information of the itemsets are not represented in the data structure. Due to this limitation, ZBDD-growth represents the database using a ZBDD-histogram, which is a vector of ZBDDs. A histogram $\{F_{m-1}, \ldots F_1, F_0\}$, represents itemsets whose frequencies are up to $(2^m - 1)$, where F_i is a ZBDD, $0 \leq i \leq (m-1)$. The i-th digit of the ZBDD vector, i.e. F_i, represents itemsets whose i-th bit of their (binary-coded) frequency values is one. Thus, F_0 represents a set of itemsets that appear at odd times (i.e. Least-significant-bit = 1), F_1 represents a set of itemsets whose second-lowest-bit of their binary-coded frequency values is one, and so on.

Let n be the maximum frequency of the itemsets, $\lceil log_2(n) \rceil$, and $bit(n, i)$ be the i-th bit of a given integer n. For a database with size n, a ZBDD-histogram is a vector of $\lceil log_2(n) \rceil$ ZBDDs. For an itemset p with frequency x, it is represented in the k-th ZBDD where $bit(x, k) = 1$. In the worst case, an itemset is replicated in each of the ZBDDs if its frequency is $2^n - 1$.

Example 14. *For example, to represent a set of itemsets (with their corresponding frequency) $\{a(3), b(2), c(3), d(1), ac(2), ad(1), bc(2), bd(1), cd(1),\ acd(1), bcd(1)\}$, the ZBDD-based database encodes each frequency value into binary: 3=11, 2=10, 1=01. Since the frequency values are encoded by 2 bits, 2 ZBDDs are constructed which are shown in Figure 3.6. The i-th ZBDD is labeled by F_i. Itemsets a and c occur in both F_1 and F_0, since they have a support of* 3*; itemsets b, ac, and bc occur only in F_1, since they have a support of* 2*, and itemsets d, ad, bd, and cd, occur only in F_0, since they have a support of 1.*

More detailed explanation of the ZBDD-growth algorithm follows. The frequent itemsets are found by performing a depth-first search over the ZBDD-histogram of the input database. This is performed based on the use of ZBDD primitive routines. It firstly chooses an item v from the ZBDD-histogram, and performs the following two subprocedures:

Itemset	Frequency	F_1	F_0
a	3	1	1
b	2	1	0
c	3	1	1
ac	2	1	0
ad	1	0	1
bc	2	1	0
bd	1	0	1
cd	1	0	1
acd	1	0	1
bcd	1	0	1

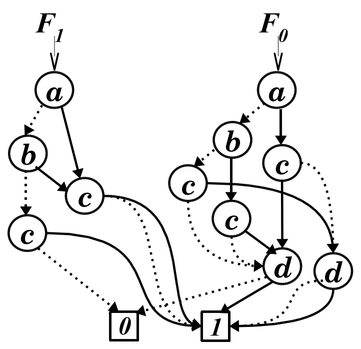


Figure 3.6: A ZBDD-histogram [112] that represents a set of itemsets $\{bc(2),\ bcd(1),\ cd(1), acd(1), ac(2), ad(1), bd(1), a(3), b(2), c(3), d(1)\}$

1. Find the v-conditional database and grow patterns which include v
2. Remove v from the database and grow patterns which do not include v

For efficiency purposes, the chosen item v is the top item that occurs in the ZBDDs in the histogram.

Based on ZBDD semantics, given v is the top item in a histogram H, H can be decomposed into the following set operation: $H = (\{v\} \times H_1) \cup H_0$, where H_1 and H_0 are the 1-child and the 0-child of the highest ZBDD-node in histogram H. Thus, the histogram which represents the v-conditional database can be obtained just by referring to H_1 and H_0, taking a constant time to compute.

ZBDD-growth makes use of ZBDD's caching principle by storing the result of each recursive call (i.e. the patterns found from each conditional database). Each entry in the cache is a (H, F)-pair, where H is the ZBDD-histogram, which represents a vector of ZBDDs, and F is the ZBDD which represents the patterns. This caching mechanism avoids duplicate processing of the same histogram, and allows the overall mining time to be almost linear to the total sizes of the ZBDDs.

3.3.2 Performance Analysis of ZBDD-growth

A ZBDD-histogram allows the support information of each pattern candidate to be represented, but it has limited efficiency when the database is large and sparse, since not many nodes can be shared, although node sharing is allowed across the ZBDDs. Moreover, performing the conditional database projections can be computationally expensive, since each operation on this histogram requires a series of operations on every ZBDD within the vector.

The results shown in [113] indicate that ZBDD-growth can achieve an exponential speed-up over FP-growth when mining frequent itemsets with a low support threshold, especially in scenarios where many ZBDD nodes are shared. For higher support threshold values, ZBDD-growth is less competitive, since a smaller number of patterns exist and the mining procedure does not require many recursive calls. Moreover, high support values also result in larger ZBDD-histograms, since $O(log_2(N+1))$ ZBDDs are required to represent itemsets with frequency N.

The performance of ZBDD-growth depends on the compactness of the ZBDDs, which is influenced by the following factors:

- The similarity between the input itemsets
- The minimum support threshold

Due to those factors, ZBDD-growth does not perform very well when the support threshold is high or in sparse data sets, since in such scenarios, not many nodes are shared in the ZBDDs, and not many cache entries are re-used.

Recent work in [76] adopts the features of the BDD's optimal variable ordering [58] for ZBDDs. They proposed heuristics are based on a dynamic weight assignment method [115] for measuring the influence of each item to the number of patterns. Their heuristics locate the more influential item higher in the structure. A theoretical study of the variable orderings for ZBDDs was presented in [110].

3.3.3 Pattern Indexing and Post Processing

Recent work in [117] shows the advantage of using ZBDDs to provide efficient indexing of large-scale frequent itemsets, over an existing frequent itemset mining algorithm namely LCM (Linear-time Closed itemset Miner) [151].

Moreover, work in [113] shows that a ZBDD is useful not only for finding the frequent itemsets, but also useful for performing post-processing operations on the output patterns for further database analysis. Such operations include finding subsets or supersets of particular itemsets, and finding itemsets of particular sizes. The efficiency of such post-processing operations is also linear with respect to the size of the ZBDD containing the patterns.

Based on the mapping function between a set of itemsets and a Boolean function, alternative Boolean function representations may be used for extracting hidden patterns, or providing alternative and more succinct pattern representations. For instance, finding simple disjoint decompositions of a Boolean function [109] is useful for identifying and merging common sub-patterns from a set of patterns; finding symmetries within a Boolean function [116] can identify items which are interchangeable within a set of patterns, which may be useful for finding item associations.

3.4 Binary Decision Diagrams for Pseudo-boolean Functions

As mentioned earlier, the ZBDD representations of itemsets are not able to represent the frequency values efficiently. Its characteristic function may be modified though, such that each item-combination is mapped to an integer value which represents its frequency. Such a function is called as a pseudo-boolean function. Given a vector of binary-valued variables as input, it represents an integer-valued function.

Many variants of BDDs have been developed for representing pseudo-boolean functions for performing algebraic computations in circuit verification. We will give an overview of two such BDDs, namely Multi-terminal binary decision diagrams (MTBDDs) [32] and Edge-Valued Binary Decision Diagrams (EVBDDs) [154], which serve to inspire our work.

3.4.1 Multi-terminal Binary Decision Diagrams

Multi-terminal binary decision diagrams (MTBDDs) [32] are generalisations of BDDs, which were introduced for manipulating sparse matrices. MTBDDs are able to represent not only binary functions, but finite set functions $f : \mathbf{B}^n \mapsto \tilde{R}$, where $\tilde{R}$ is the finite set of integers. MTBDDs allow multiple constant-valued terminal nodes, instead of only 1 and 0 terminal nodes. Moreover, MTBDDs may be used for representing more general functions from any finite space $\tilde{D} \mapsto \tilde{R}$, where D may be the finite set of integers $\{0, \ldots m-1\}$, for which $f : \tilde{D} \mapsto \tilde{R}$ is a vector, or the finite set $\{0, \ldots, m-1\} \times \{0, \ldots, n-1\}$, for which $f : \tilde{D} \mapsto \tilde{R}$ is a matrix.

More formally, a vector v of length m is represented using $\lceil \lg m \rceil$ bits. The vector thus becomes a function from the Boolean space $\mathbf{B}^{\lceil \lg m \rceil}$ onto the range of the vector,

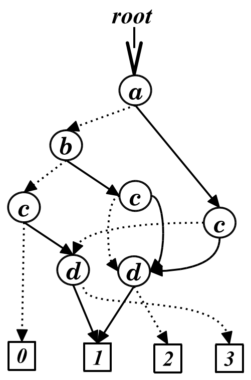


Figure 3.7: A Multi-valued Binary Decision Diagram representation of a set of itemsets $\{bc(2), bcd(1), cd(1), acd(1), ac(2), ad(1),\ bd(1), a(3),\ b(2),\ c(3), d(1)\}$

and can be represented as an MTBDD. To represent a matrix that has a dimension $m \times n$, $\lceil \lg m \rceil$ bits are used to represent the row indices, and $\lceil \lg n \rceil$ bits are used to represent the column indices. In our pattern mining context, we may use a function to map from each itemset to its frequency value, $f : \mathbf{B}^n \mapsto \tilde{R}$, and represent it as an MTBDD. The trade-offs of this representation are the existence of many terminal nodes for representing a large dataset, and the low level of node sharing among itemsets which have different frequencies.

Example 15. *Figure 3.7 shows an example of the multi-terminal valued binary decision diagrams for the set of itemsets $\{bc(2), bcd(1), cd(1), acd(1), ac(2), ad(1),\ bd(1), a(3), b(2), c(3), d(1)\}$, whose ZBDD-Histogram representation was shown in Figure 3.6.*

3.4.2 Edge-Valued Binary Decision Diagrams

Edge-Valued Binary Decision Diagrams (EVBDDs) [154] are able to represent pseudo-boolean functions as MTBDDs, without using numerous terminal nodes. EVBDDs only contain two terminal nodes, 0 and 1. Instead of representing the output values as terminal nodes, EVBDDs distribute the output values into the sub-graphs by using weighted edges. The weights are additive, which are computed based on a recursive decomposition of the represented function.

An EVBDD node $\mathbf{v}$ represents an arithmetic function f by using the following function decomposition:

$$f = x \times (value + f_l) + (1 - x) \times f_r + c \tag{3.2}$$

where x is an input variable, c is a constant, and f_l and f_r are two functions. Based on the above function decomposition, each node in EVBDDs is represented by a tuple $\langle c, \mathbf{f} \rangle$, where c is a constant and $\mathbf{f}$ is a directed acyclic graph consisting of the following types of nodes:

- The single terminal node, denoted by 0, which represents the function $f = 0$
- A non-terminal node which is described by a 4-tuple $\langle v, \mathit{child}_l(\mathbf{v}), \mathit{child}_r(\mathbf{v}), \mathit{value} \rangle$, where v is an input variable, i.e. $v \in \{x_0, \ldots x_{n-1}\}$, and $\mathit{child}_l(\mathbf{v})$ and $\mathit{child}_r(\mathbf{v})$ are EVBDDs that represent the sub-expressions f_l and f_r, respectively.

An EVBDD is *ordered* if the variables are ordered, similar to that in Ordered Binary Decision Diagrams. Moreover, it is *reduced* if both conditions are true:

- There is no non-terminal node $\mathbf{v}$ such that $\mathit{child}_l(\mathbf{v}) = \mathit{child}_r(\mathbf{v})$ with $\mathit{value} = 0$
- There are no two nodes $\mathbf{u}$ and $\mathbf{v}$ such that $\mathbf{u} = \mathbf{v}$.

Example 16. *Consider function $f(x_0, x_1, x_2) = 3 + 2x_0 - 7x_0x_1 - 5x_0x_2 + 6x_0x_1x_2 + 3x_1 - 5x_1x_2$. Below are possible decompositions of this function:*

$$f(x_0, x_1, x_2) = 1 + x_0(4 - 4x_1 - 5x_2 + x_1x_2) + (1 - x_0)(2 + 3x_1 - 5x_1x_2) \tag{3.3}$$

$$f(x_0, x_1, x_2) = 8 + x_0(-3 - 4x_1 - 5x_2 + x_1x_2) + (1 - x_0)(-5 + 3x_1 - 5x_1x_2) \tag{3.4}$$

$$f(x_0, x_1, x_2) = 3 + x_0(2 - 4x_1 - 5x_2 + x_1x_2) + (1 - x_0)(3x_1 - 5x_1x_2) \tag{3.5}$$

Figure 3.8 shows an example of the reduced ordered EVBDD representation of f, which uses the third decomposition (Equation 3.5). The value of each node is shown as a weight on its incoming edge. The function is fully decomposed into:

$$f(x_0, x_1, x_2) = 3 + x_0(2 + x_1(-4 + x_2(-4)) + (1 - x_1)(x_2(-5))) + (1 - x_0)(x_1(3 + x_2(-5)))$$

Using EVBDDs for representing itemsets is possible. However, obtaining the frequency of an itemset requires a traversal of the data structure to find the itemset and recursively compute its frequency, which can be expensive to compute. For efficient

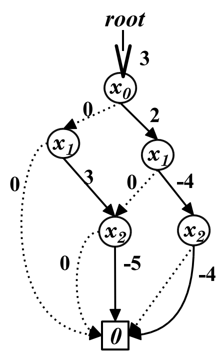


Figure 3.8: An Edge-Valued Binary Decision Diagram representation of $3 + 2x_0 - 7x_0x_1 - 5x_0x_2 + 6x_0x_1x_2 + 3x_1 - 5x_1x_2$

pattern mining, where frequency counting is a core computation, we need to be able to efficiently obtain the frequency of an itemset and the total frequency given a set of itemsets.

3.5 Summary

We have given an overview of Binary Decision Diagrams, Zero-suppressed Binary Decision Diagrams, and related BDD variants namely Multi-terminal Binary Decision Diagrams and Edge-Valued Binary Decision Diagrams. Multi Valued Binary Decision Diagrams and Edge-Valued Binary Decision Diagrams, can represent numerical values, but their manipulations can be expensive for representing large scale functions. Zero-suppressed Binary Decision Diagrams are potentially the most useful data structures for solving our pattern mining problems, but, they cannot straightforwardly represent the frequency information of the patterns.

Chapter 4

Efficient Mining of High-Dimensional Frequent Itemsets Using Weighted Zero-suppressed Binary Decision Diagrams

Mining frequent patterns [70], such as frequent itemsets, is a fundamental and well studied problem in data mining. Frequent itemsets correspond to combinations of items (or attribute values) which occur frequently in the data set. Mining them in high dimensional data can be costly in terms of both time and space, since the number of item combinations is exponential in the number of dimensions and a large amount of patterns may exist. To address this issue, we propose an original variant of Zero-suppressed Binary Decision Diagrams (ZBDDs) [106], called **Weighted Zero-suppressed Binary Decision Diagrams**, and propose efficient algorithms using them as a primary data structure. The purpose of this chapter is to analyse the behaviour of frequent itemset mining in high dimensional datasets, which exist in the bioinformatics domain.

4.1 Introduction

State-of-the-art techniques, such as those found in the Frequent Itemset Mining Implementation (FIMI) Repository [78, 62], have made attempts to address the challenges in frequent itemset mining. Many of them are based on the use of prefix tree data structures, or combinations of prefix trees with other data structures, to compress the data representation. An optimised FP-growth implementation [64] was one of the winning techniques and used FP (frequent pattern)-trees [70]. A review of the FP-growth technique was given in Chapter 2.

Zero-suppressed Binary Decision Diagrams (ZBDDs) are useful for mining frequent patterns [112]. There exist several works [109, 116, 112, 117] which studied the use of ZBDDs for mining frequent itemsets. We identified that the key features of ZBDDs allow efficient manipulation of a large scale of data. In FP-trees, identical sub-trees may exist, which may be merged and shared across multiple ZBDDs. By the sharing of identical sub-trees, higher data compression can potentially be achieved, which is one of the key features of ZBDDs. Furthermore, there exist efficient ZBDD library routines which can be employed, which are attractive for mining high dimensional frequent itemsets efficiently.

On the other hand, ZBDDs have a limited ability to represent the frequency values of the itemsets. To address this issue, work in [112] proposed to use a vector of ZBDDs, called ZBDD-histogram. A review of their technique was given in Chapter 3. In such a representation, however, an itemset may be duplicated across multiple ZBDDs within the histogram, which may limit the time and space efficiency when mining a large scale of patterns in the high dimensional data sets. Work in [117] uses the ZBDD-histogram as data structure for mining the closed frequent itemsets based on the LCM framework [151]. Their technique employs the ZBDDs for storing and maintaining the output patterns. The aim of our research, however, is to investigate whether BDD manipulations can be integrated in the mining procedure, for manipulating the input database, and searching for pattern candidates.

In this chapter, we introduce an original variant of BDDs, called the **Weighted Zero-suppressed Binary Decision Diagrams (Weighted ZBDDs)**, whose edges are weighted to represent the frequency of each itemset in the structure. The weighted edges in Weighted Zero-suppressed Binary Decision Diagrams are designed to fit nicely in the depth-first search framework of frequent pattern mining, which allows efficient retrieval of frequency through out the recursive database projections. There exist other weighted variant of BDDs, such as the Edge-Valued Binary Decision Diagrams [154] which were proposed for efficient representation of discrete functions, but their weight function is not suitable for our mining task.

4.1.1 Objective and Contributions

The main objective of this chapter is to identify and explain situations where ZBDDs are advantageous compared to FP-trees. In particular, we aim to address questions such as:

1. Does the canonical property of ZBDDs allow a scalable and efficient algorithm for frequent itemset mining to be developed?
2. How much data compression can a ZBDD achieve compared to an FP-tree?
3. Does the use of a more compact data structure always mean that mining is more efficient?

Our main contributions in this paper are three-fold:

- We present an algorithm that can mine frequent itemsets and their maximal/closed variants, based on the use of a ZBDD as the primary data structure. A supplementary bitmap is used for support checking (similar to the MAFIA algorithm [29] reviewed in Chapter 2). A particularly attractive feature of our technique is the use of multiple shared-ZBDDs to represent the input database, the intermediate databases, as well as the final output, allowing them to share common sub-trees. This feature is something which is not possible in prefix-tree-based techniques like [64, 94, 132]. We also show how our mining technique can be adapted to the row-wise mining approach [136, 125, 124], which has been an alternative solution for mining high dimensional itemsets.
- We introduce Weighted Zero-suppressed Binary Decision Diagrams. The edge weights allow itemsets and their corresponding frequencies to be compactly represented. Hence, support counting can be performed more efficiently compared to when the bitmap is used. Moreover, their canonical property allows a more efficient mining technique to be developed, which is achieved by re-using intermediate results. It is advantageous especially for mining large and dense data sets, in which bitmap manipulations may be costly.

- We experimentally investigate the behaviour of our techniques, according to various characteristics of high dimensional biological data sets. Our techniques are compared against the competitive FP-growth implementation, FP-growth* [64], and LCM over ZBDD (LCM-ZBDD) [117]. Our results show a number of situations where the use of a ZBDD (either weighted or non-weighted) is able to give significant improvements over FP-growth* or LCM-ZBDD. Our techniques are also compared against the CARPENTER algorithm [125], which represent the algorithms in the row-wise mining framework. We find that our techniques are performing efficiently for mining at higher supports in this row-wise mining.

4.2 Frequent Itemset Mining Algorithms Using ZBDDs

In this section, we firstly introduce our algorithm for mining frequent itemsets which is based on the use of Zero-suppressed Binary Decision Diagrams, namely **FIMiner**, then show how it can be used for finding maximal/closed frequent itemsets. Table 4.1 lists some pre-defined ZBDD library operations which are used in our algorithms. We use slightly different notations from that in previous work [111, 118] though.

Example:

1. Given $P = \{\{a,b,d\},\{b,c\}\},\ Q = \{\{b,c,d\},\{a,c,d\}\}$
   $P \cap_Z Q = \{\{\}\}$
   $P \cup_Z Q = \{\{a,b,d\},\{b,c\},\{b,c,d\},\{a,c,d\}\}$
   $P \cup_{Z_{max}} Q = \{\{a,b,d\},\{b,c,d\},\{a,c,d\}\}$
   $P \cup_{Z_{min}} Q = \{\{a,b,d\},\{b,c\},\{a,c,d\}\}$

2. Given $P = \{\{b,c\},\{a,c,d\},\{a,b,d\}\},\ Q = \{\{a,b,d\},\{a,b,c\}\}$
   $P \setminus Q = \{\{b,c\},\{a,c,d\}\}$
   `NotSubSet`$(P,Q) = \{\{a,c,d\},\{a,b,d\}\}$

3. Given $P = \{\{a,d\},\{b,c\}\},\ Q = \{\{b,d\},\{a,b\}\}$
   `CrossProd`$(P,Q) = \{\{d\},\{a\},\{b\}\}$
   `DotProd`$(P,Q) = \{\{a,b,d\},\{b,c,d\},\{a,b,c\}\}$

Our algorithm, called `FIMiner`, adopts the FP-growth approach that has been described in Chapter 2. The `FIMiner` is also similar to the ZBDD-growth algorithm, except that we use one ZBDD, instead of a vector of ZBDDs, as database representation,

Table 4.1: Primitive ZBDD operations, P and Q are two ZBDDs

Notation	Description
0	The 0-terminal node; an empty set, i.e. $\emptyset$
1	The 1-terminal node; a set of an empty itemset, i.e. $\{\emptyset\}$
change(P, x)	Invert all occurrences of item x in P
$P \cap_Z Q$	Set-intersection: itemsets which are element of both P and Q
$P \cup_Z Q$	Set-union: itemsets which are element of either P and Q
$P \cup_{Z_{min}} Q$	Minimal set-union: minimal itemsets which are element of either P and Q
$P \cup_{Z_{max}} Q$	Maximal set-union: maximal itemsets which are element of either P and Q
$P \setminus Q$	Set-subtraction: itemsets which are element of P but not Q
`NotSubSet`(P, Q)	Not Subset: itemsets which are element of P which are not subset of any itemset in Q
`CrossProd`(P, Q)	Cross Product: pair-wise intersections between the itemsets in P and Q
`DotProd`(P, Q)	Dot Product: pair-wise unions between the itemsets in P and Q

and we use a secondary *bitmap* database representation for calculating the frequency of itemsets.

To give an overview, `FIMiner` recursively partitions the database into two databases, and grows frequent patterns incrementally. A chosen item x induces an *x-conditional database*, which corresponds to the itemsets in which it occurs, and patterns which contain the item are found locally within the conditional database. To find patterns which do not contain x, item x can be safely removed from the database and we call the resulting database as *x-reduced database.* Removing x does not change the output patterns because all frequent patterns which contain x only occur in the x-conditional database.

Shown in Algorithm 4.1, the algorithm is invoked by `FIMiner`$(Z_D, \alpha, prefix)$, where Z_D is a ZBDD containing the input data set, and $prefix$ is initially empty itemset (i.e. $prefix = \{\}$). Frequent itemsets are grown from the given prefix, using the input ZBDD which is traversed in a top-down fashion. The support of the itemsets is calculated using the *bitmap* database representation. For a given $prefix$ itemset, *bitmap*$(prefix)$ refers to the bit-vector of the occurrence of the itemset in the initial input data set. We compute *support*$(prefix)$ by counting the number of 1's in *bitmap*$(prefix)$, denoted as $|bitmap(prefix)|$.

We now explain the algorithm line by line. For a given input ZBDD Z_D, the top-node's label is firstly used to grow the current prefix, since it is trivial to obtain its conditional database. Let x be the label of Z_D. By Shannon's decomposition, the 1-child node, labeled as Z_{D_x} contains all itemsets in Z_D which contain x. Thus, the x-conditional database is represented in Z_{D_x}, and patterns which contain x are recursively found by calling `FIMiner` on Z_{D_x} (line 9). To find the x-reduced database, we compute the set-union between the two child-nodes of Z_D (line 11). For higher data compression and efficiency purposes, the non-maximal itemsets are simultaneously removed, which can be computed using the ZBDD's library routine $\bigcup_{Zmax}$. We refer to this operation as *DB-merging.* By the ZBDD's canonical property, nodes may be shared across the x-conditional database, x-reduced database, and the intermediate output.

In addition to the standard ZBDD primitive operations, we push the anti-monotonic support constraint deep inside the routine using an *infrequent prefix pruning* strategy (line 6-7), which is based on the anti-monotonic property of *support.* The support of $prefix \cup \{x\}$ can be computed incrementally, by taking the bit-wise intersection between bitmap$(prefix)$ (carried from the previous mining iteration) and bitmap$(\{x\})$. When the new prefix itemset is known to be infrequent, the sink-0 node is returned to delete the prefix itemset and its supersets from the output (based on the zero-suppression rule). Finally, the recursion terminates when the database is empty (line 1-2).

Since the library ZBDD operations store computation results in a cache, DB-merging can be computed efficiently if many of the conditional DBs share common subtrees. The output ZBDD is incrementally built from each recursion step, using the same variable ordering as the input ZBDD. The intermediate output from the x-conditional database and from the x-reduced database become the 1-child and 0-child of the output node (line 12). Combining the output from each recursion level incrementally builds the output ZBDD in a bottom-up fashion. To obtain the same ordering of database projections, the items are ordered in the ZBDD by their increasing frequency. Comparison between several variable orderings was studied in [110], which showed that the increasing frequency ordering was one of the optimal orderings.

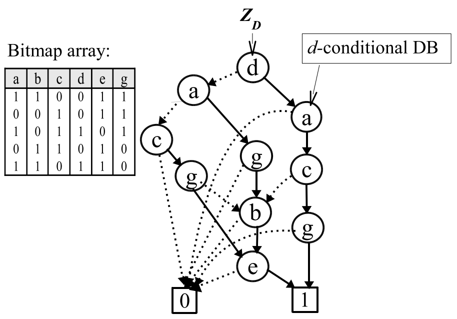
The increasing frequency variable ordering is similar to that used by the FP-growth algorithm for projecting the conditional databases. FP-growth orders the item in each FP-tree by a decreasing frequency, but it projects the first conditional database using the least frequent item. Moreover, the FP-growth allows dynamic variable ordering, but our `FIMiner` algorithm does not.

Example 17. *Figure 4.1(a) shows the input database D. The variables are ordered by their increasing frequency, item d being the least frequent. Thus, d is the first item chosen to grow a prefix. Figure 4.2(b) illustrates the* DB-merging *operation between the child-nodes on Z_D. Identical nodes in the merged database are shared with the input database, as well as the other databases It shows the ZBDD which represents the d-conditional database (conditional DB), which contains itemsets $\{a, c, g\}$ and $\{a, b, e\}$ from transaction 3 and transaction 5 in the input dataset D..*

4.2.1 Maximal Frequent Itemset Mining

We now describe some optimisations that can be applied to our FI mining technique for mining maximal frequent itemsets (MFIs). We call the algorithm **MFI-Miner** (Algorithm 4.2). Using the same core operations as FI-Miner, MFI-Miner has an additional procedure for removing the non-maximal patterns. This is performed using a progressive focusing technique [29], which removes the locally non-maximal patterns from each conditional DB. It can be computed using a ZBDD primitive set-subtraction routine i.e. $Z_{FI_{\overline{x}}} \setminus Z_{FI_x}$, where Z_{FI_x} and $Z_{FI_{\overline{x}}}$ are basically computed as in Algorithm 1. This subtraction operation removes the frequent extensions of $prefix$ which also

id	Itemset
1	$\{a, b, e, g\}$
2	$\{c, e, g\}$
3	$\{a, c, d, g\}$
4	$\{b, c, e\}$
5	$\{a, b, d, e\}$



a	b	c	d	e	g
1	1	0	0	1	1
0	0	1	0	1	1
1	0	1	1	0	1
0	1	1	0	1	0
1	1	0	1	1	0

(a) Input database D

(b) ZBDD representation with increasing frequency variable ordering ($d < a < c < g < b < e$)

Figure 4.1: A ZBDD representation for an input database D; the d-conditional database share nodes with the 1-child of the top node in the ZBDD

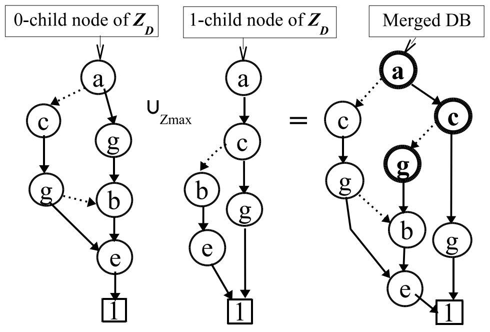


Figure 4.2: Set-union between two child nodes of Z_D to compute the DB-merging operation (nodes marked with bold lines are the newly created nodes as a result of this operation)

Algorithm 4.1 `FIMiner`(Z_D,α, $prefix$)

Input: A ZBDD Z_D containing the database induced by a prefix itemset, a minimum support threshold α, a prefix itemset $prefix$ which projects Z_D.

Output: A ZBDD Z_{FI} containing the set of frequent itemsets in Z_D

```
1: if ($Z_D$ is a sink node) then
2:     return $Z_D$ /* Terminal case */
3: end if
4: /* Let $Z_D = node(x, Z_{D_x}, Z_{D_{\overline{x}}})$ */
5: $prefix_x = prefix \cup \{x\}$
6: if ($support(prefix_x) < \alpha$) then
7:     $Z_{FI_x} = 0$ /* Infrequent prefix pruning */
8: else
9:     $Z_{FI_x} =$ FIMiner($Z_{D_x}, \alpha, prefix_x$) /* Grow new prefix $prefix_x$ and mine patterns which
       contain $x$ from the $x$-conditional database */
10: end if
11: $Z_{FI_{\overline{x}}} =$ FIMiner($Z_{D_{\overline{x}}} \bigcup_{Z_{max}} Z_{D_x}, \alpha, prefix$) /* DB-merging: reduce database $Z_D$ by $x$ and
    mine patterns which do not contain $x$ */
12: return $Z_{FI} =$ getNode($x, Z_{FI_x}, Z_{FI_{\overline{x}}}$) /* Combine the output patterns */
```

Note: $support(prefix_x) = |bitmap(prefix) \cap bitmap(\{x\})|$.

occur as frequent extensions of $prefix_x$ since they are non-maximal local to the current database. Additionally, our algorithm can adopt some of the advanced pruning techniques used in [29, 64, 158]. For this purpose, an itemset *tail* is maintained for each database. It contains the items that occur in the relevant database, and used for pre-processing each conditional database in the following ways.

The first subroutine **mineMFI**$_x$ grows the new prefix $prefix_x$. It employs two pruning strategies. The x-conditional database, denoted Z_{D_x}, is pruned early by removing items which are infrequent in the conditional database. This pruning uses an itemset $freqTail$ which contains items in *tail* whose bitmap support exceeds the minimum support threshold. Since $freqTail$ is a single itemset, the `CrossProd` operation between the ZBDDs representing $freqTail$ and Z_{D_x} (line 6), the result is the intersection between each itemset in Z_{D_x} with $freqTail$. The second pruning uses an itemset *classEquivalent* (line 7) which is a maximal itemset that occurs in every transaction which contains the top-item x.

The second subroutine **mineMFI**$_{\overline{x}}$ finds patterns which do not contain x. It uses an itemset *classSubsumed* (line 1) which contains items that occur only in transactions which contain x. The items of *classSubsumed* can be removed from the x-reduced database, since their frequent supersets which are found from the reduced database are non-maximal.

The `CrossProd()`, `DotProd()`, and `NotSubSet()` are ZBDD library routines for computing set intersection, pair-wise intersection, and non maximal removal between two sets of itemsets.

Algorithm 4.2 `MFIMiner`(Z_D, α, $prefix$, $tail$)

```
Input: A ZBDD $Z_D$ containing the database induced by a prefix itemset, a minimum support
    threshold $\alpha$, a prefix itemset $prefix$ which projects $Z_D$, a tail itemset $tail$ which contains
    items which occur in $Z_D$.
Output: A ZBDD $Z_{MFI}$ containing the set of MFIs in $Z_D$
1: if ($Z_D$ is a sink node) then
2:   return $Z_D$ /* Terminal case */
3: end if
4: /* Let $Z_D = node(x, Z_{D_x}, Z_{D_{\overline{x}}})$ */
5: $prefix_x = prefix \cup \{x\}$
6: $Z_{MFI_x}$ = Subroutine mineMFI$_x$ /* Grow new prefix $prefix_x$ */
7: $Z_{MFI_{\overline{x}}}$ = Subroutine mineMFI$_{\overline{x}}$ /* DB-merging and mine patterns which do not contain
   $x$ */
8: $Z_{MFI_{\overline{x}}}$ = NotSubSet($Z_{MFI_{\overline{x}}}, Z_{MFI_x}$) /* Progressive focusing: remove non-maximal pat-
   terns */
9: $Z_{MFI}$ = getNode($x, Z_{MFI_x}, Z_{MFI_{\overline{x}}}$) /* Combine the output patterns */
Subroutine mineMFI$_x$: grow $prefix_x$
1: $freqTail$ = find the set of frequent tail items
2: $classEquivalent$ = find the set of items which occur in every instance in $Z_{D_x}$
3: if ($support(freqTail) \geq \alpha$) then
4:   return $freqTail$ /* FHUT pruning */
5: end if
6: $Z_{D_x'}$ = CrossProd($Z_{D_x}$, $\overline{freqTail}$) /* Infrequent items pruning */
7: $Z_{D_x''}$ = CrossProd( $Z_{D_x'}$, $\overline{classEquivalent}$) /* Class-equivalent pruning */
8: $Z_{MFI_x}$ = MFIMiner($Z_{D_x''}$,$\alpha$,$prefix_x$, $freqTail$)
9: return DotProd($Z_{MFI_x}$, $classEquivalent$)
Subroutine mineMFI$_{\overline{x}}$: merge database and mine patterns which do not contain $x$
1: $classSubsumed$ = find the set of items which do not occur in $Z_{D_{\overline{x}}}$
2: $Z_{project_x}$ = CrossProd($Z_{D_x}$, $\overline{classSubsumed}$) /* Class-subsumed pruning */
3: $Z_{D_{x+\overline{x}}} = (Z_{D_{\overline{x}}} \bigcup_{Z_{max}} Z_{project_x}) \setminus Z_{MFI_x}$ /* DB-merging and pre-computed MFI pruning
   */
4: return MFIMiner($Z_{D_{x+\overline{x}}}$, $\alpha$, $prefix$, $tail$)
```

Note: $\overline{P} = I - P$, where I is the set of domain items. `CrossProd()`, `DotProd()`, and `NotSubSet()` are ZBDD library routines

4.2.2 Closed Frequent Itemset Mining

Our algorithm for mining closed frequent itemsets, `CFIMiner`, uses a similar framework as `MFIMiner`, and employs a progressive focusing technique for removing the non-closed patterns. However, the closed constraint requires the support of an itemset to be compared against its subset(s). Thus, the support information has to be represented in the output data structure, which was not necessary for `FIMiner` and `MFIMiner`. Additionally, `CFIMiner` can also adopt the more advanced pruning techniques found in existing algorithms, using a similar mechanism to `MFIMiner` which maintains a *tail* itemset.

To represent the patterns' support in the ZBDD output, it uses additional variables, which are appended to each pattern. We refer to these extended pattern representations as *item-support-sets*. In order to achieve higher compression, we use the binary representation of the support values. For a database containing N transactions, we reserve $log_2(N)$ binary variables to represent the itemsets.

Example 18. *Given a database containing 5 transactions, 3 support-encoding variables are reserved. Let* r_0 r_1 r_2 *be the support-encoding binary variables, such that* $r_2 = 0$, $r_1 = 0$, $r_0 = 1$ *represents a support of* 1, $r_2 = 0$, $r_1 = 1$, $r_0 = 0$ *represents* 2, *etc. For instance, the* item-support-set *representation of itemsets (with their corresponding frequencies)* $\{\{b,e\} : 3, \{a,b,e\} : 2, \{a,b\} : 2\}$ *is* $\{\{b,e,r_1,r_0\}, \{a,b,e,r_1\}, \{a,b,r_1\}\}$. *Furthermore, itemsets in the maximal item-support-sets correspond to closed itemsets. Item-support-set* $\{a,b,r_1\}$ *is not maximal since* $\{a,b,e,r_1\}$ *exists. However,* $\{a,b,e,r_1\}$ *is maximal, thus, itemset* $\{a,b,e\}$ *is a closed itemset.*

4.2.3 Row-wise Closed Frequent Itemset Mining

Let us now describe how our ZBDD-based mining framework can be adopted to the row-wise mining framework for finding closed frequent itemsets (CFIs) in sparse high dimensional datasets, which have been introduced in [125, 96, 124]. In this framework, the patterns are mined by searching for possible row (instead of item) combinations. We use the terminologies defined in Chapter 2 (Section 2.3.3).

Existing row-wise mining algorithms [125, 96] perform a depth-first search in the lattice of row-id combinations. We will describe a pattern growth, bottom-up algorithm [125] based on the use of a ZBDD, although adapting it to the top-down algorithm [96] is also certainly possible.

The row-wise mining algorithm, namely **RowCFIMiner**, is shown in Algorithm 4.3. We employ the same framework as the column-wise algorithm `FIMiner` (Section 4.2), except that the ZBDD variables now correspond to row-IDs, instead of items. The input ZBDD database is traversed top-down, and the conditional database is recursively projected by the top-item of the ZBDD. Here, the database is transposed so that it contains the *bit_support* of each item in the original database. A supplementary bitmap data structure is also used for computing the *row_support_set* for each row combination. A comparison between `RowCFIMiner` and `FIMiner` follows:

In a similar prefix-growth framework to `FIMiner`, the row-wise algorithm grows prefixes of the rowsets (i.e. sets of row-ids) from the input database, instead of itemsets. The rowsets are grown until its length exceeds the minimum support threshold, or until the *row_support_set* is empty. More specifically, as each rowset is grown, its *row_support_set* is incrementally computed based on the bitmap data representation. When the length of the rowset reaches the minimum support, its *row_support_set* is a fully-grown closed frequent itemset. Unlike `FIMiner`, where the output ZBDD is built incrementally bottom-up, the row-wise algorithm inserts each pattern one at a time into the output ZBDD, using the ZBDD's primitive routine to compute set-union.

Unlike in our column-wise mining algorithms, there is no sharing between the input and the output ZBDDs in row-wise mining, since now they contain different sets of variables. To maximise node sharing within the output ZBDD, which stores the closed frequent itemsets, we use the same variable ordering as in our column-wise algorithm i.e. ordered by increasing frequency.

Our algorithm adopts the pruning strategies in [125, 124]. Additionally, using ZBDD's caching utility, we store the height (i.e. the number of items in its longest path) of the ZBDD which represents each conditional database in a cache. Since the ZBDD represents the row-ids, the height of a ZBDD gives the number of itemsets of the database. If the height of the database is less than the minimum frequency threshold, then we can prune that database, since all of its subsets are infrequent.

4.3 Weighted Zero-suppressed Binary Decision Diagrams (Weighted ZBDDs)

Support counting using bitmaps in our algorithms described in the previous section can be costly, especially in dense high dimensional datasets, which contain many long patterns. To eliminate this overhead, we introduce an original variant of ZBDD, namely the **Weighted Zero-suppressed Binary Decision Diagram (Weighted ZBDD)**. A WZBDD allows the counts of the itemsets to be represented using edge-weights, which in turn allows more efficient frequent itemset mining. In a Weighted ZBDD, every edge is attributed by a positive integer value. The node's weight refers to the weight of the

Algorithm 4.3 `RowCFIMiner`(Z_{TD},α, $rprefix$)

Input: A the transposed database Z_{TD}, a minimum support threshold α, and a prefix rowset $rprefix$ that projects Z_{TD}

Output: A ZBDD Z_{CFI} containing the set of closed frequent itemsets in Z_{TD}

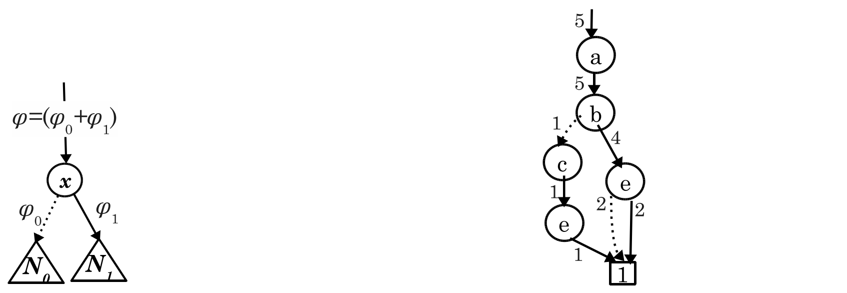
```
1:  if ($Z_{TD}$ is a 0-sink node) or ($row\_support\_set(rprefix)$ is empty) then
2:      return 0 /* Terminal case */
3:  end if
4:  if ($Z_{TD}$ is a 1-sink node) and ($rsupport(rprefix, D) < \alpha$) then
5:      return 0 /* Infrequent itemset pruning */
6:  end if
7:  if (rsupport($rprefix$) $\geq \alpha$) then
8:      return $row\_support\_set(rprefix)$
9:  end if
10: /* Let $Z_{TD} = node(r, Z_{TD_r}, Z_{TD_{\overline{r}}})$ */
11: $rprefix_r = rprefix \cup \{r\}$
12: $Z_{CFI_r} =$ RowCFIMiner($Z_{TD_r}, \alpha, rprefix_r$) /* Grow new rowset $rprefix_r$ and mine
    patterns which occur in row $r$ */
13: $Z_{CFI_{\overline{r}}} =$ RowCFIMiner($Z_{TD_{\overline{r}}} \bigcup_{Z_{max}} Z_{TD_r}, \alpha, rprefix$) /* DB-merging and mine pat-
    terns which do not occur in row $r$ */
14: $Z_{CFI} = Z_{CFI_r} \bigcup Z_{CFI_{\overline{r}}}$ /* Combine the output patterns */
15: return $Z_{CFI}$
```

Note: $row\_support\_set(rprefix_r) = row\_support\_set(rprefix) \cap row\_support\_set(\{r\})$.

incoming edge of that node. There exist other weighted types of Decision Diagrams [27, 154, 123] for manipulating pseudo-boolean functions, which we also discussed earlier in Chapter 3, but they use different semantics from our Weighted ZBDDs. Formally, we define an internal Weighted ZBDD node as follows:

Definition 16. *Each node in a Weighted Zero-suppressed Binary Decision Diagram is a pair $\langle \varphi, \vartheta \rangle$, where ϑ is a ZBDD node, and φ is the weight of this node.*

The weights in a Weighted ZBDD are anti-monotonic. Their values are decreasing as the nodes are positioned lower in the structure. The weight of a node corresponds to the total frequency of the represented itemsets below it. Given a node N ,we define weight(N), which gives the weight of node N. If the node is a 1-sink node, $\varphi = 1$ and weight(N) represents the frequency of the empty itemset $\{\}$. If the node is a 0-sink node, $\varphi = 0$ and weight(N) is 0, since the node represents an empty data set. If the node is an internal node, its two child-nodes correspond to two partitions of the data set, hence, the node's weight is equal to the sum of the weights of its child-nodes (Figure 4.3a), calculated by the following function:



Var. ordering: $a < b < c < d < e$

Weighted ZBDD node

(a)

Weighted ZBDD representation for $\{\{a,b,e\}:2,\{a,c,e\}:1,\{a,b\}:2\}$

(b)

Figure 4.3: Weighted ZBDDs

$$weight(N) = weight(N_1) + weight(N_0) \tag{4.1}$$

where N_1 (resp. N_0) denotes the 1-child (resp. 0-child).

Weighted ZBDDs are canonical, that is, nodes which contain the same set of itemsets with the same corresponding supports are merged. Consequently, ZBDD's set-union routine needs to be adapted for Weighted ZBDDs to add the counts of any itemset which occurs in both of its operands.

Example 19. *Figure 4.3b shows an example of a Weighted ZBDD representing a set of itemsets (with their counts): $\{\{a,b,e\}:2,\{a,c,e\}:1,\{a,b\}:2\}$. The weight of node* a *is 5, i.e. the total number of itemsets. The two node* e*'s are not merged, since their child-nodes have different weights, respectively.*

Table 4.2 shows a comparison between Weighted ZBDDs and the other types of BDDs. The zero-suppression rule is employed in ZBDDs and Weighted ZBDDs. BMDs [27] also have weighted edges, which were proposed for representing pseudo-boolean functions, but BMDs use a different function decomposition from Weighted ZBDDs. BMDs decompose the incoming weight for each node into a multiplicative and additive function of the outgoing weights, whereas Weighted ZBDDs only use simple addition, thus, allowing trivial frequency calculation of the variable in the root node.

Table 4.2: Comparison between Weighted ZBDD and other BDD Variants

Data Structure	Zero-suppressed	Weighted Edges
Binary Decision Diagrams (BDDs)	No	No
Zero-suppressed BDDs (ZBDDs)	Yes	No
Binary Moment Diagrams (BMDs)	No	Yes
Weighted ZBDDs (WZBDDs)	Yes	Yes

4.4 Complexity Analysis

In the following discussion, we discuss the computational complexity for creating and manipulating nodes in Weighted ZBDDs.

4.4.1 Constructing Weighted ZBDDs

The computational cost for creating a Weighted ZBDD node is the same as for a non-weighted ZBDD, requiring one look-up operation to the *uniquetable*, which is the same as that in a BDD. A *uniquetable* stores every node in the BDD/ZBDD/Weighted ZBDD, which is usually implemented as an array of linked lists. If two keys have the same hash value, then the two entries are stored in the linked list. More details about the BDD's *uniquetable* can be found in Chapter 3. The same hash function for ZBDDs can be used for weighted ZBDDs, as is. Each Weighted ZBDD node, however, takes up larger memory size, to store the additional weight value. When representing the same database, the number of nodes in the Weighted ZBDD cannot be smaller than the number of nodes in the ZBDD.

Theorem 4. *The time complexity for creating a Weighted ZBDD that consists of N nodes is $O(Nlog_2N)$.*

Proof. When creating a node, the hash key for that node is computed and one cache-lookup operation is performed to find a pre-existing entry. Given that nodes with the same hash key are stored as a sorted linked list, the lookup operation requires a single traversal of that list which has $O(log_2n)$ time complexity, where n is the number of unique nodes. Thus, when creating N nodes, the overall time complexity is $O(Nlog_2N)$, since N lookup operations are performed and the number of unique nodes is $O(N)$. □

Theorem 5. *Consider a database containing k unique itemsets with a maximum size of L items. The number of nodes in the Weighted ZBDD representation is $O(kL)$, and the height of the Weighted ZBDD is $O(L)$.*

Proof. In the given database, there are kL elements in total. In the worst case, each of those elements is represented by a node in the Weighted ZBDD, resulting in a Weighted ZBDD with kL unique nodes. The height refers to the maximum length of any path in the Weighted ZBDD, which is L, since each path corresponds to a unique itemset. □

In practice, the number of nodes is much smaller than kL, since Weighted ZBDDs allow node-sharing between common prefix paths, as well as suffix paths across multiple intermediate databases. Since the number of nodes of the Weighted ZBDD is $O(kL)$, hence, its construction has $O((kL).log_2(kL))$ time complexity.

The construction of a ZBDD has a similar time complexity to that of a Weighted ZBDD, but there is a lower node-sharing probability in a Weighted ZBDD. However, due to their weighted-edges, node-sharing is only limited amongst itemsets which have the same frequency. If there are many itemsets which share the same frequency, then many nodes can be shared in the Weighted ZBDD.

The basic ZBDD operations, such as set-union and set-subtraction, have polynomial time-complexity with respect to the number of nodes in the input Weighted ZBDDs, similar to the complexity of basic ZBDD operations. Our algorithm for mining frequent itemsets employs the primitive Weighted ZBDD's *add*() operation which adds two sets of itemsets and combines their frequencies. The procedure is similar to the predefined procedure for ZBDDs.

4.4.2 Efficiency of Weighted ZBDDs for Frequent Itemset Mining

By the canonical property of Weighted ZBDDs, multiple intermediate databases (i.e. conditional databases and their intermediate outputs) may share nodes or subgraphs. Representing the frequency values as edge weights in the data structure is very useful for mining frequent itemsets, since it allows efficient identification between any two identical databases (i.e. containing the same set of itemsets and frequencies).

Two identical Weighted ZBDDs represent the same set of itemsets and the same corresponding support, thus, they also contain the same set of frequent itemsets. This is possible to be done by adopting BDD's caching mechanism. By caching the computed results from each conditional database, Suppose different prefixes project the same conditional database, patterns from the earlier computation can be reused and redundant computation can be avoided.

In our study, we use the caching implementation used by an existing BDD package called JINC [123]. The package was developed for studying a different type of weighted BDDs. To cache an operation, the cache uses a hash function which maps the memory address of the input nodes and the address of the output of the operation. The caching mechanism is provided by the BDD package, we do not explicitly show it in the algorithm described shortly. A caching mechanism in frequent itemset mining has been studied also in cache-conscious prefix trees [60], but they aim to minimise access to physical memory. On the other hand, BDD's caching aims to make similar intermediate computations able to re-use the computed results and reduce the overall number of database projections.

4.5 Performance Study

In this section, we analyse the performance of our ZBDD-based techniques for mining MFIs and CFIs. The algorithms were implemented in C++ using the BDD library which was used in the work in[123]. All experiments were performed on a 2.0 GHz CPU, 3 GB RAM, running Solaris, with a CPU-timeout limit of 100,000 seconds per mining task.

We concentrate on gene-expression data sets, since they are one of the most challenging kinds of high dimensional data sets for pattern mining. They typically consist of only a few number of samples (i.e. rows). The number of frequent itemsets in a microarray data set can be enormous [35]. Works in [24, 34, 136, 84, 88, 86] have also studied other itemset mining problems in microarray data sets.

The following are the data sets used in our experiments: i) Leukaemia ALL-AML [1], ii) lung cancer[2] which was previously studied in [124]. The ALL-AML data set contains 72 samples (i.e. transactions), each sample is described by 7129 genes (i.e. attributes). The lung cancer data set contains 32 samples, described by 12533 genes. Continuous attribute values are discretised using an entropy discretisation method.

In the item-wise framework, we implement our techniques, i.e. `ZBDDMiner`, `WZDDMiner`,

[1]http://research.i2r.a-star.edu.sg
[2]http://www-genome.wi.mit.edu/cgi-bin/cancer

which are our ZBDD and WZDD based algorithms, using only the basic infrequent prefix pruning. `ZBDDMiner*` and `WZDDMiner*` are their respective variants which use the more advanced pruning techniques. We employ an increasing frequency variable ordering, so that the least frequent item is positioned at the top of the ZBDD/WZBDD, which allows more effective pruning based on the *infrequent prefix* pruning strategy.

Their performance is compared against the state-of-the-art FP-tree based algorithms, i.e. `FP-growth*` [64], `FP-close*`, and `FP-max*`[3], which performed best on dense data sets. We also compare the performance of our algorithms against the latest version of `LCM`, referred to as `LCM-ZBDD` [117], which is based on a combination of prefix-tree and array data representations, and employs a ZBDD for managing the output patterns. `LCM-ZBDD` was shown to have a linear time complexity with respect to the number of patterns. All of these contender algorithms were obtained from their respective original authors.

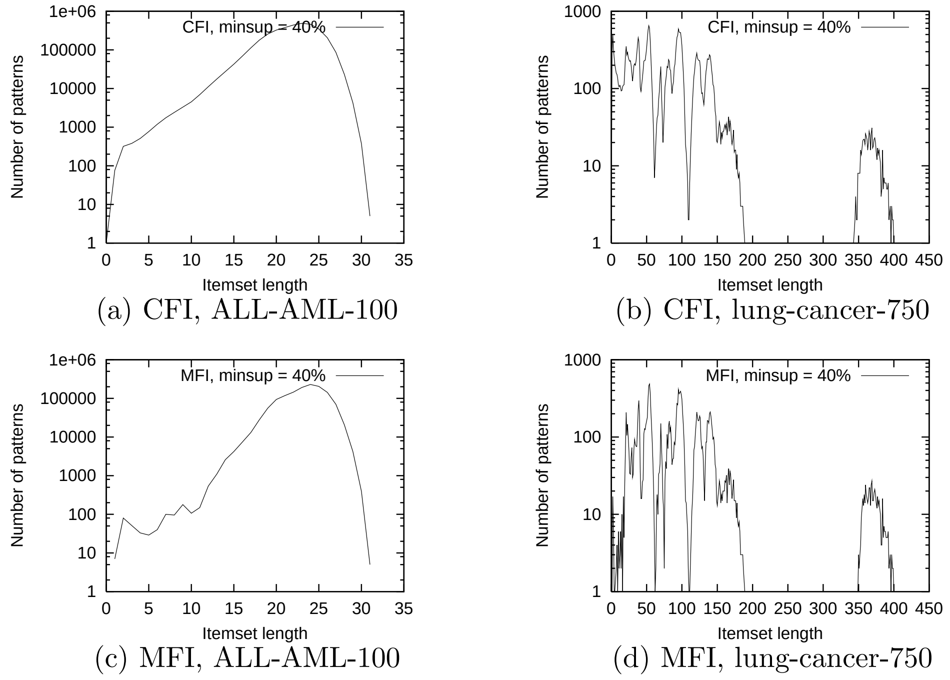
From each algorithm, we measure the following: i) the CPU time spent for mining, ii) the number of nodes in the output FP-trees, ZBDDs, or WZDDs, and iii) the total node usage, which is the total number of nodes used through out mining including those used for storing the input database, the intermediate databases, and the final output patterns. For ZBDDs or WZDDs, shared nodes are counted only once.

We performed experiments for mining frequent itemsets (FIs), closed frequent itemsets (CFIs), and maximal frequent itemsets (MFIs). When WZDDs are used for mining FIs or MFIs, storing the support values in the output is not necessary, thus, the output patterns are represented in a ZBDD. We also performed experiments for mining CFIs in the row-wise mining framework. We choose the `RERII` [124] algorithm as a baseline in our study, which we obtained from its author, since it is the state of the art algorithm for row-wise mining, and it is based on the use of FP-trees.

4.5.1 Data and Pattern Characteristics

To obtain results within the CPU-timeout limit for low support threshold values, we used the first 100 attributes from the ALL-AML data set, where the attributes are ordered according to their entropy values from highest to lowest (i.e. attr.1 has highest entropy). All methods could not complete mining at low support thresholds when all of the attributes were used due to the CPU time out constraint. For a similar reason, we

[3]`FP-growth*` uses more advanced pruning strategies than `FP-growth`, and `FP-close*` (resp. `FP-max*`) is the extension of `FP-growth*` for mining closed (resp. maximal) frequent itemsets.



(a) CFI, ALL-AML-100 (b) CFI, lung-cancer-750
(c) MFI, ALL-AML-100 (d) MFI, lung-cancer-750

Figure 4.4: Pattern length histogram

used the first 750 attributes from the lung cancer data set. In the following discussions, we refer to the respective data sets as ALLAML-100 and lung-cancer-750.

We firstly study the characteristics of each data set, by measuring the compactness of their ZBDD representations, based on two factors: node fan-out and node fan-in. To measure the fan-out compression factor, we implement a Binary Decision Tree (BDTree), which is a relaxed type of BDD with no node fan-in. The fan-in compression factor[4] is the relative difference between the number of nodes in the BDD and the BDTree. The fan-out compression factor[5] is the relative difference between the number of nodes in the BDTree and the total number of items in the input transactions. Table 4.3 shows that the ALL-AML-100 data set has a higher fan-out and fan-in compression factor than lung-cancer-750. It indicates that the itemsets in the ALL-AML-100 data set have more items in common, for which the ZBDD representation has more node sharing than that for the lung-cancer-750 data set.

To further investigate the density/sparsity of the data sets, we also study the pattern characteristics in each data set. Since the set of maximal frequent itemsets is

[4] $fanIn = 1 - \frac{|BDD|}{|BDTree|}$
[5] $fanOut = 1 - \frac{|BDTree|}{total\_number\_of\_items}$

Table 4.3: Fan-out and fan-in compression factors of the input ZBDD

Dataset name	ZBDD	
	fanOut	*fanIn*
ALL-AML-100	0.012	0.202
lung-cancer-750	0.001	0.012

a subset of the closed frequent itemsets, we firstly investigate the distribution of the closed itemsets.

Figures 4.4a and 4.4b show the length distribution of the CFIs for the two data sets, respectively. Given a support threshold 40%, the ALL-AML-100 data set contains millions of long patterns, i.e. the size of the patterns which has the highest frequency is around 23-25 items, or about 80% of the maximum size. In the lung cancer data set, given the same support threshold, there exist fewer patterns than that in the ALL-AML data set, and the patterns are longer with a median size of 100 items, or about 25% of the maximum size of 400.

Moreover, the length distribution of the maximal frequent itemsets in the ALL-AML-100 data set (Figure 4.4c) shows a smaller number of short patterns, whose length are shorter than the median, compared to the closed frequent itemsets. It shows that many of the long closed frequent itemsets are also maximal frequent itemsets. For the lung-cancer-750 data set (Figure 4.4d), however, there is a reduction in the number of patterns whose length is around the median, showing that many of the closed frequent itemsets are not maximal.

We henceforth regard the ALL-AML-100 as a *dense* data set, and the lung-cancer-750 as a *sparse* data set. When mining the frequent itemsets in the lung-cancer data set, however, we used only the first 50 attributes. None of the algorithms could perform within a reasonable time if more attributes were included, due to the excessive number of long patterns that exist when the support threshold value is relatively low.

4.5.2 FI Mining Performance

Figure 4.5 shows that for the ALL-AML data set, `WZDDMiner` is 10 times faster than `ZBDDMiner`, and it is also up to 100,000 times faster than `FP-growth*` and `LCM-ZBDD` given a support threshold of 35%. For the lung cancer dat set, which contains many long patterns, moreover, `WZDDMiner` is able to achieve almost a million times speed-up than `LCM-ZBDD`, completing within 1 second when the support threshold value is 36%.

This significant speed-up shows that WZDDMiner is superior for mining a large number of long frequent itemsets.

4.5.3 CFI Mining Performance

Let us now observe the performance comparison between our algorithms for mining CFIs in the sparse data set against FP-close*. Figure 4.6 shows that LCM-ZBDD is the fastest, and the WZDD algorithms are faster than the ZBDD algorithms. Due to the CPU timeout constraint, the ZBDD algorithms could only complete mining for support threshold $\geq 50\%$, whilst the WZDD algorithms are more scalable and able to complete mining for support threshold $\geq 36\%$. Now let us look closer at the effectiveness of the pruning strategies. The two figures 4.6a and 4.6b show that the pruning strategies in either WZDDMiner* and ZBDDMiner* do not give much improvement on WZDDMiner and ZBDDMiner.

Figure 4.7a shows that both WZDDs and ZBDDs (which include the support encoding variables) are able to achieve higher compression than the FP-tree for storing the output patterns. For support threshold of 40% whose length distribution of the patterns is shown earlier, the WZDD is about 50 times smaller than the FP-tree, excluding the header table of the FP-tree.

Furthermore, the total node usage of each algorithm is shown in Figure 4.7b. It shows that FP-close* uses the least number of nodes for representing all the databases (including the input, the output patterns, and the conditional databases). The reason for this is the dynamic variable ordering employed by FP-close*, which allows different conditional databases to use different variable orderings. The ZBDD/WZBDD, on the other hand, uses a static variable ordering for maximising the amount of node sharing across multiple conditional databases. The results from this experiment show that the conditional databases projected by FP-close* are more compact than their ZBDD/WZBDD representations. Moreover, the header tables used by the FP-trees are relatively smaller than the secondary bitmap used by the ZBDDs. The discrepancies between FP-close* and our ZBDD/WZDD algorithms are decreasing as the support threshold decreases. ZBDDMiner* uses about 5 times fewer nodes than ZBDDMiner given minimum support threshold of 68%, which shows the effectiveness of the advanced pruning strategies in ZBDDMiner* for reducing the size of the intermediate ZBDD structures. On the other hand, WZDDMiner* and WZDDMiner use 5 times as many nodes as ZBDDMiner, while avoiding the use of the secondary bitmap representation which are exhaustive for a sparse data set.

Figure 4.8 shows the runtime comparison between the algorithms for mining CFIs in the dense data set. `WZDDMiner` has the fastest runtime for low support thresholds, i.e. $< 40\%$. When advanced pruning strategies are used, `WZDDMiner*` is up to 500 times slower than `WZDDMiner`. On the other hand, `ZBDDMiner*` can achieve up to 100 times speedup factor compared to `ZBDDMiner`. `ZBDDMiner` could not complete within the CPU time limit for support threshold $< 50\%$, and `FP-close*` exceeds the memory limits for support threshold as low as 27%.

Figure 4.9a shows that the WZDDs for storing the output patterns have significantly fewer nodes than the FP-trees, which demonstrates the ability of WZDDs to compactly represent huge volume of patterns that exist in a dense data set. Due to the additional support-encoding variables, the ZBDDs contain about 100 times more nodes than the WZDDs. Being similar to `WZDDMiner*`, `WZDDMiner` is able to achieve high data compression. This shows that without the advanced pruning strategies, `WZDDMiner` can achieve time as well as space efficiency.

The total node usage of each of the algorithms is shown in Figure 4.9b, which shows that `WZDDMiner` uses the least number of nodes, except for support threshold $\geq 60\%$, where the `FP-close*` has the least total node usage.

4.5.4 MFI Mining Performance

Figure 4.10 shows the runtime comparison between the algorithms for mining MFIs in the sparse data set. For low support threshold, i.e. $< 50\%$ in the sparse data set, `LCM-ZBDD` is faster than the other algorithms, and `WZDDMiner` and `WZDDMiner*` are exponentially faster than `FP-max*`. In the dense data set, `WZDDMiner` can be 5 times faster than `LCM-ZBDD`, particularly when the support threshold is $> 20\%$.

Unlike in mining CFIs, `WZDDMiner*` does improve the mining efficiency of `WZDDMiner` for mining MFIs given a low support threshold, being 10 times faster when the support threshold is 36%. The effectiveness of the pruning strategies in this scenario can be justified by the characteristics of the patterns (shown in Section 4.5.1), which shows that there are many closed frequent itemsets which are not maximal.

Figure 4.11a shows that the ZBDD representations for storing the output patterns are smaller than the FP-trees for support threshold $< 60\%$, being up to 1000 times smaller at support threshold of 40%. This shows the significant data compression which can be achieved by ZBDDs for storing a huge number of short patterns which exist in a sparse data set.

To compare the effectiveness of the advanced pruning strategies in ZBDDMiner*, Figure 4.11b shows that ZBDDMiner* uses the least number of nodes through out mining for low support threshold values, i.e. $< 60\%$, although its run time is slower than ZBDDMiner. More specifically, ZBDDMiner* can achieve about 50 times data compression over both WZDDMiner and WZDDMiner*. FP-max* uses up to 100-1000 times more nodes than both WZDD and ZBDD algorithms when the support threshold is lower than 50%. While the FP-trees require additional memory for their header tables, and the ZBDDs for thei bitmap representations, WZBDDMiner* allows the highest compression for this data set.

In the dense data set, the relative performance between WZDDMiner and FP-max* is similar to that in the sparse data set. Figure 4.12 shows that WZDDMiner outperforms FP-max* for support threshold values $\leq 50\%$. WZDDMiner*, ZBDDMiner, and ZBDDMiner* could not complete mining within the CPU time limit constraint for support threshold values less than 40%, 50%, 45%, respectively. The LCM-ZBDD, interestingly, is twice slower than WZDDMiner for support threshold $\geq 30\%$, but it is exponentially faster for lower support threshold. This shows the strength of LCM-ZBDD for mining MFIs at low support threshold in this sparse data set, using its pruning strategies which prune the non-closed itemsets early. When the support threshold is relatively high, its pruning is less effective, whilst WZDDMiner performs well, since it does not have the overhead cost for checking closed itemsets.

Figure 4.13b shows that ZBDDMiner and ZBDDMiner* use smaller number of nodes than WZDDMiner, being 10 times smaller for support threshold of 70% at the expense of the use of the bitmap representations. The discrepancy decreases as the support threshold decreases, though. WZDDMiner* has a larger node usage than WZDDMiner and the other ZBDD algorithms. This shows that the pruning strategies are not very effective for mining many short patterns in this data set. Moreover, FP-max* has the largest total node usage for support threshold values $\leq 55\%$, which is also the case when mining the sparse data set.

4.5.5 Effectiveness of Caching in WZDDMiner

One benefit of using WZDDs instead of ZBDDs is that WZDDMiner allows the conditional databases to be cached and re-used if the same database is projected by different prefixes. We study the *hit rate*, which is the successful rate of finding a conditional database in the cache, when mining CFIs in both data sets using both `WZDDMiner` and `WZDDMiner*`. Similar trends are found from the maximal frequent itemset mining scenarios.

The figures in Figure 4.14 show that the hit rate increases as the support threshold value decreases, for which the number of conditional database projections increases. In the dense data set, a maximum hit rate of 0.42 can be achieved by `WZDDMiner*`. Interestingly, `WZDDMiner` shows a different behaviour compared to `WZDDMiner*`. The hit rate achieved by `WZDDMiner` decreases when the support threshold values are less than 50%, whereas `WZDDMiner*` can achieve higher or constant hit rate. This shows that without employing the advanced pruning strategies, many more conditional databases are projected, and they are more dissimilar. The pruned conditional databases, on the other hand, are more similar, since the unnecessary items have been removed.

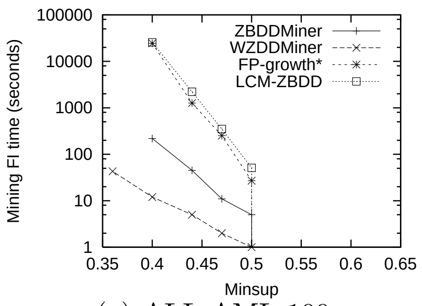
Figure 4.15 shows the absolute counts of the database projections created by both algorithms, which show that `WZDDMiner*` projects exponentially fewer conditional databases than `WZDDMiner` in the sparse data set, and about 5-10 times fewer in the dense data set.

These results show that the pruning strategies in `WZDDMiner*` result in a significant reduction in the number of conditional databases, and improvement in the number of re-used databases. However, it does not guarantee a better time performance if the data set is sparse, due to the small level of node sharing in the initial database.

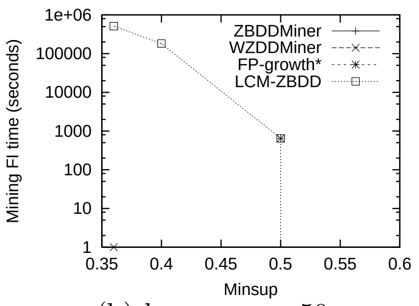
4.5.6 Row-wise CFI Mining Performance

For this set of experiments, we used lung cancer data set with the first 100 attributes. We compare our technique, labeled as `ZBDD rowCFI-Miner`, against `RERII` [124] for which we implemented two variations, being different only in the structure (FP-Tree or ZBDD) for storing the output patterns: i) `RERII-FPTree`, ii) `RERII-ZBDD`.

Figure 4.16a shows the running time of the three algorithms. `RERII-FPtree` is twice as fast as `RERII-ZBDD`, and it performs best for support threshold $\geq 20\%$. `ZBDD rowCFI-Miner` outperforms `RERII-FPtree` at support threshold 10%, but is slightly slower on larger thresholds. `ZBDD-rowCFI-Miner` is at least 1.5 times faster than `RERII-ZBDD` for all support thresholds. To further study ZBDD's efficiency, we compared the output size of `RERII-ZBDD` with the output size of `RERII-FPtree` in terms of the number of nodes (shown in Figure 4.16b). The output ZBDD has 4-8 times

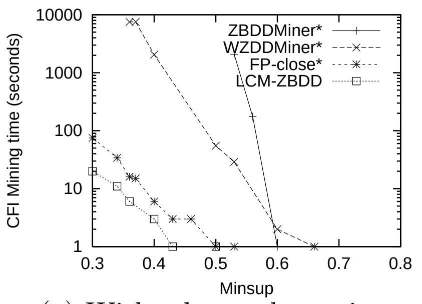


(a) ALL-AML-100

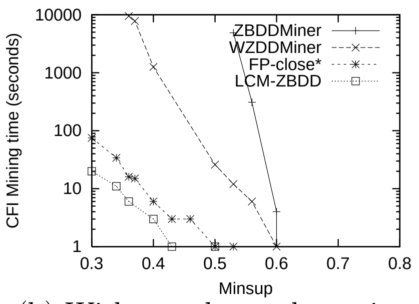


(b) lung-cancer-50

Figure 4.5: Runtime for mining all frequent itemsets (FIs)



(a) With advanced pruning



(b) Without advanced pruning

Figure 4.6: Runtime for mining CFIs in lung-cancer-750 data set (sparse)

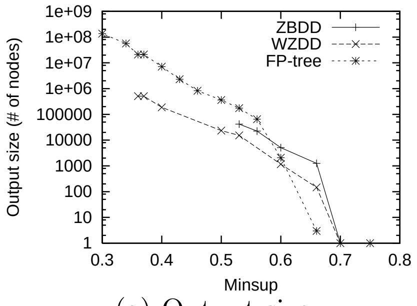
fewer nodes than the FP-tree, with the reduction factor exponentially increasing as the support threshold decreases. This shows that ZBDD can achieve better compression compared to FP-tree, but the tradeoff being more expensive to construct.

4.6 ZBDD vs FP-tree for Mining Frequent Itemsets

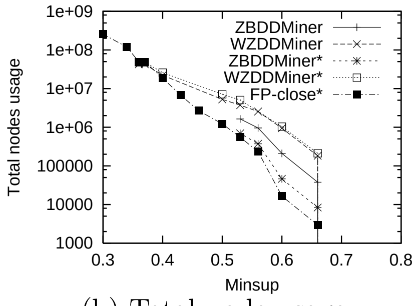
In this section we discuss some advantages and disadvantages of using ZBDDs in an FI mining framework, based on our experimental results, with respect to a number of dimensions: the compactness of ZBDD data structure, the effectiveness of pruning in a ZBDD-based technique, and the efficiency of operations upon it.

4.6.1 ZBDD's Canonicity

The canonical structure of a ZBDD is a a powerful feature which we have shown useful for not only compressing high dimensional output patterns, but also for compressing

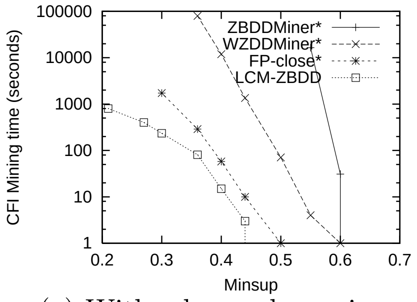


(a) Output size

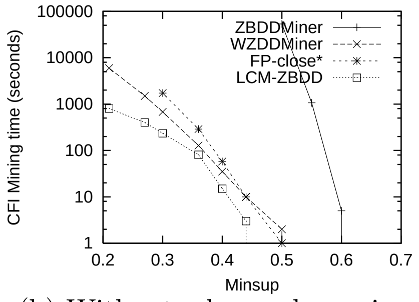


(b) Total node usage

Figure 4.7: Nodes used for mining CFIs in lung-cancer-750 data set (sparse)

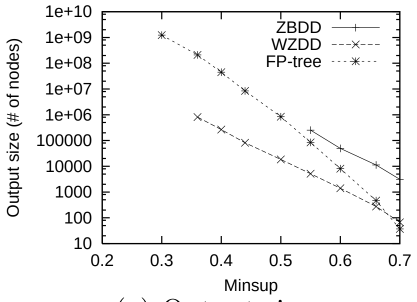


(a) With advanced pruning

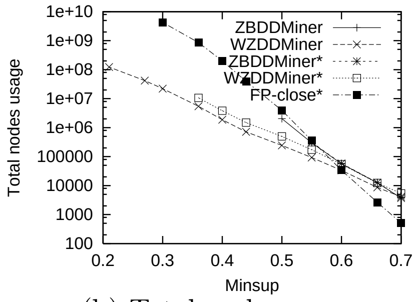


(b) Without advanced pruning

Figure 4.8: Runtime for mining CFIs in ALL-AML-100 data set (dense)

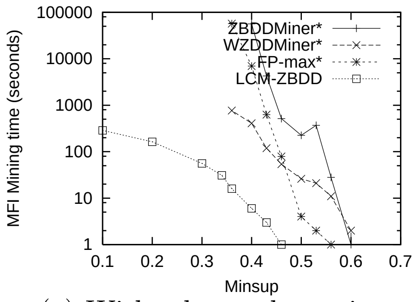


(a) Output size

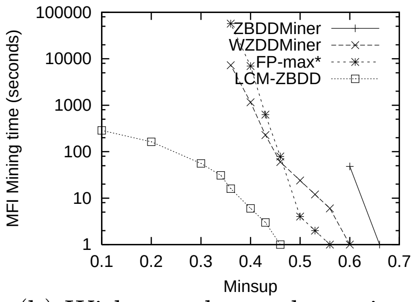


(b) Total node usage

Figure 4.9: Nodes used for mining CFIs in ALL-AML-100 data set (dense)

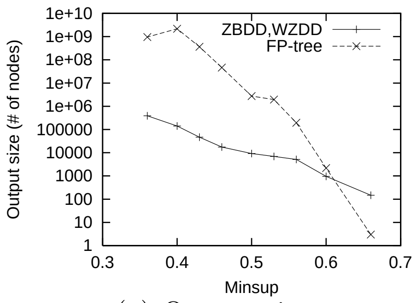


(a) With advanced pruning

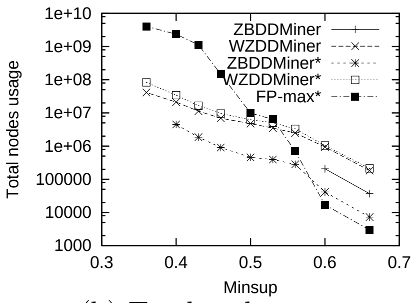


(b) Without advanced pruning

Figure 4.10: Runtime for mining MFIs in lung-cancer-750 data set (sparse)

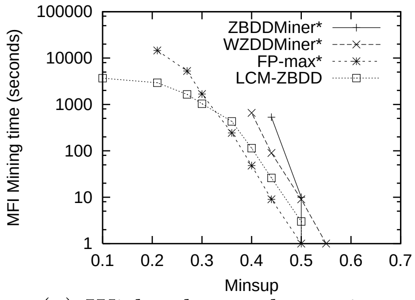


(a) Output size

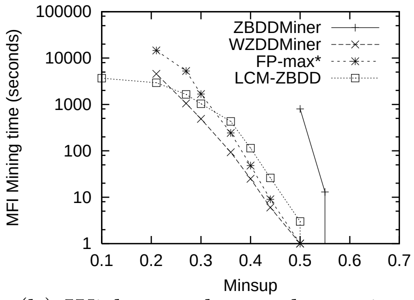


(b) Total node usage

Figure 4.11: Nodes used for mining MFIs in lung-cancer-750 data set (sparse)

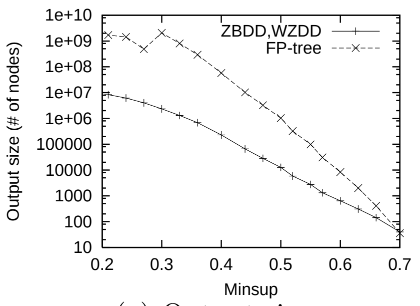


(a) With advanced pruning

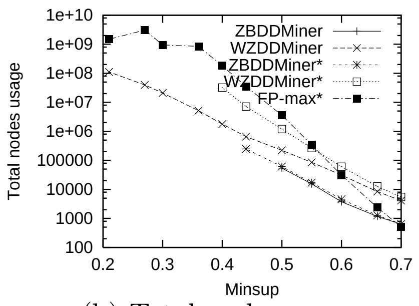


(b) Without advanced pruning

Figure 4.12: Runtime for mining MFIs in ALL-AML-100 data set (dense)

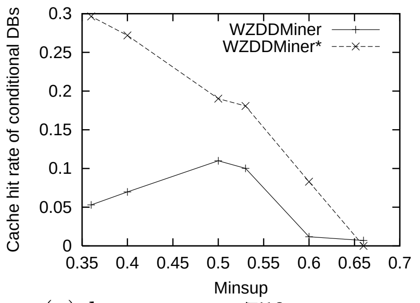


(a) Output size

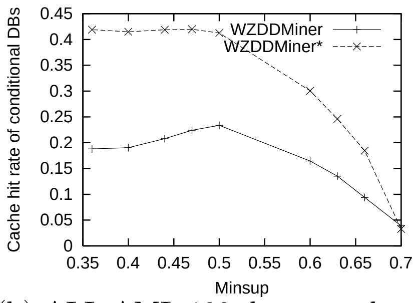


(b) Total node usage

Figure 4.13: Nodes used for mining MFIs in ALL-AML-100 data set (dense)

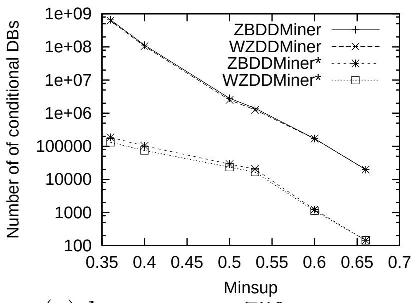


(a) lung-cancer-750, sparse

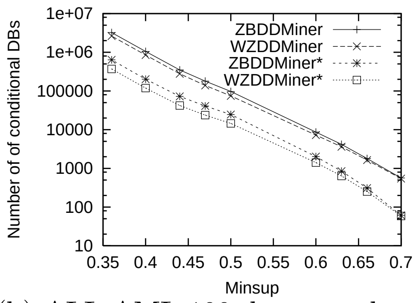


(b) ALL-AML-100 data set, dense

Figure 4.14: Cache hit rate of conditional databases in WZDD-based algorithms for mining CFIs

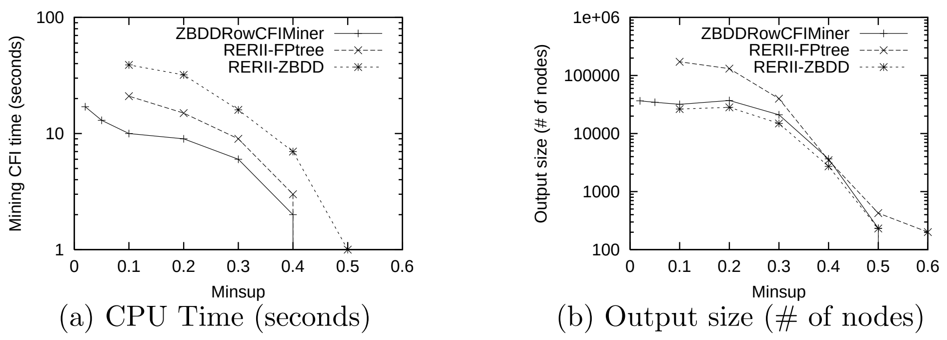


(a) lung-cancer-750, sparse



(b) ALL-AML-100 data set, dense

Figure 4.15: Number of conditional databases for mining CFIs



(a) CPU Time (seconds) (b) Output size (# of nodes)

Figure 4.16: Results from row-mining CFIs in lung-cancer-100 data set (sparse)

the intermediate structures used for mining them, by allowing the various databases to share nodes. This compression has been proven in our experimental results when mining MFIs and CFIs at a relatively low support threshold, for which the FP-trees had billions of overall nodes usage. In particular, in the dense dataset, WZDDs are able to achieve up to 500 times overall data compression over FP-trees, as shown in the total nodes usage comparison. In such a circumstance, many long patterns exist and many sub-trees may be shared across multiple databases (including the conditional DBs) which is not possible using FP-trees. On the other hand, in the sparse dataset in which not too many conditional DBs are being projected, ZBDDs allow a higher overall data compression than WZDDs, being able to share more nodes when representing the intermediate databases but they require the use of a *bitmap* for support counting as a tradeoff.

Furthermore, when ZBDDs are used for computing CFIs and the output data structure contain additional support-encoding variables, we found that although the size of the ZBDDs is increased, they can still contain fewer nodes than FP-trees when the support threshold is relatively low.

4.6.2 Pruning Effectiveness

Our techniques use a basic infrequent prefix pruning, and some advanced pruning strategies which are also used in existing algorithms. Firstly, let us consider the effectiveness of infrequent prefix pruning. FP-trees may allow earlier pruning of infrequent prefixes by allowing different item orderings to be used for different database projections. On the other hand, ZBDDs and WZDDs use static item ordering as a tradeoff for achieving data compression. This explains a number of situations in our experimental results when ZBDDs and WZDDs are less efficient than FP-trees, when mining at a high support threshold, in which a large search space can be pruned earlier by the FP-trees.

Secondly, the effectiveness of advanced pruning strategies rely upon the amount of search space reduction being achieved over the extra cost of performing the pruning routines. As part of the advanced pruning routines, our `ZBDDMiner*` algorithm uses a bit-wise-and operation, and `WZDDMiner*` traverses the databases, for support counting, which is an expensive computation to perform in high-dimensional datasets. Based on our experiments, ZBDDMiner* does improve the mining efficiency of `ZBDDMiner`, especially when the total size of the conditionalDBs is significantly reduced. The pruning routines in `WZDDMiner*` rely upon some library ZBDD routines which can reuse computation results, allowing efficient pruning when a large number of conditional DBs (or their substructures) are being shared as is the case when mining MFIs in dense datasets at low support threshold. Moreover, the effectiveness of the advanced pruning strategies is influenced by the closedness of the patterns. Due to this reason, without employing advanced pruning, `WZDDMiner` performs well in circumstances where the number of patterns is much larger than the number of CFIs.

4.6.3 Cached Computation Results

One powerful feature of ZBDDs is their caching principle. The ZBDD routines used in our framework allow the result from intermediate computations, to be cached. More particularly the WZDD-based framework allows the output from every conditional DB which has been proven useful. In dense datasets, or in sparse datasets with low support threshold, millions of long patterns exist and different prefix itemsets may project similar conditional DBs (and thus, similar patterns). This has proven a significant time improvement by WZDDs over FP-trees, despite the use of static item ordering which may hurt its efficiency as we have discussed earlier. In particular, our experimental results show the WZDD-based technique (without advanced pruning) outperforms the FP-tree based technique (with advanced pruning) when mining huge volume of MFIs (indicated by the number of nodes in the output data structures) at low support thresholds, and it has similar time performance as the FP-tree based technique for mining CFIs in the relatively denser dataset.

4.7 Summary of Results

We now return to the questions which were posed at the beginning of the paper:

1. *Does the canonical property of ZBDDs allow an efficient and scalable algorithm for frequent itemset mining to be developed ?*

   As we have seen in our experimental results, the WZDD based algorithm is superior over both ZBDDs and FP-trees for mining MFIs at a low support threshold, or mining CFIs in a dense dataset. In such a circumstance, millions of long patterns exist, and hence, numerous auxiliary DBs are induced and their canonical WZDD representations (which are substantially smaller than FP-trees) allow mining to be performed efficiently through re-using intermediate results.

2. *How much data compression can a ZBDD achieve compared to an FP-tree ?*

   (a) When mining CFIs, the total size of non-weighted ZBDDs used throughout mining may be larger than the FP-trees, this being a result of having the extra support encoding variables on the output for closed frequent itemsets. They, however, could achieve slightly higher overall data compression by a factor of 2 compared to FP-trees, as shown in our results when mining CFIs at low support in dense datasets.

   More specifically, for storing the output closed frequent itemsets, ZBDDs with support-encoding variables increase the size of the traditional ZBDDs (without support-encoding variables) by 100 times, yet, they may contain fewer nodes than FP-trees when mining CFIs at a low support threshold.

   (b) When mining MFIs, non-weighted ZBDDs are able to achieve further overall data compression up to 100 times more compact than WZDDs, as found in our experiments. This shows WZDDs achieve lower data compression than ZBDDs due to their weighted edges. However, WZDDs are still much more compact than FP-trees for representing all of the databases created throughout mining, more particularly if the support threshold is not too high, as shown in our results that they used up to 1000 times fewer nodes than the FP-trees

3. *Does the use of a more compact data structure really mean that mining is more efficient?*

Not always. In situations where the intermediate databases are highly compressed and the total number of nodes is much less than FP-trees, intermediate results can be reused effectively which increases mining speedups over FP-trees. Such situations were found in our experimental results when mining MFIs/CFIs in a dense dataset. On the other hand, if the WZDD data representations were of similar size than FP-trees, the FP-tree based techniques are more efficient as they use a more flexible variable ordering which allows earlier search space pruning.

Moreover, although the ZBDD representations in the intermediate computations are often smaller than the WZDDs, based on our findings, it does not always mean a more efficient mining was obtained. This can be explained by the extra cost of using a secondary data bitmap with (non-weighted) ZBDDs, which may require expensive computation in high dimensional datasets.

4.8 Related Work

We are aware of several other works which use ZBDDs for itemset mining [113, 116]. Work in [116] demonstrated that ZBDDs are useful for mining patterns in sparse high dimensional amino acid datasets. Such sparse data sets contain only a few rows, but a large number of items, which contain a large number of patterns. They studied the mining of item symmetry using ZBDDs, which are useful for providing a succinct representation of the patterns.

Work in [113, 112] proposed a ZBDD-based pattern growth mining of frequent itemsets, which uses a different data representation to our proposed technique. Its optimised variant for mining closed frequent itemsets is proposed in [114]. Their ZBDD representation of the databases encode the support values by storing the itemsets in multiple ZBDD functions based on their binary support values, whose representations are referred as *tuple histograms*. Their experiments show their technique outperforms FP-growth [70] for mining the traditional type of datasets when huge volume of patterns exist. Such a representation is less compact and requires more complex routines to construct, instead of the simpler, and faster, basic ZBDD routines used in our framework. A review of their technique was given in Chapter 3. Therefore, their technique does not seem scalable for mining the more challenging microarray datasets which have exponentially large search space. Furthermore, we have shown our technique can be adapted to the row-wise mining framework.

More recent work in [76] proposed a method for choosing a good variable ordering for ZBDDs in data mining applications. The method computes the variable ordering after the ZBDD has been built, and re-arranges the variables. This method is different from ours, since we decide the variable ordering prior to constructing the ZBDDs. Our variable ordering heuristics aim to achieve an efficient mining of frequent itemsets, as well as to achieve an overall compact data representation across multiple shared ZBDDs, which represent the intermediate data structures used throughout mining. Their method, on the other hand, finds a variable ordering which is optimised for a particular ZBDD.

A vast number of other techniques for mining frequent itemsets have been proposed. The survey can be found in the FIMI Repository [78]. Tree data structures are widely used in many techniques in the FIMI repository, such as FP-growth [70], AFOPT [94], CLOSET+ [158], and PatriciaMine [132]. None of these techniques use a data structure which allows sharing across the auxiliary DBs, which is a key feature of our ZBDD-based technique.

CLOSET+ [158] is an FP-tree-based algorithm, which is designed for closed itemset mining. It explores not only the itemset space, but the combinations of itemset and row-id space to find the closed frequent itemsets.

AFOPT [95] introduced a combination of trees and arrays for representing dense and sparse conditional databases, respectively. For a very small conditional database, it uses a *bucket counting* technique [1] which counts the frequency for every combination of the items instead of employing a divide-and-conquer mining. The AFOPT tree [94] is an FP-tree-like structure with an inverted item ordering, i.e. ascending frequency ordering having the least frequent item at the top. Like the item ordering in the FP-growth algorithm, the items in the AFOPT-tree are visited least-frequent-first, so that long-narrow or short-wide conditional trees are projected. Hence, the AFOPT-tree is traversed top-down, which requires extra computations for for constructing and maintaining parent links from each node.

Although prefix sharing is less likely to occur in an AFOPT-tree than that in an FP-tree, the AFOPT-tree saves much space from not storing parent links (statistics are shown in [95]). Moreover, the AFOPT algorithm has adopted the pruning techniques for mining maximal frequent itemsets and able to outperform FP-max* for mining short patterns in sparse data sets. It has a worse performance than FPmax*, though, for mining long patterns in dense datasets.

PatriciaMine [132] optimises the prefix-growth algorithm by using a Patricia trie as data structure. It merges successive items which have the same frequency in each path, which makes it able to achieve a more compressed structure. However, the trie structure is expensive to compute. PatriciaMine performs well for mining patterns in a sparse data set. For dense data sets, which require numerous database projections, the construction of the conditional databases is a bottleneck of this algorithm.

4.9 Summary

In this chapter, we have examined the use of advanced data structures, Zero suppressed BDDs (ZBDDs), for mining (maximal/closed) frequent itemsets. We also identified situations where they are superior over state of the art techniques. Overall, we found that ZBDDs allow much higher data compression for storing huge volume of long patterns as well as the intermediate structures used in mining them. We also introduced a weighted ZBDD, which is able to improve mining efficiency than the traditional ZBDD. Although our algorithm is not uniformly superior than the other algorithms, it allows more efficient mining in dense high dimensional datasets at low support thresholds. We believe this result suggests that ZBDDs can serve as a very valuable tool in data mining.

Chapter 5

Mining Frequent Subsequences Using Weighted Sequence Binary Decision Diagrams

In this chapter, we study an important pattern mining task in sequential databases. In particular, we study the mining of frequent subsequences which is a fundamental type of sequential pattern. Given a minimum frequency threshold, it is defined as a frequently occurring ordered list of events or items, where an item may be a web page in a web access sequence, or a nucleotide in a DNA sequence. There are many useful applications, including analyzing time-stamped market basket data, finding web access patterns (WAP) from web logs [130], finding relevant genes from DNA sequences [101] and classifying protein sequences [54, 143].

5.1 Introduction

Frequent subsequence miners must explore an exponentially sized search space [129], which can be computationally expensive when a large number of long frequent subsequences exist, such as in DNA or protein sequence data sets [135, 55, 19], which have a small alphabet. or in other data sets at a low minimum support [130]. Popular techniques for mining frequent subsequences such as those in [129, 47], adopt a *pattern growth* approach which is thought to be the most efficient category of approaches for sequential pattern mining. The generation of infrequent candidate patterns is avoided by projecting the database into smaller conditional databases recursively.

However, such a technique can be computationally expensive due to the following factors: 1) Projecting numerous conditional databases takes considerable time as well as space, especially since item reordering optimizations do not make sense for sequential mining, like they do for frequent itemset mining. 2) Each conditional database is processed independently, although they often contain some common sub-structures.

Highlighted in the previous chapter, BDDs can achieve greater data compression than prefix trees, since node fan in (suffix sharing) as well as node fan out (prefix sharing) is allowed in a BDD. Complementing their compactness, BDDs are also engineered to enforce reuse of intermediate computation results. The use of BDDs in a pattern growth framework has been shown useful for mining frequent itemsets, more particularly, for compressing and efficiently manipulating the numerous conditional databases. Instead of a prefix tree, we will demonstrate how a directed acyclic graph (DAG), such as a Binary Decision Diagram (BDD) [25], can be used as an alternative data structure for mining frequent subsequences. We focus on mining a large number of long subsequences efficiently, in terms of both time and space.

PrefixSpan [129], is one of the best algorithms for mining frequent subsequences. Its optimisation technique completely avoids physically constructing database projections. The benefit of using a tree data structure for efficiently mining frequent subsequences is questioned in [129]. The compactness of a prefix tree relies on the existence of common prefix-paths, which cannot be fully exploited in the sequence context. Recent work [47] proposed a tree based PLWAP algorithm for mining sequential web access patterns in weblogs, using a WAP-tree data structure which allows common prefixes to share nodes in the tree. The PrefixSpan is useful for mining long subsequences which occur rarely in the database, whereas PLWAP is more useful for mining short but many subsequences. A review of these techniques can be found in Section 2.4.2 (Chapter 2).

5.2 Contributions

In this chapter, we introduce a mining technique based on the use of a special kind of BDD, which can provide an overall compressed representation of the intermediate databases. In contrast to previous thinking, our technique can benefit from the explicit construction of the BDD by relying on sophisticated techniques for node sharing and caching. It is not straightforward to efficiently encode sequences into BDDs, however.

To this end, we introduce an original variant, namely the **Sequence Binary Decision Diagram (SeqBDD)**, which is suitable for compactly representing sequences and efficiently mining frequent subsequences. Based on the canonical property of SeqBDDs, multiple databases may share sub-structures, which in turn allows high overall data compression to be achieved and redundant computations to be avoided. The questions which we address in this research are:

- *Can sequences be compactly represented using a BDD?*
- *Can the use of a BDD benefit frequent subsequence mining?*
- *Can a BDD-based frequent subsequence mining technique outperform state-of-the-art pattern growth techniques?*

To summarise, our contributions in this chapter are three-fold:

- We introduce a compact and canonical DAG data structure for representing sequences, which we call a **Sequence Binary Decision Diagram (SeqBDD)**, and its weighted variant which allows the frequency (or support) of the sequences to be represented. Unlike prefix-trees, SeqBDDs allow node fan-out as well as fan-in, which is a novel data representation in the sequence mining literature.
- We show how a weighted SeqBDD can be used for mining frequent subsequences in the prefix-growth framework. In particular, node-sharing across multiple databases is allowed, and BDD primitives which promote re-use of intermediate computation results are adopted in the mining procedure, allowing high data compression and efficient mining to be achieved. A novel feature of our mining technique is its ability to avoid duplicating similar conditional databases which are projected by different prefixes.
- We experimentally investigate the behavior of our technique for finding frequent subsequences in biological data sets, which have a small alphabet domain, in weblog data sets, and several synthetic data sets. We compare its performance against the competitive pattern growth algorithms, PLWAP [47] and PrefixSpan [129]. SeqBDDMiner is proven to be superior when the input sequences are highly similar, or when the minimum support threshold is low, where the conditional databases share many common sub-structures.

5.3 Overview of ZBDD-based Representations of Sets of Sequences

Directly representing sets of sequences as ZBDDs is not possible, since ZBDDs do not allow a variable to appear multiple times in any path. A natural way to encode a sequence is by introducing variables which encode each item and its position in the sequence. In the following discussion, to differentiate the original sequential representation from its itemset representation, we refer to items in the original domain as alphabet letters.

Let L be the maximum length of a sequence S, and N be the size of the alphabet domain A. As a naive encoding, S can be expressed as an itemset over a new domain I' containing $L \times N$ items, each item represents an alphabet letter at a particular position in the sequence. Another encoding requires $L \times log_2 N$ items [80], which may be more efficient if the sequences contain many elements. We will give more details about each encoding shortly. We refer to the ZBDD representations based on the alternative encodings as ZBDD$_{naive}$, and ZBDD$_{binary}$, respectively.

5.3.1 Naive Encoding Scheme

Let x_i be an item in I', where $i \in [1 \dots L]$, and $x \in A$. Item x_i in the itemset representation of a sequence S represents the occurrence of letter x at the i'th position in S.

Example 20. *Suppose we have an alphabet domain $A = \{a, b, c\}$, and the maximum length of the sequences is 5. The itemset domain contains 15 items. Let D be a sequence database containing p_1 = aabac, p_2 = baba, p_3 = aaca, p_4 = bbac. The itemset representation of p_1 is $\{a_1, a_2, b_3, a_4, c_5\}$. Figure 5.1 shows the itemset encoded database.*

5.3.2 Binary Encoding Scheme

The binary encoding scheme represents each alphabet letter by $n = \lceil log_2(N+1) \rceil$ binary variables. Suppose there are 3 letters, a, b, and c, they are represented using 2-bit binary variables v_1, v_0, such that $(v_1, v_0) = (0, 1)$ represents a, $(1, 0)$ represents b, and $(1, 1)$ represents c. At a given position $i \in [1 \dots L]$ in the sequence, and $j \in [0, \dots, n]$,

Sequence	Itemset Encoded
$p_1 = aabac$	a_1, a_2, b_3, a_4, c_5
$p_2 = baba$	b_1, a_2, b_3, a_4
$p_3 = aaca$	a_1, a_2, c_3, a_4
$p_4 = bbac$	b_1, b_2, a_3, c_4

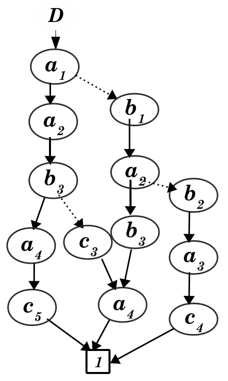


Figure 5.1: Sequence to itemset translation using Naive encoding, and the resulting ZBDD representation of the sequence database

Symbol	Binary Code (v_1, v_0)	Itemset Encoded
a	$(0, 1)$	$x_{j.0}$
b	$(1, 0)$	$x_{j.1}$
c	$(1, 1)$	$x_{j.1}x_{j.0}$

Sequence	Itemset Encoded
$p_1 = aabac$	$x_{1.0}, x_{2.0}, x_{3.1}, x_{4.0}, x_{5.1}, x_{5.0}$
$p_2 = baba$	$x_{1.1}, x_{2.0}, x_{3.1}, x_{4.0}$
$p_3 = aaca$	$x_{1.0}, x_{2.0}, x_{3.1}, x_{3.0}, x_{4.0}$
$p_4 = bbac$	$x_{1.1}, x_{2.1}, x_{3.0}, x_{4.1}, x_{4.0}$

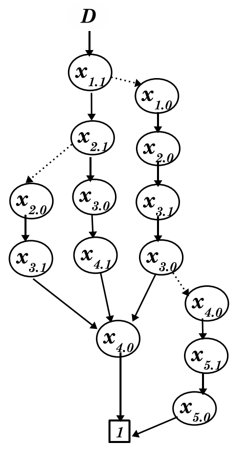


Figure 5.2: Sequence to itemset translation using binary coding, and the resulting ZBDD representation of the sequence database

item $x_{i.j}$ encodes the binary coding of a sequence element $(v_n, \ldots, v_0)$ such that $v_j = 1$. Figure 5.2 shows the encoded database D and its ZBDD representation.

5.3.3 Compactness of ZBDD Representations

The work in [80] uses a lexicographic variable ordering for the ZBDD. Other variable orderings may be used, but it does not have much influence on the compactness of the ZBDD, due to the position-specific item representation. The binary encoding allows more flexible node sharing between different alphabet representations, at the cost of using more nodes in each path than the naive encoding. In the context of sequential patterns, the ZBDDs are used to store frequent subsequences, which may occur in different positions in various sequences in the database. Using the position-specific itemset encoding, there is no quick way to identify the occurrence of a subsequence in several sequences. Based on this limitation, we next propose a new type of BDD which is more suitable, and more compact, for representing sequences.

5.4 Weighted Sequence Binary Decision Diagrams

We have shown in the previous section that Zero-suppressed Binary Decision Diagrams have limitations for representing sequences. In this section we will describe our proposed data structure, namely the **Sequence Binary Decision Diagram** or SeqBDD for short, and its weighted variant which allows more efficient manipulation of sequences.

In Sequence Binary Decision Diagrams (SeqBDDs), only the 0-child nodes are ordered with respect to their parent nodes, and a variable is allowed to occur multiple times in a path, which is not possible using any of the existing BDD variants. Analogous to ZBDDs, both the node-merging and the zero-suppression rules are employed in SeqBDDs, which allow a compact and canonical graph representation of sequences.

In order to use SeqBDDs for mining sequential patterns such as frequent subsequences, the frequency of the sequences need to be represented in the database. Inspired by the weighted-variant of ZBDD for frequent itemset mining [98], we introduce **Weighted Sequence Binary Decision Diagrams (Weighted SeqBDDs)**.

5.4.1 Sequence Binary Decision Diagrams

In SeqBDD semantics, a path in a SeqBDD represents a sequence, in which the nodes are arranged in-order to the positions of their respective variables in the sequence. More specifically, the top node corresponds to the head of the sequence, and the successive 1-child nodes correspond to the following elements, respectively. A SeqBDD node $N = node(x, N_1, N_0)$ denotes an internal node labeled by variable x, and N_1 (resp. N_0) denotes its 1-child (resp. 0-child). The label of node N has a lower index (appear earlier in the variable ordering) than the label of N_0. We denote the total number of descendant nodes of node N, including N itself, by $|N|$.

Let x be an item and P and Q be two sets of sequences. We define an operation $x \times P$ which appends x to the head of every sequence in P, and a set-union operation $P \bigcup Q$ which returns the set of sequences which occur in either P or Q.

Definition 17. *A SeqBDD node $N = node(x, N_1, N_0)$ represents a set of sequences S such that $S = S_{\overline{x}} \bigcup (x \times S_x)$, where S_x is the set of sequences which begin with element x (with the head elements being removed), and $S_{\overline{x}}$ is the set of sequences which do not begin with x. Node N_1 represents set S_x, and node N_0 represents set $S_{\overline{x}}$.*

Moreover, every sequence element in the SeqBDD can be encoded into a set of binary variables, using a similar binary encoding scheme used in ZBDD$_{binary}$ which was discussed in Section 5.3.2. However, each path is much likely longer, but the improvement in terms of node sharing is relatively small. Thus, in most cases, the database representation with the binary encoding id less compact. In the following discussion, we will compare the compactness of SeqBDDs with ZBDDs under the naive itemset encoding.

Compactness of a SeqBDD: By removing the variable ordering between each node and its 1-child, the SeqBDD's merging rule allows common suffix-paths between sequences to share nodes, regardless of their length. This is something which is not possible in the ZBDD representations. We employ a lexicographic variable ordering for the SeqBDD. Other variable orderings may be used, however, this would not have much influence on the compactness, since, it is only employed partially.

The following theorems state the compactness of SeqBDDs relative to ZBDDs and prefix trees, where the size is proportional to the number of nodes.

Theorem 6. *Given a sequence database, the SeqBDD, ZBDD, and prefix tree representations satisfy the following relation: SeqBDD $\leq$ ZBDD $\leq$ Prefix tree, where SeqBDD is the most compact, and the sequences in the ZBDD are encoded by the naive itemset encoding.*

Proof. The proof for this theorem follows from Lemma 3. □

Lemma 3. *Given a sequence database, nodes in the ZBDD have a many-to-one relationship with the SeqBDD, and nodes in the prefix tree have a many-to-one relationship with the ZBDD.*

Proof. Since the merging rule criteria affects either the ZBDD or the SeqBDD only if there is some common suffix between the sequences, assume there are two non-identical sequences in the input database, p and q, which share a common suffix namely *suffix*. Let P_Z and Q_Z be two suffix paths in the ZBDD which correspond to *suffix* in p and

q, respectively, P_{Seq} and Q_{Seq} be corresponding paths in the SeqBDD, and P_{tree} and Q_{tree} be the corresponding paths in the prefix tree.

If $|p| \neq |q|$, the suffixes of p and q are represented using different sets of variables and P_Z and Q_Z can be mapped to P_{tree} and Q_{tree} respectively. If $|p| = |q|$, however, P_Z and Q_Z are merged, and both P_{tree} and Q_{tree} correspond to one path in the ZBDD.

Regardless of the length of p and q, the suffix *suffix* is encoded similarly in both sequences and P_{Seq} and Q_{Seq} are merged, which shows the many-to-one mapping between the ZBDD nodes to SeqBDD. □

5.4.2 Weighted Sequence Binary Decision Diagrams

A Weighted SeqBDD is a SeqBDD with weighted edges. Every edge in a weighted SeqBDD is attributed by an integer value, and each internal node's incoming edge corresponds to the total frequency of all sequences in that node. Thus, the weight of the incoming link is monotonically decreasing as the node is positioned lower in the structure. We define a Weighted SeqBDD node by the following definition.

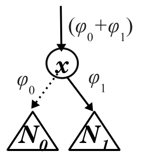
Definition 18. *A Weighted SeqBDD node N is a pair of $\langle \varphi, \vartheta \rangle$ where φ is the weight of N, and ϑ is a SeqBDD node.*

The weight of node N, i.e. φ, represents the weight of the incoming link to N. For a node N, we define a function $weight(N) = \varphi$, which gives the total frequency of the sequences in N. If N is an internal node, N_0 and N_1 correspond to two partitions of the database, and $weight(N)$ is the sum of the total frequencies of the sequences in N_0 and in N_1:

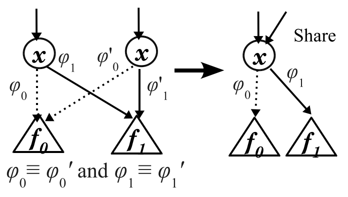
$$weight(N) = weight(N_0) + weight(N_1) \tag{5.1}$$

Two nodes in a Weighted SeqBDD are merged only if they have the same label, the same respective child-nodes, and also the same weights on the outgoing edges respectively. Hence, a Weighted SeqBDD may be less compact than the non-weighted SeqBDD, since two nodes which contain similar sequences cannot be merged in the Weighted SeqBDD if their respective frequencies are different. Figure 5.3 illustrates a weighted SeqBDD node and the merging rule for weighted SeqBDDs.

Example 21. *Consider a set of sequences, with their respective frequencies, $S = \{aaa : 3, aba : 2, bc : 2, bbc : 2\}$. Figure 5.4(a) shows an example of a Weighted SeqBDD representation of S. Assuming a lexicographic ordering, the prefix-path a is shared*



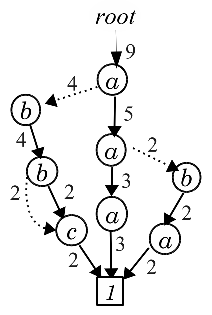
(a) Node $N = \langle \varphi, \vartheta \rangle$;
$\vartheta = \text{node}(x, N_1, N_0)$; $\varphi = \varphi_1 + \varphi_0$.



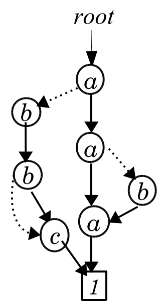
x
φ_1
φ'_0
φ_0
φ'_1
f_0
f_1
$\varphi_0 \equiv \varphi_0{}'$ and $\varphi_1 \equiv \varphi_1{}'$
Share

(b) Weighted SeqBDD's Merging Rule applies only if $\varphi_0 \equiv \varphi'_0$ and $\varphi_1 \equiv \varphi'_1$

Figure 5.3: Illustration of Weighted SeqBDDs



(a) Weighted SeqBDD



(b) SeqBDD

Figure 5.4: Weighted SeqBDD vs SeqBDD for a set of sequences $S = \{aaa : 3, aba : 2, bc : 2, bbc : 2\}$, using a lexicographic variable ordering

between sequences aaa and aba, and prefix b is shared between sequences bc and bbc. The lower node representing suffix c is shared between sequences bc and bbc, but the bottom a-nodes are not merged because they have different frequencies. As a comparison, the SeqBDD for the same sequences, being 1 node smaller, is shown in Figure 5.4(b).

Monotonic property of weights in a Weighted SeqBDD: When a Weighted SeqBDD is used for storing subsequences, we can use the weights to efficiently prune sequences which do not satisfy the minimum support constraint due to the monotonic property of the weight function. In particular, its monotonicity is described by the following theorem.

Lemma 4. *Given a weighted SeqBDD node N, weight$(N_0) \leq$ weight(N) and weight$(N_1) \leq$ weight(N)*

Proof. The proof for this lemma is straightforward from Equation 5.1. Since the weight of each node is a positive integer, and it is the sum of the weights of its child nodes, i.e. N_0 and N_1, the weight of a node is no smaller than the weight of its child nodes. □

Based on Lemma 4, if the weight of a node N is less than the minimum support, node N and its child-nodes correspond to infrequent sequences. Thus, they can be safely removed and replaced by the sink-0 node. Being monotonic, the weights in Weighted SeqBDDs may be used for representing other monotonic functions, other than the total frequency which we have defined, such as the maximum (or minimum) length of the sequences.

Weighted SeqBDD primitive operations: Basic ZBDD operations (listed in Table 4.1) such as *set-union* and *set-subtraction* can be adapted for Weighted SeqBDDs (and for the general SeqBDDs). In the following discussion, we show how the weight function can be integrated with ZBDD's set-union operation. When a sequence co-exists across the input SeqBDDs, its frequencies are added. If both of its inputs are sink-1 nodes, the output is also a sink-1 with the weights of the input nodes being added. Moreover, the weight of each node in the output is computed using Equation 5.1. Below, we show that the operation is correct, with references made to the corresponding lines in the SeqBDD's `addSeqBDD`() procedure shown in Algorithm 5.1.

Theorem 7. Correctness of the `addSeqBDD()` procedure: *Given two Weighted SeqBDDs P and Q, the `addSeqBDD`(P, Q) correctly adds the sequences in P and Q and their corresponding frequencies.*

Proof. Consider the following cases:

1. If P is a sink-0, i.e. P is empty, the output consists of all sequences in Q. Similarly, if Q is a sink-0, P is returned as output (line 1-2).

2. If both P and Q are sink-1 nodes, i.e. each of P and Q consists of an empty sequence, then the output also consists of an empty sequence represented by a sink-1 node with weight of the total frequency between P and Q (line 5).

3. If both P and Q are internal nodes with labels x and y.

   (a) Suppose $x = y$ (line 7-8). Since x is the lowest indexed item among the head item of all sequences in P and Q, x should be the label of the output node. The 1-child of the output node should contain all sequences which begin with x, which are present in P_1 and Q_1, and the 0-child of the output node contains the remaining sequences.

   (b) Suppose x has a lower index[1] than y (line 9-10), x has a lower index than the head of all sequences in Q. Therefore, the output node should be labeled with

[1]A sink-1 is considered an internal node with the highest index.

Algorithm 5.1 `addSeqBDD`(P, Q)

```
Input: $P$ and $Q$ are the input weighted SeqBDDs, each of which represents a set of sequences
Output: $\langle Z.weight, Z\rangle$ : the weighted SeqBDD containing set-union between $P$ and $Q$
  Procedure:
 1: case $P$ is a sink-0 node : return $Z = Q$
 2: case $Q$ is a sink-0 node : return $Z = P$
 3: case $Q = P$ is a sink-1 node :
 4:     Calculate weight: $Z.weight = weight(P) + weight(Q)$
 5:     return $\langle Z.weight, 1\rangle$
 6: Let $Z$ = node($Z.var$, $Z_1$, $Z_0$), $P$ = node($x$, $P_1$, $P_0$), $Q$ = node($y$, $Q_1$, $Q_0$)
 7: case $x = y$ :
 8:     $Z.var = x$ ; $Z_1$ = addSeqBDD($P_1$, $Q_1$) ; $Z_0$ = addSeqBDD($P_0$, $Q_0$)
 9: case $x$ has a lower index than $y$ :
10:     $Z.var = x$ ; $Z_1 = P_1$ ; $Z_0$ = addSeqBDD($P_0$, $Q$)
11: case $x$ has a higher index than $y$ : return addSeqBDD($Q$,$P$)
12: Calculate weight: $Z.weight = weight(Z_0) + weight(Z_1)$
13: return $\langle Z.weight, Z\rangle$
```

x, and all sequences which begin with x are present in P_1. The remaining output sequences exist in P_0 and Q.

(c) Suppose x has a higher index than y (line 11). This condition is opposite to condition 3(b), since the operation is commutative.

In each of the above cases, any sequence which co-occurs in P and Q is identified when the sink-1 nodes are found. Hence, when a co-occurring sequence is found, its total support is computed bottom-up beginning with the sink nodes, followed by the parent nodes respectively (line 12). □

Properties of Weighted SeqBDDs: A weighted SeqBDD is similar to a prefix tree, except that identical sub-trees are merged. Table 5.1 shows a comparison between their structural properties. Firstly, Weighted SeqBDDs store the frequencies as edge-weights, and they do not have side-links which are needed for finding database projections in the prefix trees. The caching mechanism in Weighted SeqBDDs allows the cost of projecting the databases to be reduced. Moreover, weighted SeqBDDs are similar to prefix trees in a sense that node fan-out is allowed, but node fan-in is only allowed in weighted SeqBDDs.

An example of a Weighted SeqBDD and a prefix tree, presented as a PLWAP tree [47], is shown in Figure 5.5, both of which represent the initial database containing sequences $\{aabac, baba, aaca, bbac\}$. The weighted SeqBDD contains 4 fewer nodes than Prefix tree through merging of the lower a-c nodes between sequences $aabac$ and $bbac$,

Property	Weighted SeqBDD	Prefix Tree
Support storage	As edge weight	In every node
Side-links	No	Yes
Fan-out	Yes	Yes
Fan-in	Yes	No

Table 5.1: Comparison between Weighted SeqBDD and Prefix Tree

Data Structure	Zero-suppressed	Weighted Edges
SeqBDD	Yes	No
Weighted SeqBDD	Yes	Yes
Weighted ZBDD	Yes	Yes
BMD	No	Yes
FreeBDD	No	No

Data Structure	Variable Ordering
SeqBDD	A global variable ordering is enforced between each node and its 0-child
ZBDD	A global variable ordering
BMD	A global variable ordering
FreeBDD	A variable ordering is enforced on the 1-child nodes in each path, different paths may have different variable orderings

Table 5.2: Comparison between (Weighted) SeqBDD and other BDD Variants

and the lower *a*-node between sequences *baba* and *aaca*.

Table 5.2 shows a comparison between a Weighted SeqBDD and the other types of BDD. The zero-suppression rule is employed in ZBDDs and SeqBDDs, and their weighted variants [98]. BMDs [27] also have weighted edges, but they use a different weight function. In terms of variable ordering, SeqBDDs employ a global ordering which is applied only upon the 0-child nodes, whereas FreeBDDs [59] allow different paths to have different orderings. The other BDDs apply a global ordering between each node and both of its child nodes.

Table 5.3 shows some statistics of the weighted SeqBDD and ZBDD representations for three real data sets (description of these data sets is given in Section 5.6), when used for representing the frequent subsequences and their frequency values. For *yeast.L200* data set, the size of the weighted SeqBDD (W-SeqBDD) is 456 for storing 475 sequences with an average length of 2.9, which is 25% smaller than the Weighted ZBDDs (WZBDDs), labeled as WZBDD$_{naive}$ and WZBDD$_{binary}$. For the other two data sets, which contain more and longer sequences, W-SeqBDD is more compact than WZBDD$_{naive}$ (i.e. achieving 60% compression for *snake*, and 10% compression for

Dataset	minsup	Num. of subsequences	Average length
yeast.L200	0.8	475	2.9
snake	0.3	2616	5
davinci	0.001	999	5

Dataset	|W-SeqBDD|	$\|\text{WZBDD}_{naive}\|$	$\|\text{WZBDD}_{binary}\|$
yeast.L200	456	632	644
snake	1007	3293	4451
davinci	1030	1165	3331

Table 5.3: Characteristics of the represented subsequences, and size comparison between weighted SeqBDD (W-SeqBDD), WZBDD$_{naive}$, and WZBDD$_{binary}$, in terms of the number of nodes

davinci which has a larger alphabet). For all three data sets, the WZBDD$_{binary}$ is smaller than WZBDD$_{naive}$, with the *davinci* data set having the largest difference due to the large alphabet size.

Complexity analysis: In the following discussion, we discuss the computational complexity for creating and manipulating nodes in the general SeqBDDs, which is also applicable to the Weighted SeqBDDs. The computational cost for creating a SeqBDD node requires one look-up operation to the *uniquetable*, which is the same as that in a BDD.

Theorem 8. *The time complexity for creating a SeqBDD that consists of N nodes is $O(N)$.*

Proof. When creating a node, the hash key for that node is computed and one lookup operation is performed to find a pre-existing entry in the unique table. Based on the existing hash table implementation in JINC [123], an insertion or lookup operations has $O(1)$ time complexity. Thus, when creating N nodes, the overall time complexity is $O(N)$. □

Theorem 9. *Given a database containing k sequences with a maximum length of L. The number of nodes in the SeqBDD representation is bounded by kL, and the height of the SeqBDD is bounded by L.*

Proof. In the given database, there are kL elements in total. In the worst case, each of those elements is represented by a node in the SeqBDD, thus, the SeqBDD consists of kL unique nodes. The height refers to the maximum length of any path in the SeqBDD, which is L, which refers to the maximum length of the input sequence since each path corresponds to a unique sequence. □

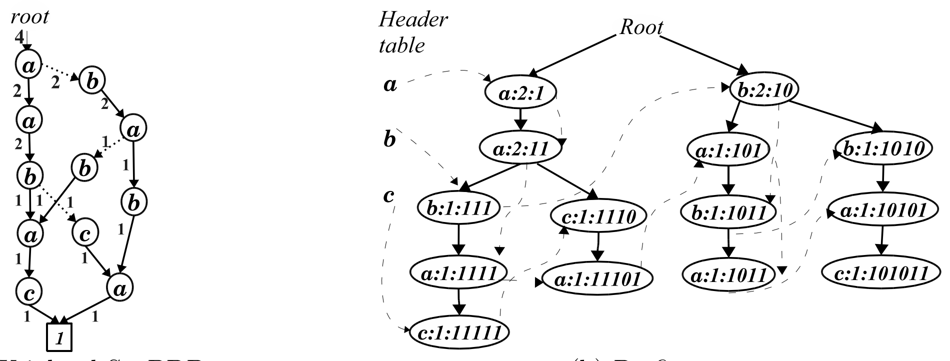
Note: in practice, the number of nodes is much smaller than kL, since many nodes in a SeqBDD which correspond to common prefixes, or common suffixes, may be shared across multiple sequences. Since the number of nodes of the SeqBDD is $O(kL)$, hence, its construction has $O((kL))$ time complexity.

The basic operations, such as set-union and set-subtraction, have polynomial time-complexity with respect to the number of nodes in the input SeqBDDs. Our algorithm for mining frequent subsequences employ the primitive SeqBDD's `addSeqBDD` operation which adds two sets of sequences and combines their frequencies. The complexity analysis of such operation follows.

Theorem 10. *Given two SeqBDDs P and Q, and a binary operation* <op>*(P, Q), where* <op> *is a primitive operation. The time complexity of* <op>*(P,Q) is* $O(|P| + |Q|)$.

Proof. The primitive operations visit each node from the two input nodes at most once. Hence, the number of recursive calls is $O(|P| + |Q|)$. Since in each recursive call, a new node may be created, the total number of unique nodes (including the input and the output nodes) is $O(|P| + |Q|)$. In the worst case, no nodes are merged between the input and the output SeqBDDs. Hence, the overall time complexity for the operation is $O(|P| + |Q|)$. □

Caching mechanism: To reduce computational cost, the SeqBDDs employ the same caching mechanism as in the BDDs, and in our implementation, we use the caching library of JINC [123]. There exist cache-conscious prefix trees [60] for mining frequent itemsets, which are designed to minimise access to physical memory. On the other hand, BDD's caching mechanism has a different aim, that is to make similar intermediate computations to re-use the computed results. It is also automatically provided by the BDD library implementation, and it is not explicitly shown in the algorithm described shortly.



(a) Weighted SeqBDD (b) Prefix tree

Figure 5.5: Weighted SeqBDD vs Prefix tree for a data set containing sequences $\{aabac, baba, aaca, bbac\}$, using a lexicographic variable ordering.

5.5 Frequent Subsequence Mining Algorithm Using Weighted Sequence BDDs

We call our algorithm for mining frequent subsequences `SeqBDDMiner`. `SeqBDDMiner` follows a prefix growth mechanism similar to PrefixSpan in [129], but uses a Weighted SeqBDD for representing the database. In the remainder of the paper, unless otherwise stated, we refer to the Weighted SeqBDD by its general term SeqBDD.

Unlike PrefixSpan, our algorithm physically creates the conditional databases but this benefits mining, since the SeqBDD allows nodes to be shared across multiple databases, and allows the results to be shared between similar intermediate computations. Using the SeqBDD's caching principle, moreover, construction of the conditional databases can be performed efficiently.

The initial SeqBDD representation of the database is built by *add*-ing the input sequences. The construction procedure is shown in Algorithm 5.2. For an item x, the x-conditional database contains suffixes of the first occurrence of x from each sequence in the input database. We find an *f-list* for each item x, which is an ordered list of elements which are frequent in the conditional database. This list allows early pruning of the conditional database by removing items which do not appear in the *f-list*. For efficiency purpose, the ordering in this list is inverted from the SeqBDD's global variable ordering, which allows the output node to be built bottom-up incrementally.

For finding the frequency of an item, we define a SeqBDD-based operation which follows a divide-and-conquer strategy. Given an input SeqBDD database P, and an

Algorithm 5.2 `buildSeqDB`(D)

```
Input: $D$ : a sequence data set.
Output: $initDB$ : the Weighted SeqBDD containing sequences in the input data set $D$.
Procedure:
1: $initDB = 0$ /* Initialise $initDB$ to be a sink-0 node */
2: for each sequence $s$ in $D$ do
3:     $P_s$ = a Weighted SeqBDD containing sequence $s$ /* Build a SeqBDD which contains a
       sequence $s$ */
4:     $initDB = initDB \bigcup P_s$ /* Add $P_S$ to $initDB$ */
5: end for
Note: $A \bigcup B$ denotes the addSeqBDD($A, B$) operation where $A$ and $B$ are SeqBDDs.
```

item x. Let var be the label of P. The frequency of x is found by recursively adding its frequency in the two database partitions: P_1 and P_0. The first partition, P_1, contains sequences which begin with var, the second partition contains the remaining sequences. If $var = x$, $weight(P_1)$ gives the frequency of x in the first database partition. On the other hand, if $var \neq x$, the frequency of x in the first database partition is computed recursively in P_1. The recursion in P_1 terminates at the first occurrence of x in each branch. The similar recursion procedure is performed in P_0. We define the operation `frequency`(P, x) for finding the frequency of item x in a SeqBDD P as the following. If P is a sink node, `frequency`$(P, x) = 0$, otherwise,

$$\texttt{frequency}(P,x) = \begin{cases} \texttt{frequency}(P_1,x) + \texttt{frequency}(P_0,x) & \text{if } P.var \neq x \\ weight(P_1) + \texttt{frequency}(P_0,x) & \text{if } P.var = x \end{cases}$$

For an item x, finding the x-suffixes (and the conditional databases), which is a major component in our mining algorithm, we define a SeqBDD operation `suffixTree` using a similar recursive strategy as the above discussed `frequency` operation. Given a SeqBDD P, and an item x, `suffixTree`(P, x) is the set of x-suffixes which are contained in P, excluding the head elements, which we call as the *x-suffix tree*. If P is a sink node, `suffixTree`$(P, x) = \emptyset$, otherwise,

$$\texttt{suffixTree}(P,x) = \begin{cases} \texttt{suffixTree}(P_1,x) \bigcup \texttt{suffixTree}(P_0,x) & \text{if } P.var \neq x \\ P_1 \bigcup \texttt{suffixTree}(P_0,x) & \text{if } P.var = x \end{cases}$$

Optimisations: The first optimisation is called *infrequent database pruning*, which is based on the monotonic property of the weights in a SeqBDD. If the weight of the top node is less than the minimum support α, then the conditional databases can be safely pruned. Additionally, `SeqBDDMiner` has a number of optimisations which rely on the use of the SeqBDD's caching library:

- **Caching of intermediate results.** The frequent subsequences from each conditional database are stored in a cache table called the *patternCache*, to be re-used if the same conditional database is projected by some other prefixes. We denote the cached patterns for a conditional database P by *patternCache*$[P]$.
- **Caching of pruned conditional databases**. For a given item x, and an input database, the (pruned) *conditionalDB* is stored in a cache table, using a similar mechanism as the *patternCache*. The procedure for obtaining the *conditionalDB* comprises of 3 operations: 1) find the x-suffix tree, 2) find its f-*list*, 3) prune infrequent items from the x-suffix tree. Each operation is associated with its own cache, so that various databases, which may share common sub-structures, may share the results of their intermediate computations.

Details of the mining algorithm: The `SeqBDDMiner` algorithm is shown in Algorithm 5.3, which we will explain line-by-line. The procedure begins with the input SeqBDD containing the initial data set with the infrequent items being removed, the f-*list* contains the frequent items which are ordered in reverse to the SeqBDD's variable ordering. Firstly, the infrequent database pruning is applied if the total frequency of the sequences is less than the minimum support (line 1). The function terminates if the database is empty (line 4), or if the output is found in the *patternCache*. In the latter case, a pointer to the cached output is returned (line 8). For each item x in the f-*list*, it projects the x-*suffix tree* (line 12). Prior to its processing, the x-conditional database is pruned in 2 steps (line 13-14): i) Find the frequent items in x-*suffix tree* using the pre-defined SeqBDD's `frequency`() operation; ii) Remove the infrequent items from x-*suffix tree*. Then, the function is called recursively on the x-conditionalDB (line 15). When the locally frequent patterns are returned from the x-conditionalDB, x is appended to the head of every pattern. The output node is built incrementally by adding the intermediate outputs for all such x (line 16).

Example 22. *Reconsider the set of sequences D in Example 20. Let* min_support *be* 3*. Construction of the initial SeqBDD database is shown in Figure 5.6, which incrementally adds the individual SeqBDD of each sequence. Let S_i be the SeqBDD which contains sequence p_i, where $i = \{1, 2, 3, 4\}$. The conditional databases for item c and b are shown in Figure 5.7. The first item in f*-list *is c, hence, c*-suffix tree *is built by finding suffixes $\{\}$ from S_1, a from S_3, and $\{\}$ from S_4, and then adding them together.*

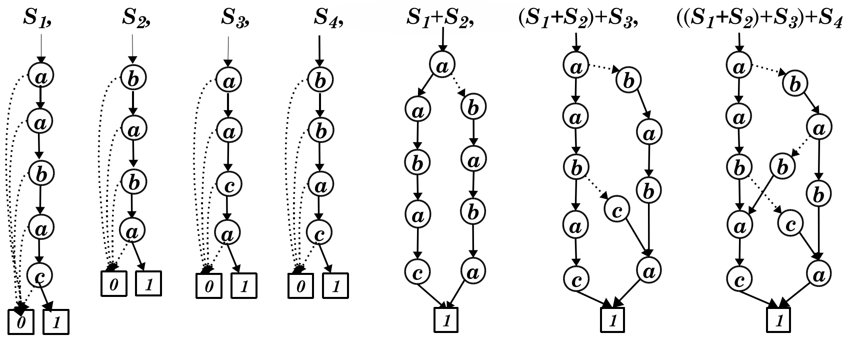


Figure 5.6: SeqBDDs representing $S_1 = \{aabac\}$, $S_2 = \{baba\}$, $S_3 = \{aaca\}$, $S_4 = \{bbac\}$, $S_1 + S_2$, $(S_1 + S_2) + S_3$, and $((S_1 + S_2) + S_3) + S_4$, using a lexicographic variable ordering

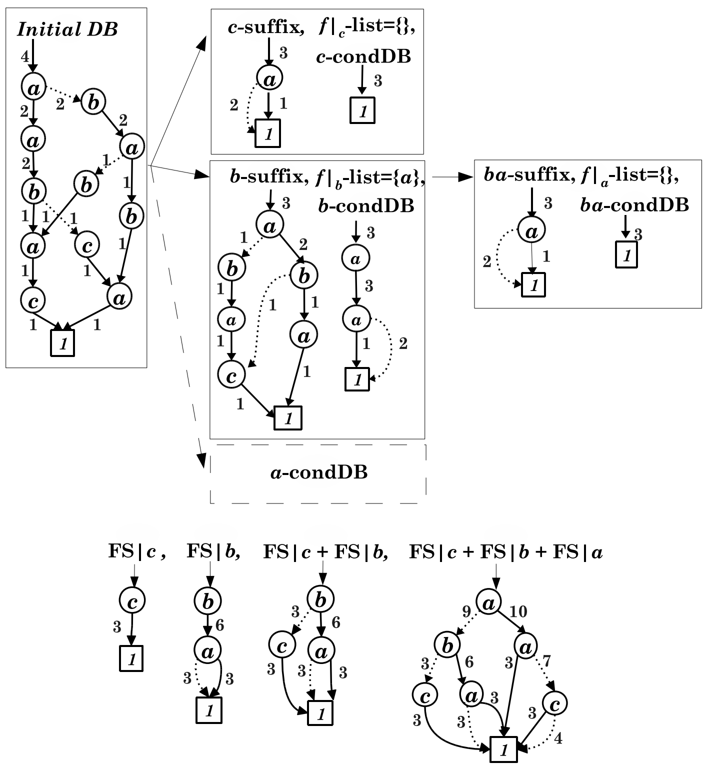


Figure 5.7: The initial database, and conditional databases when growing prefix c and prefix b, and the corresponding locally frequent subsequences, labeled as $FS|_c$ and $FS|_b$, respectively. $FS|_c + FS|_b$ shows the combined frequent subsequences. $FS|_c + FS|_b + FS|_a$ shows the globally frequent subsequences. ($minsup = 3$)

Algorithm 5.3 `SeqBDDMiner`(*inputDB*, α, *f-list*)

```
Input: inputDB : a SeqBDD containing the input database
    $\alpha$ : the minimum support threshold
    f-list : a list of items which occur frequently in inputDB
Output: allFS : a SeqBDD containing the frequent subsequences
 1: if (weight(inputDB) $< \alpha$) then
 2:     return 0 /* Infrequent database pruning: return sink-0 node */
 3: end if
 4: if (inputDB is a sink node) then
 5:     return inputDB /* Terminal case */
 6: end if
 7: if (patternCache[inputDB] is not empty) then
 8:     return patternCache[inputDB] /* Do cache lookup: return the cached entry if exists */
 9: end if
10: allFS = 1 /* Initialise the output node as a 1-sink node */
11: for each item $x$ in f-list do
12:     $x$-suffix tree = suffixTree(inputDB, $x$) /* Find the $x$-suffix tree */
13:     $f|_x$-list = the list of frequent items in $x$-suffix tree /* Find the frequent items in $x$-suffix
        tree */
14:     $x$-condDB = remove items which do not appear in $f|_x$-list from $x$-suffix tree /* Prune
        infrequent items from $x$-suffix tree */
15:     $x$-FS = $x \times$SeqBDDMiner($x$-condDB, $\alpha$, $f|_x$-list) /* Find frequent subsequences with prefix
        $x$ from the conditional DB */
16:     allFS = addSeqBDD($x$-FS, allFS) /* Incrementally build the output node */
17: end for
18: patternCache[inputDB] = allFS /* Cache the output patterns */
19: return allFS
```

Note : $x \times P$ appends item x to the head of every sequence in P, by creating a *node*$(x, P, 0)$, where P is a SeqBDD.

The f-list *in* c-suffix tree *is empty. Thus, the resulting* c-conditionalDB *contains an empty set* $\{\}$ *with a support of* 3*, no more patterns can be grown. The output node for prefix c, containing* $\{c : 3\}$ *is shown under label* $FS|_c$*.*

The next item in f-list *is item b.* b-suffix tree *contains suffixes* ac, aba, bac*, whose* f*-list (labeled as* $f|_b$-list*) contains only item a. Then,* ***`SeqBDDMiner()`*** *is called on* b-conditionalDB*. Consequently, prefix a is being grown from* b-conditionalDB*, resulting frequent subsequences* $b : 3$ *and* $ba : 3$*, labeled as* $FS|_b$*. The output node,* allFS*, which previously contains* $FS|_c$*, is now combined with* $FS|_b$*. The new output node is labeled* $FS|_c + FS|_b$*. Since for every prefix item x, the highest node of* allFS *always has a lower index than x (due to the reversed item-ordering in* f-list*),* $FS|_c + FS|_b$ *can be obtained by simply appending* allFS *to the* 0*-child of* $FS|_b$*. The same procedure is performed for the last item, a, obtaining patterns* $\{a : 4, ac : 3, aa : 3\}$*.*

Soundness and completeness of the algorithm: We will show that the algorithm is sound and complete, based on properties of the Weighted SeqBDD. As earlier discussed, the output patterns are grown recursively based on the frequent items from each conditional database. Thus, we need to show that the frequent items are correctly found from the given database. We show this by proving the correctness of the *frequency* SeqBDD operation.

Theorem 11. *Given a SeqBDD P and an item x. The function* `frequency`(P, x) *correctly calculates the frequency of item x in the given database P.*

Proof. The total frequency of an item x, can be derived from paths which contain x, since each path maps to a unique sequence. Firstly, if the database contains only a single path, which may contain multiple occurrences of x, the weight of the highest x-node gives the frequency of x in the database, since any lower x-node belongs to the same sequence. Secondly, if the database contains multiple paths and there are two x-nodes A and B such that each one is the highest x-node in their corresponding paths, A and B represent two separate partitions of the database and thus, the total weights of A and B gives the frequency of x in the overall database. □

Theorem 12. *The* SeqMiner *algorithm is sound and complete.*

Proof. We will prove this theorem in two parts: (a) the algorithm is sound, and (b) the algorithm is complete.

Soundness: We will show that growing prefix x from an item in the x-conditional database generates a frequent subsequence pattern. Given a prefix $x = x_1, x_2, \ldots x_n$, and its conditional database P, by definition, P contains items which are frequent in the longest x-suffixes from the original database. The frequency of each item in the original database is no smaller than its frequency in P. More specifically, the frequency of each item i in P, is the number of sequences in the original database which contain subsequence $x_1, x_2, \ldots x_n, i$. Thus, if i occurs in P, growing x with item i generates subsequence $x_1, x_2, \ldots x_n, i$ which is frequent in the original database.

Completeness: We will show by contradiction that there exists no frequent subsequence patterns which contain an item which does not occur in the conditional database of any of its prefixes.

Suppose there exists a frequent subsequence pattern $q = q_1, q_2, \ldots q_{n-1}, q_n$ which is not found by the algorithm, because item q_n does not occur in the conditional database of one of the prefixes of q. Let x and y be two prefixes of q, such that $x = q_1, q_2, \dots q_m$, where $m < (n-1)$, and $y = q_1, q_2, \dots q_m, \dots q_{n-1}$. Let P be the x-conditional database. Suppose item q_n does not occur in P because $z = q_1, q_2, \dots q_m, q_n$ is infrequent. The subsequence z is also a subsequence of the frequent pattern q, which contradicts the anti-monotonic property which says that all subsequences of a frequent subsequence pattern are also frequent. Therefore, such a subsequence pattern q does not exist, and the algorithm is complete. □

Complexity analysis: To analyse the complexity of our mining algorithm, we consider the space complexity for constructing the initial database, the final output, and the conditional databases, in terms of the number of nodes. The time complexity can be derived from the space complexity by Theorem 8, which is $O(N)$ where N is the number of nodes.

Given a SeqBDD P which represents a sequence database, and a SeqBDD S which represents the frequent subsequences in P. Let L be the maximum length of the sequences in P, k be the number of sequences in P.

Theorem 13. *The total number of nodes for constructing the initial database is $O(k^2.L)$.*

Proof. The initial database is constructed by incrementally adding the SeqBDD representation of each sequence to the database. Let A and B be two SeqBDDs, where A contains the SeqBDD for the first $n-1$ sequences, B contains the SeqBDD for the n-th sequence, where $n = \{1, 2, \dots k\}$,. Since $|A|$ is bounded by $(n-1).L$, and $|B|$ is bounded by L, then the output of `addSeqBDD`(A, B) contains $O(|A| + |B|)$ nodes, which is equal to $O(n.L)$. So, the overall time complexity to build the complete database is $\sum_{n=1}^{n=k} O(n.L)$, which is $O(k^2.L)$. □

Theorem 14. *The total number of nodes for constructing the conditional databases is $O(k^2.L^2.2^L)$.*

Proof. The number of frequent subsequences is $O(k.2^L)$, and each subsequence contains $O(L)$ elements. Therefore, the total number of nodes in the output SeqBDD is $O(k.L.2^L)$, which corresponds to the number of conditional databases. Since the size of an x-conditional database is bounded by the x-suffix tree, let us consider the following cases when processing `suffixTree`(P,x). Let $S =$ `suffixTree`(P, x). It is straightforward to show that $|S| = O(|P|)$ if P is a sink node or P contains only one sequence. If P is not a sink node, the output is obtained by adding `suffixTree`(P_1, x) (or P_1 if the label of P is x) with `suffixTree`(P_0, x), each of which contains $O(|P|)$ nodes. Hence, the overall size of the output is $O(|P|)$, which is $O(k.L)$ by Theorem 9, and the total number of nodes used by the conditional databases is $O(k^2.L^2.2^L)$. □

Note: In practice, many nodes are shared across multiple conditional databases, and many conditional databases are pruned by the infrequent database pruning. Hence, the total number of nodes is much less than $k^2.L^2.2^L$. Moreover, SeqBDD's caching principle avoids redundant node constructions by allowing any of the computation results (the suffix trees and conditional database) for each subtree to be cached and re-used when needed.

5.6 Performance Study

In this section we present experimental results to compare the performance of our **SeqBDDMiner** algorithm, which is based on our proposed Weighted Sequence Binary Decision Diagrams, with the state-of-the-art prefix-growth algorithms such as PLWAP [48], and PrefixSpan which has been shown superior in [129]. We implement the weighted SeqBDDs and our `SeqBDDMiner` algorithm using the core library functions from an existing BDD package, JINC[2]. All implementations were coded in C++. Similarly, PLWAP and PrefixSpan were coded in C++, which we obtained from their respective authors. All tests were performed on a 4.0 GHz CPU with 32 GB RAM, running Redhat Linux 5, with a CPU time-out limit of 100,000 seconds per mining task. We use the default table parameters as provided by the author of the BDD package shown below.

Maximum size of the unique table (bytes):	M	$=$	Unlimited
Maximum size of each cache table (bytes):	K	$=$	131072

Hash functions for the cache table:
for binary operations: $h(A, B) \;=\; ((B + (A * p_2)) * p_1) \bmod K$
for unary operations: $h(A) \;=\; ((A * p_1) \bmod K,$
where p_1 and p_2 are large prime numbers, e.g. $p_1 = 4256249$, $p_2 = 12582917$

The cached binary operations include `addSeqBDD`(P, Q), and `suffixTree`(P, x).

[2]JINC was developed by the author of [123] for studying a different type of weighted BDDs

Table 5.4: Data set characteristics and a proposed categorisation based on the average sequence length

Data set name	$\|D\|$	N	L	C	Category
yeast.L200	25	4	129	35	Short
DENV1	9	4	50	50	Long
snake	174	20	25	25	Short
PSORTb-ccm	15	20	50	50	Long
gazelle	29369	1451	652	3	V.short
davinci	10016	1108	416	2	V.short
C2.5.S5.N50.D40K	17808	50	42	3	V.short

$|D|$ = Number of sequences in the data set
N = Number of items in the domain
L = Maximum sequence length
C = Average sequence length
V.short = Very short

The unary operations include `SeqBDDMiner`(P) (finding the frequent subsequences from a given database P), and removing the infrequent items from a given database.

Our experiments aim to analyze the following factors: 1) **SeqBDD's compactness** : the amount of data compression which can be achieved by the (Weighted) SeqBDD due to its fan-out and fan-in; 2) **Runtime performance**: the runtime performance of our SeqBDD-based algorithm in comparison to the other encodings discussed in Section 5.3, and in comparison to the existing prefix growth algorithms. We will also analyze the effects of increasing similarity of the sequences, which would increase the length (and volume) of the patterns. 3) **Effectiveness of pattern caching**: how much database projection is avoided due to pattern caching. We analyse three types of real data sets: i) DNA sequence data sets, ii) protein data sets, and iii) weblog data sets. Their characteristics are shown in Table 5.4. Detailed descriptions of each type of data set are given shortly.

DNA sequence data sets typically contain long sequences which are defined over 4 letters, i.e. A, C, G, T. Due to the small alphabet size, the sequences may be highly similar and a large number of long frequent subsequences exist. We choose 2 data sets from the NCBI's website [102]: *yeast.L200*[3], which contains the first 25 sequences, with a maximum length of 200 elements, and DENV1[4], which contains genes from

[3]*yeast.L200* is obtained from NCBI's website using query: *yeast [organism] AND 1:200 [sequence length].*

[4]*DENV1* is obtained from NCBI's website using query: *dengue virus type 1 AND 1:100 [sequence length]*;

dengue virus. We remove any sequence duplicates for our experiments. Due to the large number of patterns, we only use the first 50 elements from each sequence, to allow mining to complete within a reasonable time with low support thresholds.

Protein sequence data sets are defined over 20 letters which are also relatively dense. The two data sets are *snake* [156], and *PSORTb-ccm* [143] which is smaller. Due to the length of the input sequences, we only use the first 25 elements from each sequence in the *snake* dataset, and 50 elements from each sequence in the *PSORTb-ccm* data set.

Weblog data sets: Compared to the biological data sets, weblog data sets have a larger domain and the sequences are relatively shorter. In particular, mining frequent subsequences in the weblog data sets is challenging when the minimum support is low due to the large number of sequences. We choose two weblog data sets : i) *gazelle* [156], ii) *davinci* [47].

The synthetic data sets were generated using the sequential synthetic data generator in [75] The first data set, $C2.5.S5.N50.D40K$ consists of 40,000 sequences, defined over 50 items, with average sequence length of 2.5, and the average length of maximal potentially frequent sequence is 5. This data set contains shorter sequences than the weblog data sets, although it has a smaller domain. Secondly, we use the synthetic data generator to generate data sets with a varied value of N (i.e. number of items in the domain) to analyse the effects of the alphabet size to the algorithm's performance. Moreover, to analyse the effects of the similarity of the sequences, we choose the protein *PSORTb-ccm* and append an increasing length of synthetically-generated common prefixes and common suffixes to each sequence.

5.6.1 Compactness of SeqBDDs Due to Fan-out and Fan-in

In this subsection, we examine the compactness of SeqBDDs for compressing a sequence database, due to their node fan-out and node fan-in. To calculate the fan-out compression factor, we implement a Sequence Binary Decision Tree (SeqBDTree), which is a relaxed type of SeqBDD with no node fan-in. We then calculate the compression being achieved due to node fan-out, i.e. *fanOut* and *fanIn*. [5], *fanOut* is the number of nodes in the SeqBDTree, counted as a proportion of the the total number of elements in the data set. *fanIn* is the node-count difference between SeqBDD and SeqBDTree,

[5]$fanOut = 1 - \frac{|SeqBDTree|}{total\_number\_of\_elements}$, $fanin = \frac{|SeqBDTree| - |SeqBDD|}{|SeqBDTree|}$, $|SeqBDTree|$ = the number of nodes in the *SeqBDTree*

Table 5.5: Fan-out and Fan-in compression factors of SeqBDD and a proposed data set categorisation

Dataset name	$fanOut$	$fanIn$	Category
yeast.L200	0.22	0.12	Similar
DENV1	0.53	0.07	Similar
snake	0.52	0.10	Highly similar
PSORTb-ccm	0.024	0.004	Dissimilar
gazelle	0.42	0.16	Similar
davinci	0.59	0.03	Similar
C2.5.S5.N50.D40K	0.52	0.18	Highly similar

as a proportion of the total number of nodes in the SeqBDTree, which represents the compression being achieved by the SeqBDD over SeqBDTree (or prefix tree). Table 5.5 shows for each data set, the *fanOut* and *fanIn* compression factors. The mining times using either data structures for a few representative data sets will also be shown shortly.

The *fanOut* compression factors represent the amount of common prefixes shared among the input sequences in each data set. *DENV1*, *snake*, *davinci*, and *C2.5.S5.N50.D40K* are those data sets which have more than 50% *fanOut* factors, i.e. on average, more than 50% of the prefixes are shared across sequences. Such a factor indicates a high prefix-similarity between the input sequences. The *fanIn* compression factors show that low compression factors can be achieved by the SeqBDD over the prefix tree representation. Only up to 18% is achieved, for *C2.5.S5.N50.D40K* data set, which indicates that only 18% of the suffixes are shared across sequences, given thousands of sequences that exist. The *PSORTb-ccm* data set, which contains a small number of short sequences, has the lowest *fanIn* compression of 0.4% over the prefix tree. We find that the compression factors are related to the similarity and the length of the sequences. Intuitively, for data sets which contain long sequences, the SeqBDDs are likely to have more node-sharing. But, it is true only if there eixst many common prefixes or suffixes, or a few but long common prefixes or suffixes.

The first two DNA sequence data sets are considered similar, due to their small alphabet and long sequences. Moreover, the protein data sets, which contain shorter sequences over a larger alphabet are relatively sparser, thus, have lower similarity, than the DNA sequences. The *snake* data set, however, is shown to contain much larger prefix sharing and suffix sharing compard to the other protein data set, which indicates the highly similar sequences it may contain. The two weblog data sets, both of which contain a large number of very short sequences, are considered sparse data sets. More specifically, the *gazelle* data set has lower *fanOut* but higher *fanIn* compression factor compared to the *davinci* data set. Such a high probability of sharing across either prefixes or suffixes of the sequences occurs due to the large number of sequences that exist in these data sets. It indicates that there is a relatively high similarity between sequences in these data sets. Lastly, the synthetic data set, which also contains a large number of very short sequences, is sparser than either weblog data set, but it has the highest *fanIn* compression factor due to the suffix sharing, and also allows a high *fanOut* compression factor due to the prefix sharing. These factors indicate the high similarity of sequences in this sparse synthetic data set.

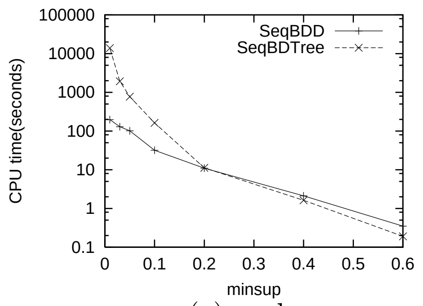
Based on the observations made above on the compression factors, we can roughly classify the data sets into the following categories:

- Highly similar (high prefix-similarity and suffix-similarity): *fanOut* ≥ 0.5 and *fanIn* ≥ 0.1.
- Similar (either one of high prefix-similarity or suffix-similarity): *fanOut* ≥ 0.5, or *fanIn* ≥ 0.1.
- Dissimilar (low prefix-similarity and low suffix-similarity): *fanOut* < 0.5 and *fanIn* < 0.1.

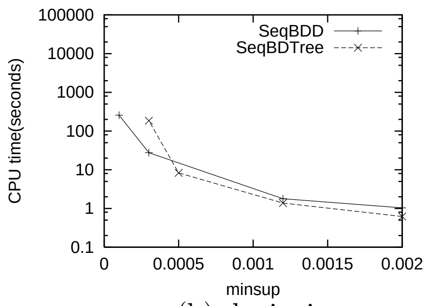
Figure 5.8 shows the respective mining times using either SeqBDD or SeqBDTree data structure for mining frequent subsequences in two data sets : *snake* and *davinci*. Both data sets have similar *fanOut*, but *snake* has a larger *fanIn*, i.e. contains more similar suffixes than *davinci*. It shows that both SeqBDTree and SeqBDD have similar runtimes when the support threshold is larger than a certain limit (20% for *snake*, and 0.04% for *davinci*). For lower support threshold values, SeqBDD-based miner achieves an exponential improvement over SeqBDTree-based miner. These trends show that the node fan-in in SeqBDDs is increasingly more effective to the mining time efficiency, as the support threshold decreases.

5.6.2 Runtime Performance of the Mining Algorithm

In Table 5.3 (Section 5.4.2), we showed some statistics in terms of the number of nodes in (Weighted) SeqBDD, and in ZBDD (with two alternative itemset encoding scheme) for representing the frequent subsequence patterns. We now show the mining time comparison between either type of data representations when they are used for

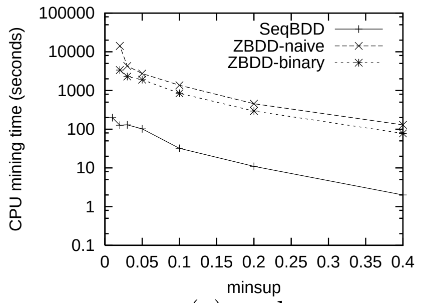


(a) snake

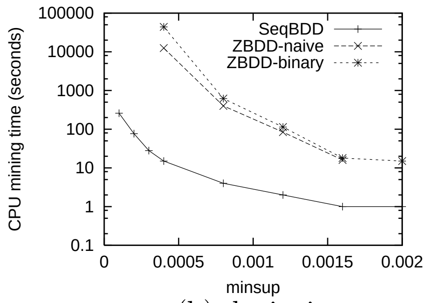


(b) davinci

Figure 5.8: Mining time comparison between SeqBDD and SeqBDTree



(a) snake



(b) davinci

Figure 5.9: Mining time comparison between SeqBDD, ZBDD$_{naive}$, and ZBDD$_{binary}$

mining frequent subsequences in *snake* and *davinci* data sets. Figure 5.9 shows that SeqBDDMiner is uniformly superior in both data sets, being at least 10 times faster than either ZBDD representation. Interestingly, ZBDD$_{naive}$ is faster than ZBDD$_{binary}$ in the *davinci* data set, but slower in the *snake* data set. The statistics in Table 5.3 show that ZBDD$_{binary}$ is faster when its size is not much larger than ZBDD$_{naive}$.

DNA sequence data sets: The runtime performance comparisons are shown in Figure 5.10. In the *yeast.L200* data set, SeqBDDMiner is 10 times slower than PLWAP and 100 times slower than PrefixSpan when the support threshold is 70% or larger, but its running time grows exponentially slower as the threshold decreases. SeqBDDMiner has, moreover, higher scalability since it can finish mining given a support threshold as low as 5% in under 1000 seconds, whereas both PLWAP and PrefixSpan could not finish within the CPU time limit given a support threshold.

In the *DENV1* data set, which contains similar sequences, SeqBDDMiner is substantially the most efficient, whereas PLWAP and PrefixSpan are exponentially slower with respect to a decreasing minimum support. For support threshold values less than

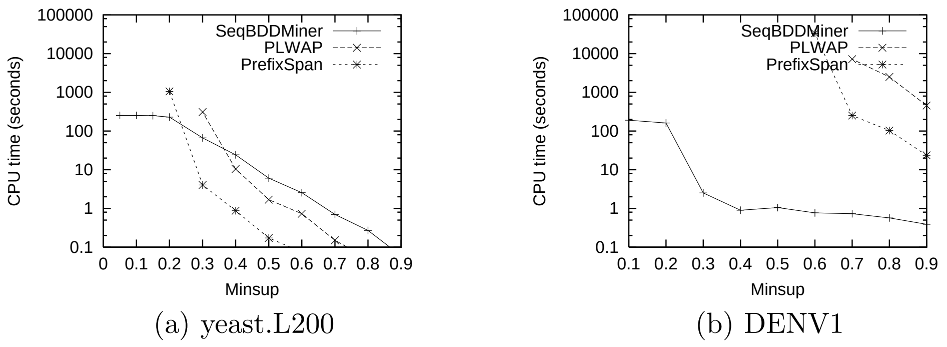


Figure 5.10: Mining times in DNA data sets

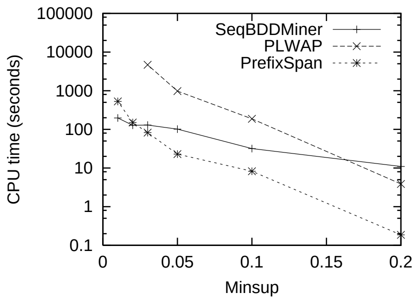
60%, PrefixSpan and PLWAP could not complete within the CPU time limit. This shows the benefits of using SeqBDDs for mining highly similar sequences, for which both PLWAP and PrefixSpan have more limited scalability.

Protein sequence data sets: The runtime performance comparisons are shown in Figure 5.11. For both datasets, SeqBDDMiner is more scalable than the other algorithms when the support threshold value is low. More specifically in the *snake* data set, given a support threshold value as low as 2%, SeqBDDMiner completes mining within 500 seconds which is 4 times faster than PrefixSpan, and PLWAP could not complete mining within the CPU time limit. In general, the runtimes of both PLWAP and PrefixSpan grow exponentially slower than SeqBDDMiner as the support threshold decreases. Similar trends are found in the small and dense data set, *PSORTb-ccm*.

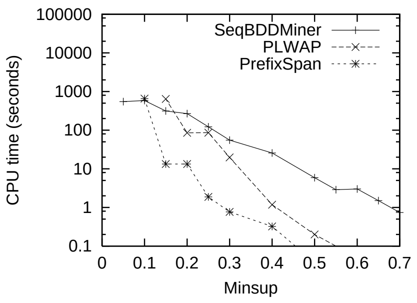
Weblog data sets: The runtime performance comparisons are shown in Figure 5.12. Overall, the runtime of SeqBDDMiner is slower than both PLWAP and PrefixSpan for high support threshold values, but SeqBDDMiner grows exponentially slower than the other algorithms with respect to a decreasing support threshold value.

In the *gazelle* data set, for support threshold larger than 0.05%, SeqBDDMiner is up to 10,000 times slower than the both PLWAP and PrefixSpan. But for a support threshold value as low as 0.02% (lower for PrefixSpan), SeqBDDMiner spends about 50,000 seconds, whilst PLWAP could not complete mining within the CPU time limit. In the *davinci* data set, which has a larger *fanOut* factor than *gazelle*, SeqBDDMiner is up to 100 times faster than PLWAP and PrefixSpan for support threshold 0.03% or lower.

Synthetic data sets: The mining time for mining frequent subsequences in $C2.5.S5.N50.D40K$ data set is shown in Figure 5.13(a). PrefixSpan has the best run-

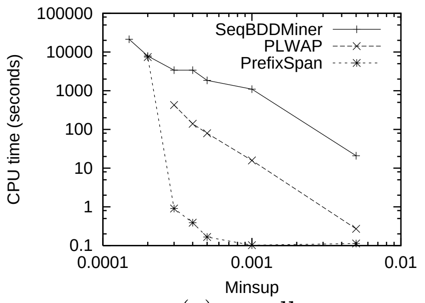


(a) snake

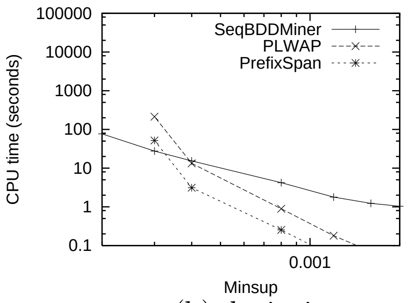


(b) PSORTb-ccm

Figure 5.11: Mining times in protein data set



(a) gazelle



(b) davinci

Figure 5.12: Mining times in weblog data sets

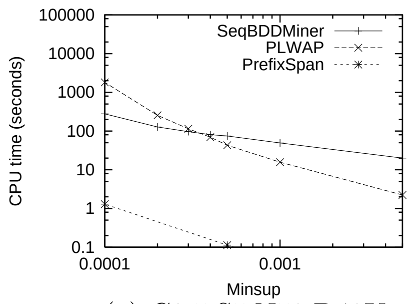
time performance in either data set, being 250 times faster than SeqBDDMiner. When compared against PLWAP, the runtime of SeqBDDMiner grows slower than PLWAP as the support threshold decreases, and more specifically, SeqBDDMiner is up to four times faster than PLWAP when the support threshold is lower than 0.04%.

Figure 5.13(b) shows the effects of increasing the domain size on various synthetic data sets, each of which is generated using a fixed $S10.D10K$ parameter and the domain size is varied between $4, 6, 8, 10, 20, 30, 40,$ and 50, with a minimum support threshold being 25%. Having fewer items in the domain consequently generates more similar sequences, and longer frequent subsequences. It shows that PLWAP and PrefixSpan have similar relative runtime performance, but SeqBDDMiner becomes more competitive as the number of domain items decreases.

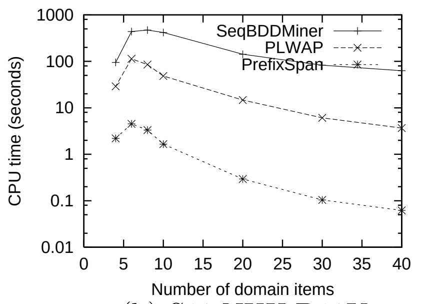
Lastly, we analyse the effect of similarity of the input sequences to the algorithms' performance by appending an increasing length of synthetic common prefix, or common suffix, to the *PSORTb-ccm* data set. Figure 5.13(c) and (d) show the trends of running time for each scenario given a support threshold of 80%. It shows that SeqBDDMiner is superior, having an almost constant (i.e. very small increase) in its mining time as the length of common prefix increases up to 20 items, whereas PLWAP and PrefixSpan increase exponentially. As the length of common suffix increases, SeqBDDMiner benefits from sharing of common sub-trees with a linear growth of running time whilst the other algorithms have an exponentially increasing running time.

5.6.3 Effectiveness of SeqBDDMiner Due to Pattern Caching

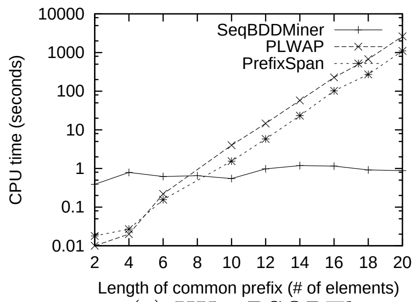
In order to analyse the effectiveness of BDD's caching ability, we compare the number of database projections performed by SeqBDDMiner against PrefixSpan. For the case of SeqBDDMiner, if a conditional database exists in the *patternCache* cache, then no further projections are performed for that database. We also count the number of conditional databases should the caching mechanism is not used by the SeqBDDs, which we refer to as SeqBDTreeMiner. As representative data sets, we show the comparison for *DENV1*, *snake*, *PSORTb-ccm*, and *gazelle* data sets in Figure 5.14. It shows that SeqBDDMiner always projects the smallest number of conditional databases. Moreover, in cases where there is a huge reduction in terms of the database projections, such as in *DENV1* data set, and in other data sets with low support threshold, SeqBDDMiner projects significantly fewer databases than PrefixSpan. When compared to SeqBDTreeMiner, it shows that the caching mechanism in SeqBDDMiner reduces the



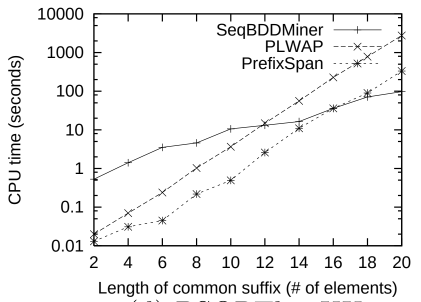
(a) C2.5.S5.N50.D40K



(b) S10.NXX.D10K



(c) XX—PSORTb



(d) PSORTb—XX

Figure 5.13: Mining times in synthetic data sets

number of database projections by up to 12 times for all data sets, being more effective when the minimum support threshold is low.

We now examine the hit rate of the cached patterns from each conditional database, by counting the number of conditional databases which are skipped because they exist in the cache table. Figure 5.15(a) shows that at a support threshold as high as 90% in the highly similar *DENV1* data set, SeqBDDMiner is able to achieve a high hit rate of 53%.

In the *snake* data set which contains short sequences, and *PSORTb-ccm* data set which contains long but dissimilar sequences, SeqBDDMiner achieves a low hit rate except when the support threshold is relatively low (Figure 5.15(b) and (c)). This shows that a large amount of node-sharing among the conditional databases at a low support threshold value can still be achieved, even though the sequences are short or dissimilar.

In the *gazelle* data set which is highly similar but contains short sequences, the hit rate does not even reach 30% for a support threshold as low as 0.03% (Figure 5.15(d)), which corresponds to the poorer time performance of SeqBDDMiner, except when the support threshold value is low.

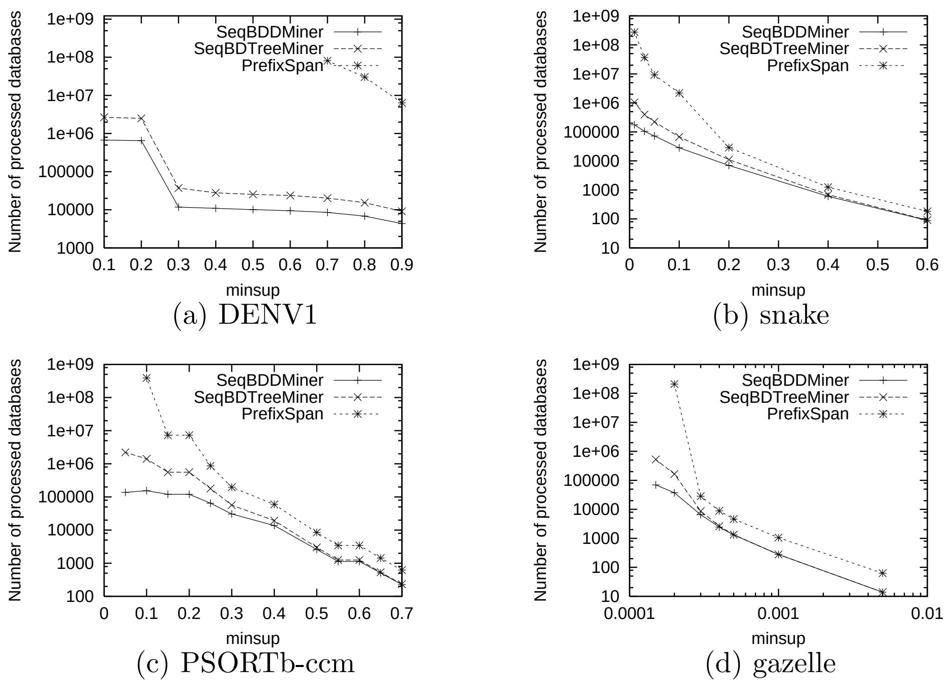


Figure 5.14: Number of projected conditional databases

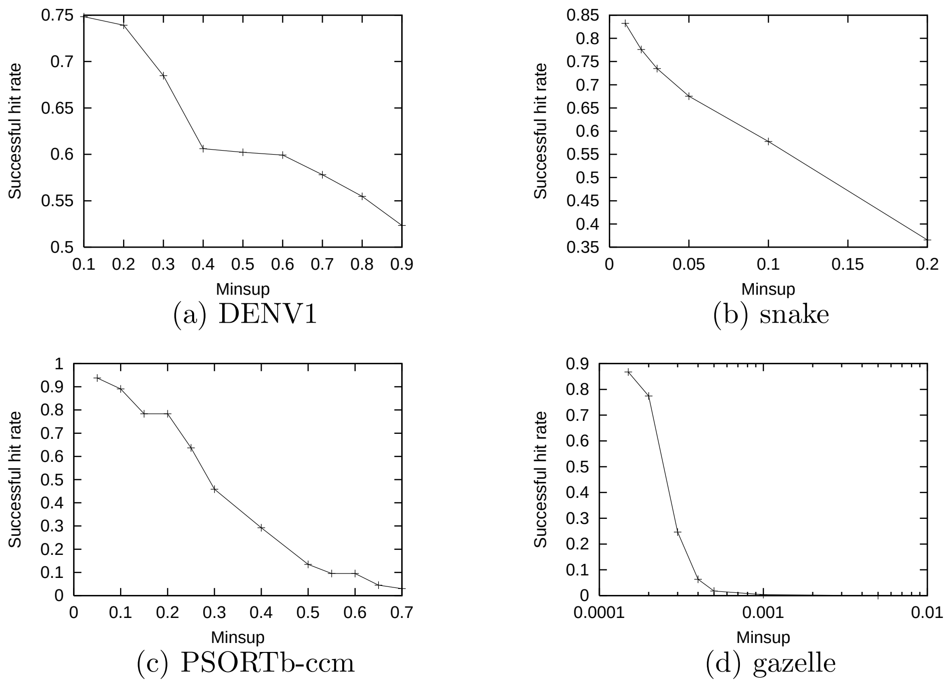


Figure 5.15: Hit rate of cached patterns in SeqBDDMiner

5.7 Weighted SeqBDD vs Prefix Tree for Mining Frequent Subsequences

In this section, we provide a detailed discussion of the performance of our SeqBDD-based algorithm in terms of the effectiveness of its caching utility which is affected by the amount of node sharing between the databases, which leads us to identify two interesting circumstances according to similarity characteristics of the input data set.

(Highly) similar sequences: In a data set which contains (highly) similar sequences, its input SeqBDD has a large amount of node fan-out or node fan-in and the conditional databases are also more likely to share many nodes, especially when the sequences are long.

Our experimental results show that SeqBDDMiner achieves a high hit rate of the pattern cache and the best runtime performance when mining the highly similar DNA or protein sequence data sets, especially at a low support threshold. Consider the following situation. Suppose p and q ($p \neq q$) are two frequent subsequences. If every sequence which contains p also contains q, then both conditional databases are identical and the pre-computed patterns can be re-used. Otherwise, the two conditional databases may still share common sub-trees given the input sequences are highly similar. When performing database projections, the databases share a lot of similar computations due to their large degree of node-sharing.

However, if the sequences are very short, the patterns are more likely to be dissimilar and the amount of node-sharing among the conditional databases may not be significant. This is proven by the poor hit rate of the cached patterns when mining the weblog *gazelle* data set. In this circumstance, the construction of the conditional databases is costly, whereas PLWAP or PrefixSpan can have a better performance since they do not physically build the conditional databases.

Dissimilar sequences: In a data set which contains dissimilar sequences, its input SeqBDD has little node fan-out and node fan-in. In general, being dissimilar, the conditional databases are also dissimilar and caching effectiveness decreases, since not many node re-use is allowed. If the support threshold is low, however, similarity of the frequent subsequences increases and the node-sharing among the conditional databases also increases, as shown by the increased hit rate of the pattern cache in our experiments with the *DNA.Homologene554*, and *DENV2* at a very low support threshold value.

In the beginning of this chapter we posed three questions which we aim to answer:

Can a BDD be used for compactly representing sequences? We showed that ZBDDs have a limited data compression ability for representing sequences. In this paper, we have proposed a more suitable type of BDD, namely Sequence BDDs, which allow sequences of various lengths to share nodes representing their common prefixes as well as suffixes, through the sharing of common sub-trees. In our experiments, we showed that a SeqBDD is able to provide at least 10% data compression over the relevant prefix tree representation. Furthermore, we found that the total amount of node sharing across the conditional databases is proportional to the compactness of the initial SeqBDD database.

Can the use of a SeqBDD benefit frequent subsequence mining? The key features of our proposed algorithm are SeqBDD's canonical structure and its caching ability. We performed experiments for examining the effects of caching in our SeqBDDMiner, and showed that regardless of the compactness of the initial SeqBDD database, maintaining the canonicity across multiple SeqBDDs is advantageous since many of the intermediate databases do share common sub-trees. Thus, redundancy can be avoided by allowing the same sub-trees to re-use their computation results.

Can our proposed SeqBDD-based miner outperform state-of-the-art pattern growth techniques in frequent subsequence mining? When the input sequences are long and similar, SeqBDDMiner outperforms the state-of-the-art pattern growth techniques such as PLWAP and PrefixSpan. When the input sequences are short, or dissimilar, SeqBDDMiner is less competitive due to the low node-sharing across the conditional databases, except when the support threshold value is low for which SeqBDDMiner has a higher scalability than the other techniques.

5.8 Related Work

Our mining technique is based on the prefix-growth framework which suits prefix-monotone constraints [131]. Such constraints include the minimum frequency (considered in this paper), minimum length, gap constraint [77], similarity constraint (measured by the longest common subsequences) [119], and many more. There also exist tough constraints [131], which are not prefix-monotone, such as sum or average constraint, and regular expressions. However, work in [131] showed that the prefix-growth framework, as well as our technique, can be extended to handle such constraints.

CloSpan [162] and BIDE [156] are extensions of the prefix-growth framework for mining closed subsequences, which are a different class of frequent subsequence patterns. Extending our algorithm for mining closed subsequences is also possible, since suffixes of a sequence which have the same frequency may share nodes in the weighted SeqBDD representation.

There appears to be little work that considers the use of BDDs for storing and manipulating sequences. Work in [80] addresses the problem of capturing/enumerating all possible n-grams (sequences without gaps), such as in a text file. Their approach is based on the use of ZBDDs and sequence-to-itemset encoding. More discussion about this approach (and some of its limitations) can be found in Section 4. Other BDD-variants exist for analysing sequential events in fault-tree analysis [144], but they consider pseudo-sequential events since any event does not occur more than once in each fault-path. There exists a type of unordered BDD, namely the Free BDDs[59], but unlike SeqBDDs, they do not allow a variable to appear multiple times in any path.

The combinatorial pattern matching community has studied the use of subsequence automata [72], which are inspired by Directed Acyclic Subsequence Graphs (DASGs) [13], for solving subsequence matching problems. Similar to SeqBDDs, identical subtrees are merged, but every node in DASGs may have m outgoing edges, where m is the size of the alphabet. Such a technique can be extended for finding frequent subsequences, but it does not have SeqBDDMiner's ability to avoid infrequent candidate generations, and to re-use intermediate computation results which we have shown to be particularly advantageous in our study.

5.9 Summary

In this chapter, we have introduced Weighted Sequence Binary Decision Diagrams (Weighted SeqBDDs) for efficient representation of sequences and shown how they may be used as the basis for mining frequent subsequences. A primary objective has been to investigate situations where the use of a Sequence BDD is superior to the prefix tree style approaches. In our experimental results, we have shown that SeqBDDs can be highly effective in improving the efficiency of frequent subsequence mining, for cases when the input sequences or intermediate computations are similar, the sequences are long, or when the mining is at low support. Based on this evidence, we believe SeqBDDs are an important and worthwhile data structure for sequence data mining.

Chapter 6

Efficient Mining of Expressive Contrast Patterns using Zero-suppressed Binary Decision Diagrams

Contrast patterns are distinguishing characteristics between classes of data, which can be expressed by patterns whose frequency is significantly different between two classes. In the previous two chapters, we have shown that Zero-suppressed Binary Decision Diagrams are useful for mining frequent patterns and frequent subsequences, especially in a high-dimensional space. In this chapter, we will introduce new definition of expressive contrast patterns, which generalises the previous definitions of simple contrast patterns. We will also propose efficient algorithms for mining contrasts, based on the use of the graph-based data structures that we have previously studied, such as Zero-suppressed Binary Decision Diagrams and their weighted variant (Chapter 4).

6.1 Introduction

In a high-dimensional space, such as in gene expression data sets [51, 38, 84], mining contrast patterns is highly challenging, due to two reasons: i) the exponential search space, ii) the multiple frequency constraints limit the search space pruning. Previous contrast mining techniques have been unable to handle more than about 60 dimensions in such circumstances. Also, existing definitions of emerging patterns [37] are limited to express only conjunction of attribute values, or data items. When disjunctions as well as conjunctions of attribute values are allowed in the more expressive contrasts, the pattern search space is considerably larger and therefore, mining them is even more challenging. A key focus of our study is the mining of contrasts for high dimensional data, especially when disjunctions are allowed.

Existing emerging pattern mining technique uses structures such as FP-trees [70] and Pattern trees [51], for storing and manipulating the input database, the conditional databases, and the output patterns. Our technique, on the other hand, uses graph-based ZBDDs, as database representations. Our technique also uses ZBDD routines which follow a divide-and-conquer strategy, which in turn allows the output ZBDD to be constructed bottom-up incrementally. We have shown that Zero-Suppressed Binary Decision Diagrams and their weighted variant (Chapter 4) can be a powerful data mining tool for mining high-dimensional frequent patterns. This chapter investigates their use for mining high-dimensional contrasts. The use of a graph-based data structure for contrast mining is a novel feature which has not been considered in existing techniques.

In this chapter, we make the following important contributions:

- We show that Zero-suppressed Binary Decision Diagrams (ZBDDs) can be employed as a tool for mining contrast patterns, which also allow the support constraints to be pushed inside the ZBDD manipulation routines. This provides an interesting alternative to popular structures such as the frequent pattern tree [70], whose variants have previously been proposed as an effective contrast mining method [14, 51].
- We present an algorithm, `EPMiner` that uses ZBDDs to mine a well-known, simple type of contrast pattern, known as the emerging pattern [37]. Experimental evaluation shows this technique achieves very large speedups over the state-of-the-art technique which is based on pattern trees [51].
- We investigate more complex contrast patterns which generalise emerging patterns, by allowing disjunction as well as conjunction. We call these patterns **disjunctive emerging patterns**. We establish the formal characteristics of such patterns, show that `EPMiner` can be adopted to this more complex scenario, and provide experimental evidence that it can be practically feasible for mining very high dimensional datasets. We are not aware of any other work which is suitable for mining this kind of contrast pattern.

6.2 Mining Emerging Patterns using ZBDDs

This section presents our algorithm for mining contrasts which is based on the use of Zero-suppressed Binary Decision Diagrams (ZBDDs) (Reviewed in Chapter 3). Recall the definition of emerging patterns in [37], which can be mined efficiently using border-representations of the input datasets.

Given a positive dataset D_p, a negative dataset D_n, and two support thresholds α and β, an emerging pattern p is an itemset which satisfies two constraints:

1. Anti-monotonic constraint: frequent in D_p, i.e. $\mathit{support}(p, D_p) \geq \alpha$
2. Monotonic constraint: infrequent in D_n, i.e. $\mathit{support}(p, D_n) < \beta$

Our ZBDD-based mining technique allows both constraints to be pushed inside the mining routines by adopting the pattern growth framework presented in Chapter 2. The bitmaps from both datasets are used as the method for computing the *support* of each pattern. For a given itemset q and a dataset D, $\mathit{bitmap}(q, D)$ denotes the bit-vector of the transactions in D which contains q.

6.2.1 Pushing the Anti-monotonic Constraint

The anti-monotonic constraint is pushed inside the mining routine by making sure that each prefix satisfies the frequency constraint in D_p. If an itemset is infrequent, so do its supersets, thus, they can be pruned. Using this principle, a candidate pattern is grown only if it satisfies the anti-monotonic constraint.

6.2.2 Pushing the Monotonic Constraint

The pattern growth framework is based on the relationships between emerging patterns and hypergraph transversals, which essentially pushes the monotonic constraint deep inside mining. In principle, our algorithm could examine a search space covering all possible itemset combinations. However, this is unnecessary and instead we traverse a search space which avoids generating candidate patterns which could never satisfy the monotonic β constraint. The search space is dictated by the contents of D_n.

For any given prefix p, we can partition D_n into the set of transactions which contain p (denoted by $D_n^{\overline{p}}$) and transactions which do contain p (denoted by D_n^{p}). If p

needs growing, then it only needs to be extended by an item which is not from at least one of the transactions in D_n^p, i.e. from the complement of one of the transactions in D_n^p, otherwise a non minimal pattern will result.

It is therefore advantageous for the input ZBDD to contain the complements of the transactions in D_n (i.e. $\overline{D_n}$). Traversing $\overline{D_n}$ and finding its minimal transversals, thus, ensure that the candidate generation space is much smaller. This pruning method is particularly effective if D_n contains few but long transactions, as is often the case for biological data.

The emerging patterns are grown in a depth-first fashion by recursively finding transversals of the hypergraph which consists of the transactions in D_n. If $\beta = 0$, this integrates the monotonic constraint inside the mining procedure, such that each minimal hypergraph transversal is a minimal itemset with zero support in D_n. If $\beta > 0$, the same principle applies on the right border (i.e. the maximal itemsets) of D_n which includes itemsets whose support in D_n is $\geq \beta$. We refer to the partially grown patterns as *prefixes*. As prefixes are grown, their support in D_p is calculated using the bitmap data representation for checking the anti-monotonic frequency constraint.

6.2.3 The Algorithm of EPMiner for Mining Minimal Emerging Patterns

Our EPMiner algorithm for finding minimal emerging patterns is shown in Algorithm 6.1, which is based on the use of ZBDD routines. Before we explain the algorithm line by line, let us firstly list the pruning strategies which are employed within the mining algorithm:

- α **constraint pruning**: This strategy is based on the anti-monotonicity of the *apriori* principle. Any itemset which does not satisfy the α constraint should have its supersets pruned. So, as a pre-processing step, any item i such that $support(\{i\}, D_p) < \alpha$ can be deleted from both D_p and D_n.
- β **constraint pruning:** This strategy is based on the monotonicity of the β constraint. If an itemset satisfies the β constraint, then it is not necessary to extend it any further, otherwise a non minimal pattern would result.
- **Non minimal pattern pruning:** Since the final output is required to only consist of minimal patterns, it is profitable to immediately prune any non-minimal patterns for each prefix which is being grown.

Algorithm 6.1 `EPMiner`(Z_D, $prefix$, D_p, D_n, α, β)

Call `EPMiner`($\overline{D_n}$, {}, D_p, D_n, α, β) to begin mining initially.

Input: A ZBDD Z_D containing a complemented projection of the negative dataset, a prefix itemset $prefix$ which projects Z_D, a bitmap D_p of the positive dataset, a bitmap D_n of the negative dataset, a min support threshold α, a max support threshold β.

Output: A ZBDD Z_{EP} containing the set of minimal emerging patterns (i.e. itemsets p s.t. $support(p, D_p) \geq \alpha$ and $support(p, D_n) < \beta$).

```
1:  if ($Z_D$ is a sink node) then
2:    if ($support(prefix, D_n) < \beta$) then
3:      return 1; /* $prefix$ satisfies the $\beta$ constraint */
4:    else
5:      return 0; /* Remove $prefix$ and its supersets from output */
6:    end if
7:  end if
8:  /* Let $Z_D = node(x, Z_{D_x}, Z_{D_{\overline{x}}})$ */
9:  $prefix_{new} = prefix \cup \{x\}$ /* Grow new prefix with the next item in the search
    space */
10: if ($support(prefix_{new}, D_p) < \alpha$) then
11:   $Z_{EP_x} = 0$ /* Prune $prefix_{new}$ by $\alpha$ constraint pruning */
12: else if ($support(prefix_{new}, D_n) \leq \beta$) then
13:   $Z_{EP_x} = 1$ /* Prune supersets of $prefix_{new}$ by $\beta$ constraint pruning */
14: else
15:   $Z_{EP_x} =$ EPMiner($Z_{D_{\overline{x}}}, prefix_{new}, D_p, D_n, \alpha, \beta$) /* Mine supersets of $prefix_{new}$
      from the projection of $Z_D$ which does not contain $x$ */
16: end if
17: $Z_{D_{reduced}} = Z_{D_{\overline{x}}} \bigcup_Z Z_{D_x}$ /* Remove $x$ from the search space by computing set-union
    between the two children of $Z_D$ */
18: $Z_{EP_{\overline{x}}} =$ EPMiner($Z_{D_{reduced}}, prefix, D_p, D_n, \alpha, \beta$) /* Explore candidates from the
    remaining search space */
19: $Z'_{EP_x} =$ NotSupSet($Z_{EP_x}, Z_{EP_{\overline{x}}}$) /* Non-minimal patterns elimination */
20: $Z_{EP} =$ getNode($x, Z'_{EP_x}, Z_{EP_{\overline{x}}}$) /* Combine the output patterns */
21: return $Z_{EP}$
```

The first parameter to the algorithm, Z_D, is a ZBDD containing the complemented negative dataset, or its projection. $prefix$ is a prefix itemset which satisfies the α constraint, but fails the β constraint. D_p and D_n correspond to the bitmaps from the respective positive and negative datasets.

The algorithm is invoked by calling `EPMiner`($\overline{D_n}$,$prefix = \{\}, D_p, D_n, \alpha, \beta$). It is called recursively on projections of $\overline{D_n}$ as the pattern candidates are explored. Lines 1-7 in the algorithm state the terminal condition of the recursion. When it reaches a sink node, 0 or 1, it has reached the end of the search space for growing the given $prefix$. If $prefix$ passes the β constraint, then accept $prefix$ as a minimal emerging pattern (line 3). Otherwise $prefix$ cannot be part of the output ZBDD, and so the the sink-0 node is returned (line 5).

Following a divide-and-conquer strategy, the algorithm comprises of two sub-procedures. The first procedure computes Z_{EP_x}, which grows $prefix$ with the next item x which is found in the currently projected database Z_D. The second procedure computes $Z_{EP_{\overline{x}}}$ which grows $prefix$ with the other items in the search space (excluding item x). Results from these two operations will then be combined in the resulting ZBDD output.

Before attempting to grow $prefix$ with the next item, x, the algorithm first tests whether the α and β prunings can be performed. Line 11 prunes $prefix_{new} = prefix \cup \{x\}$ and its supersets by the α-constraint pruning. The support of $prefix_{new}$ is calculated incrementally using *bitmap*($prefix, D$), which has been computed in previous recursion, i.e. support($prefix \cup \{x\}, D$) = the count of ones in *bitmap*($prefix, D$)$\cap$ *bitmap*($\{x\}, D$), where $D \in \{D_p, D_n\}$. Line 13 uses β-constraint pruning to stop $prefix_{new}$ from being grown. Finally, if none of these two cases holds, $prefix_{new}$ is grown further using projection of Z_D which does not contain x, storing the output in Z_{EP_x}.

Line 18 computes $Z_{EP_{\overline{x}}}$, the set of patterns which are supersets of $prefix$ but do not contain x. $Z_{EP_{\overline{x}}}$ is found from $Z_{D_{reduced}}$ which removes item x, obtained by computing the set-union between the two child-nodes of Z_D (line 17). The generated patterns from Z_{EP_x} and in $Z_{EP_{\overline{x}}}$ are locally minimal, but may not be globally minimal. Non-minimal pattern elimination is performed by a primitive ZBDD operation `NotSupSet`(P,Q) (line 19) which finds itemsets in P which are not supersets of any itemset in Q.

6.2.4 Optimisations for `EPMiner`

For the case where $\beta = 0$ (which corresponds to the common and important case of jumping emerging patterns), the computation of Z_{EP_x} (line 15 in the algorithm) can be optimised by passing it the minimal union between $Z_{D_{\overline{x}}}$ and Z_{D_x}, i.e. $Z_{D_{\overline{x}}} \bigcup_{Z_{min}} Z_{D_x}$. As a result, the computed Z_{EP_x} only contains patterns which may be non-minimal by the item $\{x\}$. Thus, line 19 in the algorithm can be replaced by a set-difference operation $Z_{EP_x} \setminus Z_{EP_{\overline{x}}}$ which is a less complex operation. This optimisation cannot be used in the general case when $\beta > 0$, since it could eliminate valid pattern candidates.

We investigated a number of heuristics for finding the optimal variable ordering for an efficient performance of `EPMiner`, using information about the item frequencies in D_p and D_n. Three alternative ordering strategies were worth considering:

- The first heuristic places the least frequent item in D_p at the top of the ZBDD, with subsequent items being ordered by increasing support in D_p. This aims to achieve early α support threshold pruning based on D_p, by locating items which are more likely to be pruned, higher up in the structure.

- The second heuristic places the least frequent item in D_n (i.e. occurring most frequently in $\overline{D_n}$) as the first item in the ZBDD, with other items being ordered by increasing frequency in D_n This can be justified on two levels. Firstly, consider line 15 in the algorithm. Having a smaller $Z_{D_{\overline{x}}}$ here is likely to be advantageous, particularly when the ZBDD at that point is large. Using the most frequent item in $\overline{D_n}$ at the top of the tree, means that $Z_{D_{\overline{x}}}$ is likely to be small for the early recursive calls. Secondly, this heuristic gives higher preference to the β constraint, in a similar manner to that for the α constraint in the first heuristic, the aim being to achieve early β-constraint pruning.

- The third heuristic places items from the same attribute close together because every (conjunctive) emerging pattern may contain at most one item from any attribute. Therefore, pattern candidates that contain two or more items from any attribute can be pruned earlier. Moreover, this ordering can be combined with the first or the second heuristic by ordering the items within each attribute by increasing support in D_p (based on the first heuristic) or in D_n (based on the second heuristic). The attributes are then ordered by increasing minimum support of its items.

6.2.5 Time Complexity of `EPMiner`

The `EPMiner` algorithm consists of two sub-procedures: i) finding patterns which contain an item x, where x is the top-variable in the input ZBDD Z_D which corresponds to the complemented projection of D_n, and ii) finding patterns which do not contain x. The first sub-procedure processes the 0-child of Z_D, and the second sub-procedure processes the union-ed ZBDD of the 0-child and the 1-child of Z_D. ZBDD's set-union operation has $O(|Z_D|)$ time complexity.

Theorem 15. *The total number of database projections, or the number of prefixes which are generated, is $O(Nlog_2N)$, and N^2 in the worst case, where N is the total size of nodes in the input ZBDDs.*

The support calculation for each prefix requires two bit-wise AND operations, one operation for calculating its support in D_p, and the other for calculating its support in D_n. The time complexity of each bit-wise AND operation is $O(|D_p|)$ and $O(|D_n|)$.

Theorem 16. *The overall time complexity of the bit-wise AND operations for all prefixes is $O(|D|log_2|Z_D|)$, where $|D| = |D_p| + |D_n|$ and $|Z_D|$ is the number of nodes in the ZBDD representing $\overline{D_n}$.*

6.2.6 More Efficient Support Checking Based on Weighted ZBDDs

As mentioned above, two bitwise-AND operations are needed to calculate the support of each prefix. This can be computationally expensive when many patterns exist. Some of them, however, may actually project the same conditional databases but it would not be identified from the bitmap representations.

As an alternative data representation, we can use the weighted ZBDDs to represent the two datasets, D_p and D_n, and project the respective conditional database for each prefix. Obtaining the support of a prefix is trivial once the conditional database has been projected. More particularly, the support of the prefix in a dataset is represented in the weight of the top-node in the weighted ZBDD representation of the conditional database.

The benefit of using the weighted ZBDD representation is that it may re-use the same conditional databases which are projected by another prefix through out the mining procedure. For a given dataset represented by a weighted ZBDD with N nodes, each database projection has $O(N)$ time complexity. The number of conditional database projections for each dataset through out mining is $O(N)$. Thus, the overall time complexity of the dataase projections for a dataset is $O(N^2)$. In practice, due to the caching principle, the computational cost may be much smaller than N^2, especially when many patterns share the same, or similar, prefixes.

6.3 Disjunctive Emerging Patterns

In this section we introduce a more general type of contrast pattern, which we will hereafter refer to as a *disjunctive emerging pattern*.

Recall that emerging patterns correspond to conjunctions of items that have high support in D_p and low support in D_n. Disjunctive emerging patterns generalise emerging patterns by allowing disjunctions as well as conjunctions for pattern descriptions. They essentially correspond to a restricted class of CNF formulae, which use items as variables and are a conjunction of disjunctions, where each disjunction contains only items coming from the same attribute domain. No negation is allowed and there must exist at least one item from each attribute domain in the formula.

Example 23. *Given a dataset having three attributes A, B, C, with domains $\{a_1, a_2, a_3\}$, $\{b_1, b_2, b_3\}$, $\{c_1, c_2, c_3\}$. A valid disjunctive emerging pattern might be represented by a formula f, where $f = (a_1 \vee a_3) \wedge (b_1 \vee b_2) \wedge (c_1 \vee c_2 \vee c_3)$. Without any ambiguity, we can alternately represent f as an itemset $\{a_1, a_3, b_1, b_2, c_1, c_2, c_3\}$, where it is implicitly understood that conjunctions exist across attributes and disjunctions exist within attributes.*

Henceforth, we will blur the distinction between disjunctive formulae and their itemset representations. Given a formula describing a disjunctive emerging pattern, we need to be able to calculate its support.

Definition 19. *Let s be a disjunctive emerging pattern. The support of s with respect to dataset D, $support(s, D)$, is the number of instances from D which are contained in (the itemset representation of) s.*

Using this revised definition of *support*, we can define appropriate α and β support thresholds for disjunctive emerging patterns.

Definition 20. *Given D_p, D_n and support thresholds α and β. A* **disjunctive emerging pattern** *is an itemset d such that*

tid	Attr. A	Attr. B	Attr. C
1	a_1	b_2	c_1
2	a_1	b_1	c_3
3	a_2	b_3	c_2
4	a_3	b_2	c_2

(a) Positive class

tid	Attr. A	Attr. B	Attr. C
1	a_1	b_3	c_1
2	a_2	b_1	c_2
3	a_2	b_3	c_2
4	a_3	b_2	c_1

(b) Negative class

Figure 6.1: A dataset example; the domain values of the attributes are $\{a_1, a_2, a_3\}$, $\{b_1, b_2, b_3\}$, $\{c_1, c_2, c_3\}$

- *d contains at least one item from the domain of every attribute*
- *$support(d, D_p) \geq \alpha$ and*
- *$support(d, D_n) \leq \beta$.*

d is said to be maximal if there does not exist another disjunctive emerging pattern d' such that $d \subset d'$.

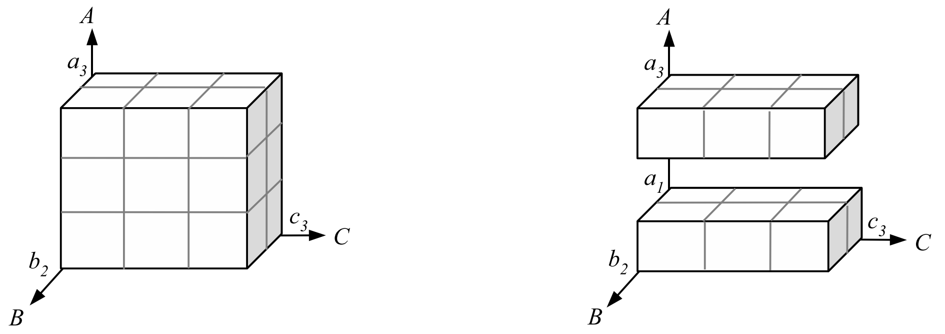
From a classification perspective, an unknown data instance seems more likely to be from the positive class if is contained in one of these sets, which we will study in the following chapter.

Example 24. *Consider the data set example in Table 6.1. Given $\alpha = 0.5$ and $\beta = 0$, the disjunctive emerging patterns include $\{a_1, b_1, b_2, c_1\}$, $\{a_1, b_1, b_2, c_1, c_2\}$, $\{a_1, b_1, b_2, c_1, c_2, c_3\}$, $\{a_1, a_3, b_1, b_2, b_3, c_3\}$, $\{a_1, a_2, a_3, b_1, b_2, b_3, c_3\}$, etc. The maximal disjunctive emerging patterns are $\{a_1, b_1, b_2, c_1, c_2, c_3\}$, $\{a_1, a_2, a_3, b_1, b_2, b_3, c_3\}$, $\{a_1, a_3, b_1, b_2, b_3, c_2, c_3\}$, $\{a_1, a_2, b_1, b_2, c_1, c_3\}$.*

One important case arises for datasets with ordered domains. Observe that a disjunctive emerging pattern corresponds to a region of high contrast. That is, a subspace which contains α instances from D_p and at most β instances from D_n. Suppose an attribute A_i has an ordered domain of items. An itemset is a contiguous subset of $dom(A_i)$ if it contains a collection of items appearing consecutively in the order of $dom(A_i)$. Hence, it is possible to define variants of disjunctive emerging patterns, such as contiguous disjunctive emerging patterns.

Definition 21. *Given datasets D_p and D_n, an itemset S is a* **contiguous disjunctive emerging pattern** *if it satisfies the following three conditions*

1. *$support(S, D_p) \geq \alpha$*



(a) $s_1 = \{a_1, a_2, a_3, b_1, b_2, c_1, c_2, c_3\}$ (b) $s_2 = \{a_1, a_3, b_1, b_2\}$

Figure 6.2: Geometric Representation of Disjunctive Emerging Patterns

2. *$support(S, D_n) \leq \beta$*

3. *S is contiguous*

Moreover, S is a **maximal contiguous disjunctive emerging pattern** *if there is no proper superset of S satisfying conditions 1-3.*

Compared to the more general disjunctive emerging patterns, contiguous disjunctive emerging patterns might be considered more meaningful to humans, since their corresponding regions cannot contain any gaps or holes, i.e. they must be connected.

Example 25. *In Figure 6.2a shows a contiguous pattern s_1, given ordered attribute domains $\{a_1, a_2, a_3\}$, $\{b_1, b_2, b_3\}$, $\{c_1, c_2, c_3\}$, such that $s_1 = (a_1 \vee a_2 \vee a_3) \wedge (b_1 \vee b_2) \wedge (c_1 \vee c_2 \vee c_3)$.*

Figure 6.2b shows a non-contiguous pattern s_2, such that $s_2 = (a_1 \vee a_3) \wedge (b_1 \vee b_2) \wedge (c_1 \vee c_2 \vee c_3)$. It is non-contiguous since its range of values on the first attribute, i.e. $\{a_1, a_3\}$, is not contiguous.

6.3.1 Relationships Between Disjunctive Emerging Patterns and Emerging Patterns

We now examine the relationship between disjunctive emerging patterns and the emerging patterns in more detail. Broadly speaking, disjunctive emerging patterns can be viewed as generalisations of emerging patterns, allowing more expressive contrasts.

The following theorem shows how disjunctive emerging patterns generalise emerging patterns.

Theorem 17. *Let p be an emerging pattern. Then p is contained in some disjunctive emerging pattern using the same α and β support thresholds.*

Observe that the converse of this theorem does not hold. It is often true that a disjunctive emerging pattern does not contain any emerging patterns.

Example 26. *Recall the example data set in Figure 6.1. Given $\alpha = 0.5$ and $\beta = 0$, emerging patterns do not exist, but several disjunctive emerging patterns exist.*

Also observe that multiple emerging patterns of lower support can be merged together to form a disjunctive emerging pattern. Again looking at Figure 6.1, both $\{a_1, b_1\}$ and $\{a_1, b_2\}$ are emerging patterns when $\alpha = 0.25$ and $\beta = 0$ and correspond to the to the boolean formulae $f_1 = a_1 \wedge b_1$ and $f_2 = a_1 \wedge b_2$, each having support in D_p of 0.25. These two emerging patterns can be *unioned* to yield $a_1 \wedge (b_1 \vee b_2)$, which is equivalent to the disjunctive emerging pattern $a_1 \wedge (b_1 \vee b_2) \wedge (c_1 \vee c_2 \vee c_3)$ (since $c_1 \vee c_2 \vee c_3$ is trivially true for any transaction), with support 0.5 with respect to D_p and zero with respect to D_n.

An interesting special case exists when the cardinality of the domain for every attribute is exactly two. In this circumstance, all disjunctive emerging patterns are guaranteed to be emerging patterns.

To summarise, the key differences between emerging patterns and disjunctive emerging patterns are:

- Disjunctive emerging patterns are more expressive. They can capture contrast regions of greater complexity. This makes them more suitable for ordered data, where it is frequently desirable for the contrasts to include disjunctions of items within specific dimensions

- For a given threshold α and β, it is often the case that a dataset may have many disjunctive emerging patterns but no emerging patterns

Of course, being more expressive, makes disjunctive emerging patterns more complex to compute. However, it turns out we can still accomplish this efficiently using a ZBDD technique similar to the `EPMiner` algorithm presented in Section 6.2.3.

6.3.2 The Algorithm of `DEPMiner` for Mining Maximal Disjunctive Emerging Patterns

We now describe our `DEPMiner` algorithm for mining maximal disjunctive emerging patterns, shown in Algorithm 6.2, which is adopted from `EPMiner` algorithm (Algorithm 6.1). Being similar, we will highlight only the main differences between `DEPMiner` and `EPMiner`.

Because of the generality of disjunctive emerging patterns, `DEPMiner` explores the search space in a depth-first top-down manner, rather than the bottom up manner that was used for emerging patterns. We start with the most general itemset (i.e. containing all items) and at each step, generate shorter itemsets as candidates. For efficiency purposes though, since disjunctive patterns are likely to contain many items, it is better to work with pattern complements, which are likely to be short (i.e. containing fewer items), rather than the patterns themselves. So, generating shorter candidates is obtained by growing a prefix bottom up in this complemented pattern space. The initial input ZBDD is built from D_n, as opposed to $\overline{D_n}$, used in `EPMiner`. Again this aims to eliminate the generation of invalid candidates, but D_n is used here instead of $\overline{D_n}$, since the enumeration of maximal disjunctive EPs is proceeding top-down, rather than bottom-up. Figure 6.3 shows an example of pattern lattice for a given set of items $I = \{a, b, c, d\}$. A bottom-up enumeration begins with the empty set $\{\}$ and generates the longer itemsets $\{a\}, \{a, b\}$, etc. as subsequent candidates. A top-down enumeration begins with the complete set $\{a, b, c, d\}$ and generates the shorter itemsets, i.e. $\{a, b, c\}, \{a, b\}$, as subsequent candidates.

The pruning based on the α and β constraints used in `DEPMiner` is similar to that in `EPMiner`. Support checking, however, must be done using pattern complements and so intersection tests, rather than containment tests are performed on the bitmaps for D_p and D_n. e.g. If a disjunctive emerging pattern p is required to have $support(p, D_n) \leq \beta$, then its complement $\overline{p}$ must intersect with at least $|D_n| - \beta$ transactions, denoted by $cover(\overline{p}, D_n) \geq |D_n| - \beta$. Similarly for the α constraint, $support(p, D_p) \geq \alpha \equiv cover(\overline{p}, D_p) \leq |D_p| - \alpha$. Thus, the conditions for α and β pruning are inverted from that in `EPMiner`, since maximal patterns, rather than minimal, are being computed.

The algorithm is initialised by passing the negative dataset D_n to its first parameter, i.e. `DEPMiner`$(D_n, \{\}, D_p, D_n, \alpha, \beta)$. Terminal cases of the algorithm are the following:

- if $support(\overline{prefix_{new}}, D_p) < \alpha$, i.e. $cover(prefix_{new}, D_p) > |D_p| - \alpha$, then per-

Algorithm 6.2 DEPMiner(Z_D, $prefix$, D_p, D_n, α, β)

Call DEPMiner(D_n, {}, D_p, D_n, α, β) to begin mining initially.

Input: A ZBDD Z_D containing a projection of the negative data set, a prefix itemset $prefix$ which projects Z_D, a bitmap D_p of the positive data set, a bitmap D_n of the negative data set, a min support threshold α, a max support threshold β.

Output: A ZBDD Z_{DEP} containing the set of minimal emerging patterns (i.e. itemsets p s.t. $cover(p, D_p) \leq |D_p| - \alpha$ and $cover(p, D_n) \geq |D_n| - \beta$).

```
1: if ($Z_D$ is a sink node) then
2:   if ( $cover(prefix, D_p) \leq |D_p| - \alpha$ and $cover(prefix, D_n) \geq |D_n| - \beta$ ) then
3:     return 1; /* $prefix$ satisfies the support constraints */
4:   else
5:     return 0; /* Remove $prefix$ and its supersets from output */
6:   end if
7: end if
8: /* Let $Z_D = node(x, Z_{D_x}, Z_{D_{\overline{x}}})$ */
9: $prefix_{new} = prefix \cup \{x\}$ /* Grow new prefix with the next item in the search
   space */
10: if ($cover(prefix_{new}, D_p) > |D_p| - \alpha$) then
11:   $Z_{DEP_x} = 0$ /* Prune $prefix_{new}$ by $\alpha$ constraint pruning */
12: else if ($cover(prefix_{new}, D_n) \geq |D_n| - \beta$) then
13:   $Z_{DEP_x} = 1$ /* Prune supersets of $prefix_{new}$ by $\beta$ constraint pruning */
14: else
15:   $Z_{DEP_x}$ = DEPMiner($Z_{D_{\overline{x}}}, prefix_{new}, D_p, D_n, \alpha, \beta$) /* Mine supersets of
      $prefix_{new}$ from the projection of $Z_D$ which does not contain $x$ */
16: end if
17: $Z_{D_{reduced}} = Z_{D_{\overline{x}}} \bigcup_Z Z_{D_x}$ /* Remove $x$ from the search space by computing set-union
    between the two children of $Z_D$ */
18: $Z_{DEP_{\overline{x}}}$ = DEPMiner($Z_{D_{reduced}}, prefix, D_p, D_n, \alpha, \beta$) /* Explore candidates from the
    remaining search space */
19: $Z'_{DEP_x}$ = NotSupSet($Z_{DEP_x}, Z_{DEP_{\overline{x}}}$) /* Non-minimal patterns pruning */
20: $Z_{DEP}$ = getNode($x, Z'_{DEP_x}, Z_{DEP_{\overline{x}}}$) /* Combine the output patterns */
21: return $Z_{DEP}$
```

Note: $cover(p, D)$ is the number of transactions in D which contain any item in p; $cover(p, D) = |D| - support(\overline{p}, D)$.

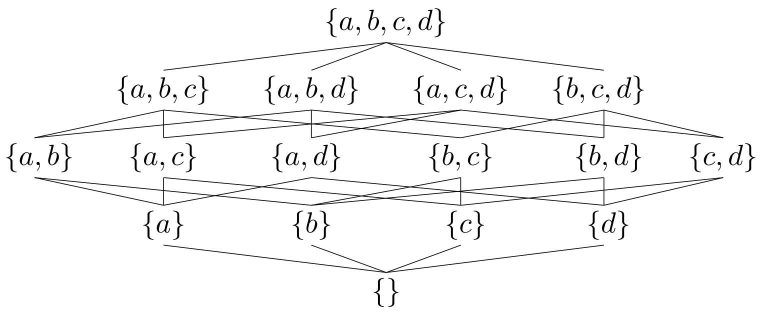

$\{a,b,c,d\}$
$\{a,b,c\}$ $\{a,b,d\}$ $\{a,c,d\}$ $\{b,c,d\}$
$\{a,b\}$ $\{a,c\}$ $\{a,d\}$ $\{b,c\}$ $\{b,d\}$ $\{c,d\}$
$\{a\}$ $\{b\}$ $\{c\}$ $\{d\}$
$\{\}$

Figure 6.3: Example of pattern lattice for $I = \{a, b, c, d\}$.

form α constraint pruning (line 11)

- if $support(\overline{prefix_{new}}, D_n) \leq \beta$, i.e. $cover(prefix_{new}, D_n) \geq |D_n| - \beta$, then perform β constraint pruning (line 13)
- if $support(\overline{prefix}, D_n) \leq \beta$, i.e. $cover(prefix, D_n) \geq |D_n| - \beta$, then remove $prefix$ from output (line 1-7)

The ordering used for the ZBDD is to put the most frequent item in D_n at the top and items are ordered in decreasing frequency in D_n thereafter. This is essentially the inverse of the second ordering heuristic that was used for EPMiner, again being due to the top-down nature of the search strategy.

The time complexity of DEPMiner is similar to EPMiner, but DEPMiner finds the pattern complements which are longer than the patterns themselves. However, the input to DEPMiner is the negative dataset in its positive form which is equal to or smaller (i.e. each transaction contains fewer items) than its complemented form. Therefore, DEPMiner performs more projections of the input database Z_D than EPMiner, but the projected databases are smaller.

Theorem 18. *The DEPMiner algorithm (Algorithm 6.2) is sound and complete.*

Proof. We will prove this theorem in two parts: (a) the algorithm is sound, (b) the algorithm is complete, based on the divide-and-conquer strategy of the algorithm.

Soundness: First, we will show that growing a prefix by a chosen item x and consequently restricting mining to the conditionally projected negative data set $Z_{D_{\overline{x}}}$ (line 15), correctly generates disjunctive emerging patterns which exclude item x. By growing a prefix itemset, DEPMiner implicitly restricts an item from the domain items

I and generates a candidate for a disjunctive emerging pattern. Since the candidate pattern is a complemented itemset of the prefix, the support of the candidate in a dataset is equivalent to the number of transactions which do not contain any item in the prefix. Should the prefix be grown by item x, the conditional database $Z_{D_{\overline{x}}}$ corresponds to itemsets in the negative data set D_n which do not contain x (line 15). This database projection narrows the search space to those transactions in D_n which do not contain x and any other items in the prefix itemset. If the projected database is empty, then, the candidate pattern (i.e. the complemented prefix itemset) has 0 support in D_n. The terminal cases of the algorithm (line 1-7) return the fully grown prefix as a disjunctive emerging pattern if it satisfies the support constraints. Thus, the algorithm is correct.

Completeness: We will show that all disjunctive emerging patterns which do not contain item x are found from the conditional database $Z_{D_{\overline{x}}}$. When projecting the conditional database, the itemsets of Z_D which contain x are removed. This database projection is complete, since any disjunctive pattern which does not contain x can only have support in the transactions which do not contain x. The supporting transaction must be a subset of the pattern. Thus, no valid candidates are omitted by the conditional database projection, and the algorithm is complete. □

6.3.3 The Algorithm of `pseudoContigDEP` for Mining Pseudo-Contiguous Disjunctive Emerging Patterns

As we have seen, contiguous disjunctive emerging patterns are a subclass of disjunctive emerging patterns. We introduce our strategy for mining pseudo-contiguous patterns, that is, patterns which contain gaps constrained by a maximum gap size *maxGap*. Contiguous patterns correspond to patterns satisfying *maxGap* = 0 constraint. The technique employs two steps:

1. Mine the disjunctive emerging patterns using `DEPMiner` algorithm (Algorithm 6.2)

2. Use post processing on the set of disjunctive emerging patterns to derive the set of contiguous disjunctive emerging patterns. This is done using a ZBDD-based operation which splits each disjunctive emerging pattern into a maximal contiguous pattern and then retains those patterns that satisfy the α threshold (all such patterns are guaranteed to satisfy the β threshold).

The ZBDD post processing operation `pseudoContigDEP` is described in Algorithm 6.3. The main idea is that every disjunctive emerging pattern has a set of pseudo-contiguous splits induced for each dimension. Then, the pseudo-contiguous splits across dimensions are conjugated using an efficient ZBDD pair-wise unions operation, `DotProd`. The checking of the α support constraint is also pushed inside the ZBDD routines, in a similar manner to that of the algorithm that was used for mining disjunctive emerging patterns.

Now we explain the `pseudoContigSubsets` operation, shown in Algorithm 6.4, for splitting a dimension of a given pattern into subsets which satisfy a *maxGap* constraint. The algorithm is initialised by setting p to be an itemset projection of a pattern in the i-th dimension, *dom* to be the domain items of the i-th dimension. $prefix$ is initially an empty itemset $\{\}$. Each of itemsets p, and *dom*, is represented as a ZBDD. *gapSize* is the gap between the last item in $prefix$ and the first item in *dom*. *gapSize* is initially 0. The two itemsets p and *dom* are traversed in parallel. Given that p is a complement of a disjunctive emerging pattern, $prefix$ is the partially grown pseudo-contiguous split from $\overline{p}$ which contains items in *dom* which do not occur in p (line 6-7). Every sequential item that occurs in p increases *gapSize* (line 9). If *gapSize* is equal to *maxGap* then $prefix$ is stopped from growing and returned as a pseudo-contiguous split (line 11-12). The split procedure continues with the remaining items in p. When p is empty, the pair-wise union between *dom* and $prefix$ is returned as output (line 2).

Example 27. *Suppose `pseudoContigSubsets` is given an input itemset $p = \{a_2, a_3\}$ and a set of domain items dom $= \{a_1, a_2, a_3, a_4\}$. Suppose maxGap $= 1$. Since the first item in dom, i.e. a_1, does not occur in p, a new prefix $\{a_1\}$ is grown. The next item in dom, i.e. a_2, occurs in p, which increases the gap size to 1. The following item is a_3, which also occurs in p. The gap size is now 2 while the prefix itemset remains to be $\{a_1\}$. Since the gap is exceeding its maximum size, it is not possible to grow prefix $\{a_1\}$ with other items without exceeding the maximum gap size. Thus, $\{a_1\}$ is a valid pseudo-contiguous split. Another pseudo-contiguous split from $\overline{p}$ is $\{a_4\}$.*

The post-processing operation `pseudoContigDEP` iterates through each pattern in the set of maximal disjunctive emerging patterns. And for each pattern, it iterates through each item across all of the attribute domains. Suppose N is the total number of items in the data set. Given that the pattern set is represented by a ZBDD, we will firstly analyse the complexity to extract an itemset from a ZBDD, and then the complexity of finding the pseudo-contiguous subsets for each itemset.

Algorithm 6.3 `pseudoContigDEP`(P,[$dom_1, dom_2, ...dom_k$], $maxGap$)

Input: A ZBDD P containing the complements of maximal disjunctive emerging patterns, a vector of ZBDDs [$dom_1, dom_2, ...dom_k$] representing the domain of each attribute, and a maximum gap size $maxGap$

Output: A ZBDD $zOut$ containing the set of maximal pseudo-contiguous disjunctive emerging patterns

```
1: $zOut = \{\}$ /* Initialise the output */
2: for all $p$ in $P$ do
3:     $partialSplits = \{\{\}\}$ /* Initialise the set of pseudo-contiguous splits for pattern $p$ */
4:     for all $i$ in 1,..$k$ do
5:         $project_i$ = CrossProd($p, dom_i$) /* Find the projection of $p$ in the $i$-th dimension */
6:         $splits_i$ = pseudoContigSubsets($project_i, dom_i, \{\}, 0$) /* Find pseudo-contiguous splits of $p$ in the $i$-th dimension */
7:         $partialSplits$ = DotProd($partialSplits, splits_i$) /* Conjugate the pseudo-contiguous splits across all dimensions of pattern $p$ */
8:     end for
9:     $zOut = zOut \bigcup_{Zmax} partialSplits$ /* Combine the pseudo-contiguous splits across all patterns in $P$ and remove the non-maximal patterns */
10: end for
11: return $zOut$
```

Algorithm 6.4 `pseudoContigSubsets`(p, dom, $prefix$, $gapSize$)

Input: a ZBDD p containing an itemset, a ZBDD dom containing an attribute domain ZBDD (i.e. $p \subseteq dom$), a ZBDD $prefix$ containing a partially grown pseudo-contiguous subset, and the size of the partially grown gap $gapSize$ that follows after the last item in $prefix$.

Output: a ZBDD $zOut$ containing the set of pseudo-contiguous subsets of $\overline{p}$ w.r.t. dom satisfying the $maxGap$ constraint

```
1:  if ($p$ is a sink node) then
2:     $zOut$ = DotProd($dom$,$prefix$) /* $p$ is empty, the pair-wise union between $dom$
       and $prefix$ is returned */
3:  else
4:     /* Let $p = node(x, p_1, p_0)$, $dom = node(y, dom_1, dom_0)$, where $p_0 = 0$ and
       $dom_0 = 0$, since each of $p$ and $dom$ contains only one itemset */
5:     if ($x > y$) then
6:        $prefix_y$ = change($prefix, y$) /* The first item $x$ in $p$ have a higher index (w.r.t
          the variable ordering) than the first item $y$ in $dom$, i.e. $y$ is not int $p$, so $prefix$
          is grown by $y$ */
7:        $zOut$ = pseudoContigSubsets($p, dom_1, prefix_y, 0$) /* Try to grow the new
          prefix $prefix_y$ with $gapSize$ reset to 0 */
8:     else if ($x = y$ and $gapSize < maxGap$) then
9:        $zOut$ = pseudoContigSubsets($p_1, dom_1, prefix, gapSize$+1) /* Item $x$ has
          the same index than $y$; since gap is allowed, increment $gapSize$ and try to
          grow $prefix$ using the remaining items in $p_1$ */
10:    else if ($x = y$ and $gapSize = maxGap$) then
11:       $zRem$ = pseudoContigSubsets($p_1, dom_1, \{\}, 0$) /* Item $x$ has the same index
          than $y$; since no larger gap is allowed, $prefix$ is a complete pseudo-contiguous
          split; try to find other pseudo-contiguous splits using the remaining items in
          $p_1$ */
12:       $zOut = prefix \bigcup_Z zRem$ /* Pair-wise union between the pseudo-contiguous
          splits */
13:    end if
14: end if
15: return $zOut$
```

The ZBDD's operation to extract an itemset from a ZBDD has $O(h)$ time complexity where h is the height of the ZBDD, or the maximum number of items in any path of the ZBDD, which is less than N. For each of the extracted itemsets, the time complexity of `pseudoContigSubsets` operation is $O(N)$ since it iterates through all items across all attribute domains.

Theorem 19. *Given k maximal disjunctive emerging patterns, where L is the number of items in the longest pattern, and N is the number of items, the overall time complexity of `pseudoContigDEP` operation is $O(kLN)$*

6.4 Performance Study

In this section we evaluate our techniques for mining emerging patterns and disjunctive emerging patterns. All algorithms were implemented in C++, and all experiments were conducted on a 2.0 GHz CPU, 3 GB RAM, running Solaris, with a cpu-time limit 100,000 seconds. We carried out experiments on three gene-expression data sets[1], the Leukaemia data set ALL-AML (previously studied in [84]), lung cancer data set, and colon tumor data set. Table 6.1 shows their characteristics. Column 1 (resp. Column 2) shows the number of instances in the positive (resp. negative) class. These data sets were chosen due to their challenging characteristics. As is common for biological data, they contain many dimensions but only have a few instances. Work in [86, 88, 84] has previously studied mining minimal emerging patterns for these data sets.

All of the data sets were discretised using an entropy discretisation method, which had the effect of removing some of the attributes. After discretisation, the ALL-AML data set reduced to 865 attributes, the lung cancer reduced to 2172 attributes, and the colon tumor reduced to 135 attributes. We also ordered the discretised attributes according to their entropy value from highest to lowest (i.e. attribute1 has highest entropy, attribute2 as second highest, etc).

6.4.1 Effects of Variable Ordering to the Performance of `EPMiner`

We firstly study the effects of using various different variable orderings in the ZBDD to the performance of `EPMiner` for mining emerging patterns. The three figures in Figure 6.4 show a comparison between the different variable ordering heuristics we

[1]http://research.i2r.a-star.edu.sg/rp/

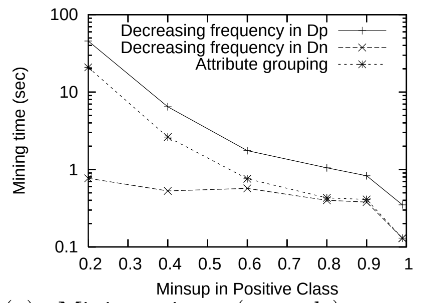
Table 6.1: Data Characteristics

Data set	$\|D_p\|$	$\|D_n\|$	# of attr.	# of attr. after discretisation (# of items)
ALL-AML	27	11	7129	865 (1700)
lung cancer	16	16	12533	2172 (4371)
colon tumor	20	42	2000	135 (270)

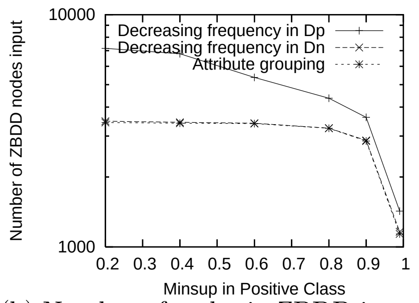
considered for the ZBDD. The first heuristic is employed by ordering the variables by their decreasing frequency in D_p. The second heuristic is employed by ordering the variables by their decreasing frequency in D_n. Lastly, the third heuristic is employed by arranging items from the same attribute close to each other and two-level ordering is used, i.e. items within each attribute are ordered by decreasing support in D_n and the attributes are ordered by decreasing maximum support of its items.

Figure 6.4a shows that employing the second ordering on the ZBDDs achieves the fastest mining time as it reduces the complexity of the decomposed subtasks. Figure 6.4b shows the corresponding size (number of nodes) of the input ZBDDs. The input ZBDDs have similar sizes using either the second or the third ordering, which shows that the input ZBDD sizes are not much influenced by the grouping of items within attributes. The algorithm's mining time with the third ordering, however, is slower than that using the second ordering, and the corresponding mining times for the third ordering grow exponentially as α decreases. It shows that the attribute-grouping of the items limits the pruning effectiveness of the algorithm, due to the larger conditional databases being projected.

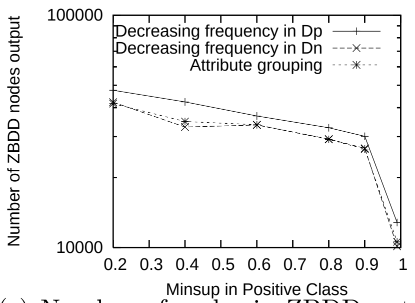
Using the first ordering, the input ZBDDs are larger compared to the other orderings, since the input ZBDDs are built based on the negative class D_n while the first variable ordering heuristic is based on the positive class D_p. This big start-up also results in its mining time being the slowest compared to the other orderings. Moreover, there is only a small difference between all the three orderings in terms of the corresponding sizes of the output ZBDDs, shown in Figure 6.4c, with the second and the third ordering giving the smallest ZBDDs. This shows that the second and third variable orderings allow similar compression for representing the emerging patterns. Based on this comparison, therefore, the following experiments are performed using the second variable ordering heuristic.



(a) Mining time (seconds) w.r.t. min. support in D_p



(b) Number of nodes in ZBDD input w.r.t. min. support in D_p



(c) Number of nodes in ZBDD output w.r.t. min. support in D_p

Figure 6.4: Comparison between different variable orderings from mining emerging patterns in lung cancer data set with 500 attributes, $\beta = 0$

6.4.2 Scalability of `EPMiner`

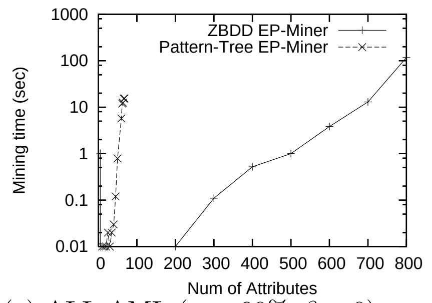
We study the scalability of our ZBDD-based technique for mining (minimal) emerging patterns, and compare it against a baseline emerging pattern mining technique which is based on a variant of frequent pattern trees [51] (hereafter referred to as `Pattern-Tree EPMiner`, whose implementation was obtained from its original author). Other techniques for mining emerging patterns exist (e.g. [16, 14]), but have similar, or inferior running behaviour to that of [51] for our data sets and so we do not include them in our comparison.

We compare the scalability of our `EPMiner` algorithm, which is labelled as `ZBDD EPMiner`, against the `Pattern-Tree EPMiner` [51], by running it on the ALL-AML data set, using constraints $\alpha = 90\%$ and $\beta = 0$, and increasing the data dimensionality (i.e. number of dimensions). Figures 6.5a and 6.5b show that the mining time of `ZBDD EPMiner` is substantially faster than `Pattern-tree EP-miner`, being approximately 100 times faster for 40-68 attributes (the running time of `ZBDD EPMiner` is very close to the x-axis in these scenarios). Moreover, the `ZBDD EPMiner` was able to run effectively for up to 800 attributes, whereas `Pattern-Tree EPMiner` could not complete mining for more than 68 attributes, due to the memory limits being exceeded. This is in line with previously published results from [84, 38], where emerging patterns were previously only mineable for data sets with no more than 70 attributes.

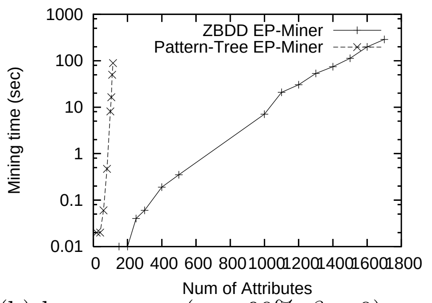
Figure 6.5b shows the running time for the lung cancer data set. Emerging patterns in this data set appear to be easier to mine due to the relatively small number of patterns. The `Pattern Tree EPMiner` is able to mine emerging patterns for a larger number of attributes, compared to that for the ALL-AML data set. Moreover, the `ZBDD EP-miner` was able to run effectively for up to 1700 attributes. It is substantially superior in running time compared to the `Pattern-Tree EP-miner`, giving speedups of over 100 times.

Similar behaviour exists in the colon tumor data set, which is smaller than the other two data sets. Figure 6.5c shows that the `ZBDD EPMiner` is able to mine emerging patterns using all attributes, achieving up to 200 times speedup factor.

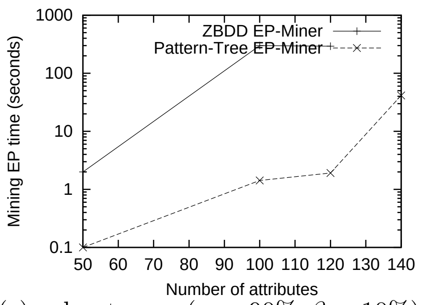
Secondly, we also compare the performance between using Weighted ZBDD and ZBDD for mining the emerging patterns. Interestingly, `WZBDD EPMiner` does not always outperform `ZBDD EPMiner`. Figures 6.6a and 6.6c show that `WZBDD EPMiner` is superior for the ALL-AML and colon tumor data set, achieving a significant speedup for the colon tumor data set. However, Figure 6.6c shows that it is slower than `ZBDD EPMiner`



(a) ALL-AML ($\alpha = 90\%, \beta = 0$)



(b) lung cancer ($\alpha = 90\%, \beta = 0$)



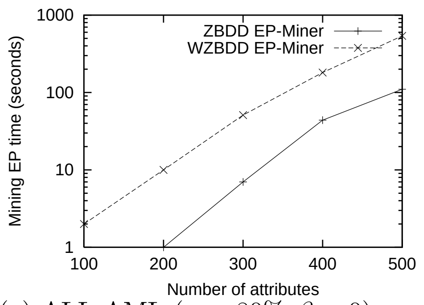
(c) colon tumor ($\alpha = 90\%, \beta = 10\%$)

Figure 6.5: Comparison between the mining time using ZBDD or Pattern Tree for mining emerging patterns, w.r.t various data dimensionalities (i.e. number of attributes)

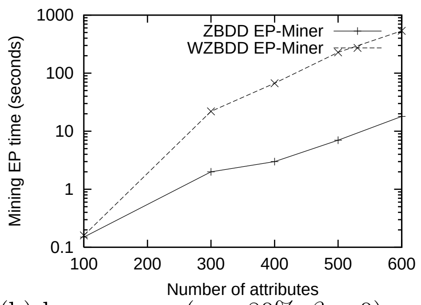
for the lung cancer data set. It shows that the bitmap operation for frequency counting performed by `ZBDD EPMiner` is not too expensive as long as the size of the positive class is small.

6.4.3 Effects of Varying Support Threshold to the Performance of `EPMiner`

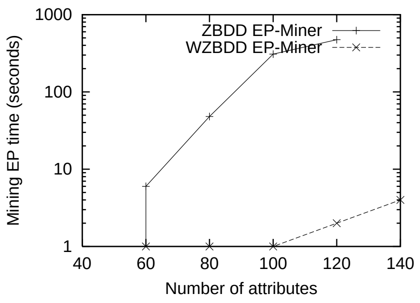
Figure 6.7 shows the run time of all the EPMiner algorithms with respect to a varying minimum support threshold in the positive class. The `Pattern Tree EPMiner` is the fastest for the ALL-AML dataset when the minimum support threshold is high, but it is slowest for all three data sets when the minimum support threshold is low. It shows that the run time of `Pattern Tree EPMiner` grows exponentially as the minimum support threshold decreases. In general, `WZBDD EPMiner` is shown to be less sensitive to the minimum support threshold. Figures 6.7a and Figure 6.7b show that compared with `ZBDD EPMiner`, `WZBDD EPMiner` is faster for the ALL-AML data set, but slower for the lung cancer data set except when the minimum support threshold is very low (i.e. 0.1).



(a) ALL-AML ($\alpha = 80\%, \beta = 0$)

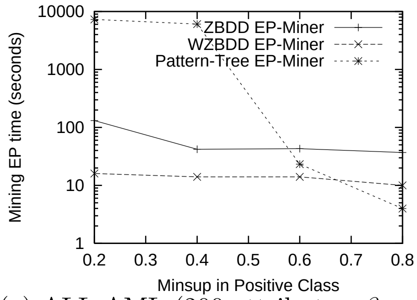


(b) lung cancer ($\alpha = 80\%, \beta = 0$)

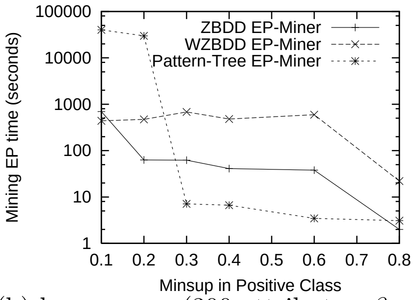


(c) colon tumor ($\alpha = 90\%, \beta = 10\%$)

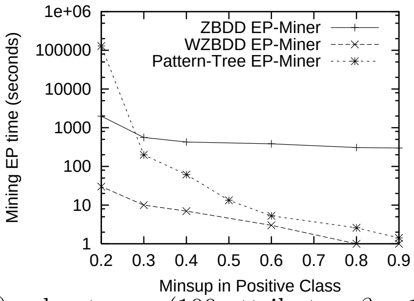
Figure 6.6: Comparison between the mining time using weighted ZBDD, or ZBDD for mining emerging patterns, w.r.t various data dimensionalities (i.e. number of attributes)



(a) ALL-AML (200 attributes, $\beta = 0$)



(b) lung cancer (300 attributes, $\beta = 0$)



(c) colon tumor (100 attributes, $\beta = 10\%$)

Figure 6.7: Comparison between the mining time using ZBDD, weighted ZBDD, or Pattern Tree for mining emerging patterns, w.r.t minimum support threshold *minsup* in the positive class

6.4.4 Effects of Varying Support Threshold to the Performance of `DEPMiner`

The evaluation performed in this subsection aims to study the comparative output of the disjunctive emerging patterns over emerging patterns, and the required mining times of those patterns, given the correctness and completeness of `DEPMiner` are theoretically proven. Evaluation on the effectiveness of disjunctive emerging patterns, such as their performance for classification will be studied in the following chapter.

Before studying the runtime performance of `DEPMiner`, we will firstly study the output sizes. Figures 6.8a, 6.8c, and 6.8e show the output patterns in the lung cancer data set (using 115 attributes), in the ALL-AML data set (using 300 attributes), and in the colon tumor data set (using 100 attributes) with respect to different values of minimum support α (given $\beta = 0$). We can see that for the lung cancer data set, with a relatively small number of attributes, the number of patterns output is not as many as those in the other data sets. Morever, the colon tumor data set contains the most number of patterns compared to the other data sets, containing up to 700,000 patterns when the minimum support is 0.2.

We also compare the relative volumes of patterns that are output, for a given data set. Figure 6.8a shows the relative number of i) minimal emerging patterns, ii) maximal disjunctive patterns, iii) maximal contiguous disjunctive emerging patterns, in the lung cancer data set. For this scenario, it is clear that the number of non-disjunctive EPs is fewer than the disjunctive EPs and their contiguous variants. This is expected, since emerging patterns are more specific versions of either type of disjunctive patterns. Though it is not shown here, in our experience, it can often be the case that under given support thresholds, a data set may contain zero EPs, but may contain many hundreds of (possibly contiguous) disjunctive EPs. In the ALL-AML and the colon tumor data sets, the number of disjunctive EPs is not much greater than the number of EPs and contiguous disjunctive EPs.

The corresponding mining times for all data sets are shown in Figures 6.8b. 6.8d, and 6.8f. For the lung cancer data set, the mining times of `EPMiner` and `DEPMiner` lie on the X-axis, while the times for mining contiguous disjunctive EPs are higher due to the post-processing `pseudoContigSplit` operation. It can be seen that the splitting time is constant with respect to a varying number of patterns from varying α. Indeed, all the algorithms have a roughly constant time with respect to α for this scenario. For the other data sets, the mining time for finding contiguous disjunctive EPs increases exponentially as the minimum support decreases, whereas the mining time for finding disjunctive EPs is always faster than the mining time for finding EPs.

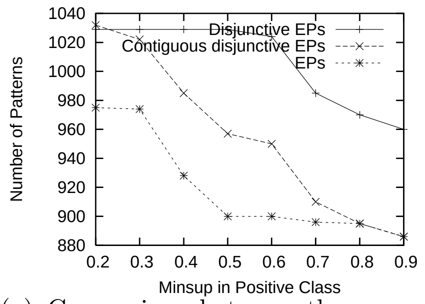
6.4.5 Scalability of `DEPMiner`

We now study the scalability of our ZBDD-based algorithm `DEPMiner` for mining maximal disjunctive emerging patterns, either using ZBDD or Weighted ZBDD (WZBDD), which are labeled as `ZBDD DEPMiner` and `WZBDD DEPMiner`, respectively. In particular, we compare their behaviour as we vary number of attributes and the minimum support α in D_p. No comparison is made against other systems, since we are not aware of any other work that is suitable for mining these patterns.

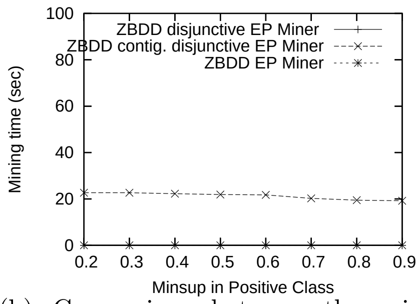
The number of patterns output with respect to an increasing number of attributes, for the three data sets are shown in Figure 6.9. Their corresponding mining times are shown in Figure 6.10 The thresholds of $\alpha = 90\%$ and $\beta = 10\%$ are used for the ALL-AML data set and the lung cancer data set; the thresholds of $\alpha = 90\%$ and $\beta = 0\%$ are used for the colon tumor data set.

Figures 6.9a and 6.9b show that the ALL-AML data set contains many more patterns than the lung cancer data set, although it has lower dimensionality. Figure 6.9c, moreover, shows that the colon tumor data set contains more patterns than the ALL-AML data set. We consider the lung cancer data set sparse, and the colon tumor data set dense. Since the patterns being output are stored in a ZBDD, it is interesting to reflect on the compression this structure it can achieve. Figure 6.9d shows the number of ZBDD nodes in the output, for the lung cancer data with respect to a varying dimensionality, i.e. number of attributes. When 2172 attributes are present, the ZBDD requires 1,236,100 nodes, to store the 2,080,960 maximal disjunctive emerging patterns. Figure 6.9e gives a more detailed picture, presenting a length histogram of the patterns. We can see that most of the patterns are close to the maximum length of 4371 items having an average length of around 4367 items.

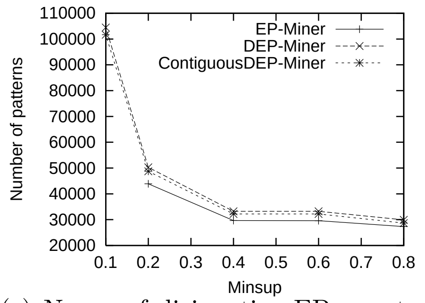
More importantly, it shows that our `DEPMiner` algorithm is able to mine these complex kinds of patterns even when there are a very large number of attributes, e.g. 600 attributes from the ALL-AML data set which contain about half a million patterns (Figure 6.9a). Figure 6.10a shows that `WZBDD DEPMiner` is able to be 5 times faster than `ZBDD DEPMiner` for the ALL-AML data set. Figure 6.10b shows that `WZBDD DEPMiner` is 10 times faster than the `ZBDD DEPMiner` Moreover, Figure 6.10c shows that `WZBDD DEPMiner` achieved 100 times speed up for the colon tumor data set.



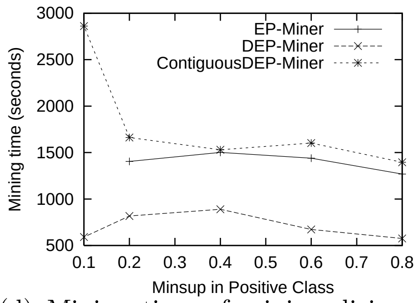
(a) Comparison between the num. of EPs, Disjunctive EPs, and Contiguous Disjunctive EPs w.r.t min supp in D_p, lung cancer data set (115 attr.; $\beta = 0$)



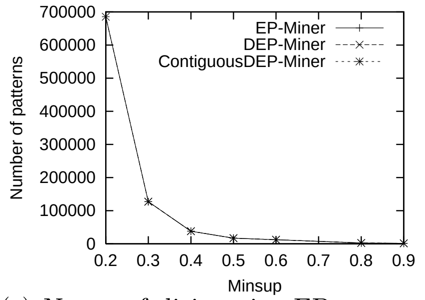
(b) Comparison between the mining time of `ZBDD EPMiner`, `ZBDD DEPMiner`, and `ZBDD Contiguous DEP Miner` w.r.t min supp in D_p, lung cancer data set (115 attr.; $\beta = 0$)



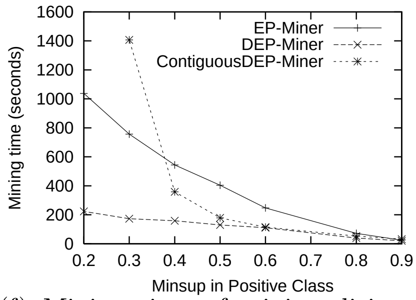
(c) Num. of disjunctive EPs w.r.t min supp in D_p for ALL-AML data set (300 attr.; $\beta = 0\%$)



(d) Mining time of mining disjunctive EPs w.r.t min supp in D_p for ALL-AML data set (300 attr.; $\beta = 0\%$)

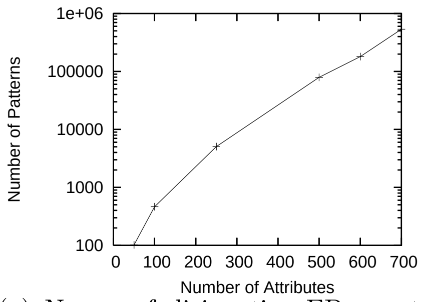


(e) Num. of disjunctive EPs w.r.t min supp in D_p for colon tumor data set (100 attr.; $\beta = 0\%$)

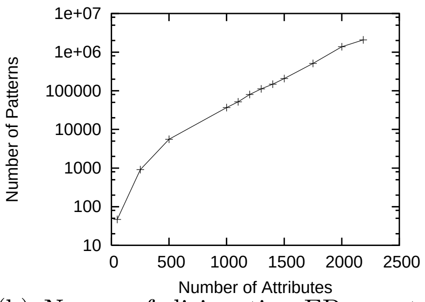


(f) Mining time of mining disjunctive EPs w.r.t min supp in D_p for colon tumor data set (100 attr.; $\beta = 0\%$)

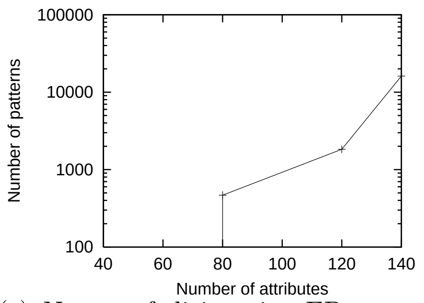
Figure 6.8: Performance results from various minimum support thresholds



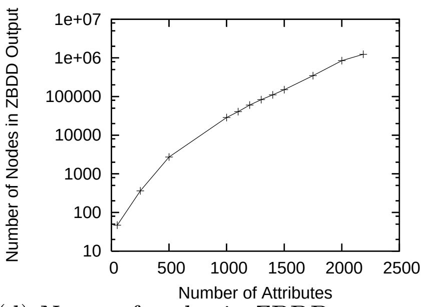
(a) Num. of disjunctive EPs w.r.t num. of. attr. for ALL-AML data set ($\alpha = 90\%$, $\beta = 10\%$)



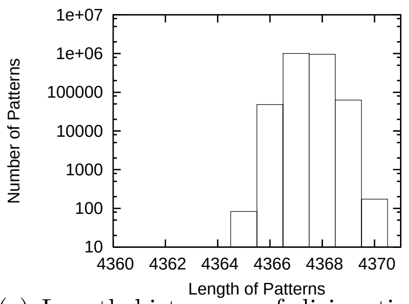
(b) Num. of disjunctive EPs w.r.t num. of. attr. for lung cancer data set ($\alpha = 90\%$, $\beta = 10\%$)



(c) Num. of disjunctive EPs w.r.t num. of. attr. for colon tumor data set ($\alpha = 90\%$, $\beta = 0$)

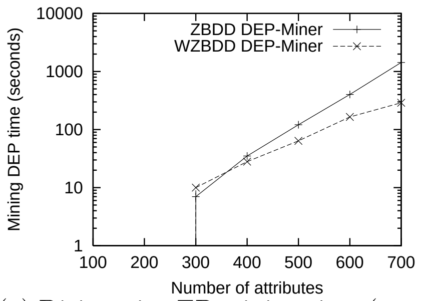


(d) Num. of nodes in ZBDD output w.r.t num. of. attr. for lung cancer data set ($\alpha = 90\%$, $\beta = 10\%$)

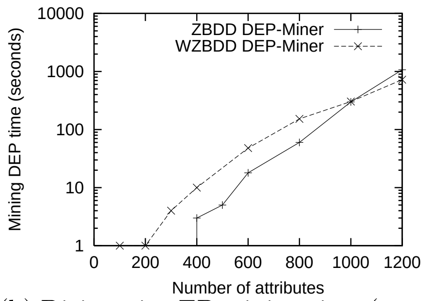


(e) Length histogram of disjunctive emerging patterns for lung cancer data set (2172 attributes; $\alpha = 90\%$, $\beta = 10\%$)

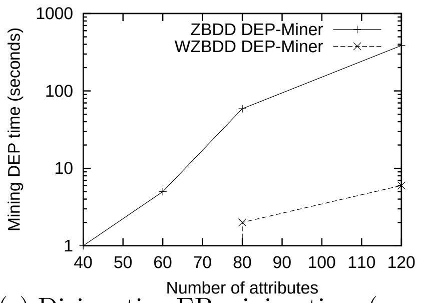
Figure 6.9: Performance results from various data dimensionalities



(a) Disjunctive EP mining time (seconds) w.r.t num. of attr. for ALL-AML data set ($\alpha = 90\%$, $\beta = 10\%$)



(b) Disjunctive EP mining time (seconds) w.r.t num. of attr. for lung cancer data set ($\alpha = 90\%$, $\beta = 10\%$)



(c) Disjunctive EP mining time (seconds) w.r.t num. of attr. for colon tumor data set ($\alpha = 90\%$, $\beta = 0\%$)

Figure 6.10: Comparison between the mining time using ZBDD, or weighted ZBDD, for mining disjunctive emerging patterns, w.r.t various data dimensionalities (i.e. number of attributes)

6.5 Discussion

In this section, we will discuss the strength and weakness of pattern tree, ZBDD, and WZBDD, for mining contrast patterns, either they are expressed as simple emerging patterns, or disjunctive emerging patterns.

6.5.1 ZBDD vs Pattern-Tree

The main difference between using ZBDD and Pattern-Tree is that the Pattern-Tree allows a dynamic variable ordering to prune the search space. This is not desirable for ZBDDs, since different variable orderings would limit node sharing between the intermediate ZBDD databases which is the main feature of the ZBDD-based algorithm.

Some evidence is shown in our experimental results, where ZBDD Miner outperforms the Tree Miner when many patterns exist, such as in circumstances with high data dimensionalities, or strict anti-monotone constraint with regards to the patterns' support in the positive class.

When the minimum support for the positive class is high, not many patterns exist. Moreover, when the data dimensionality is high, the patterns are relatively short compared to the input itemsets. In such a circumstance, the dynamic variable ordering employed by the Pattern-Tree Miner is effectively useful for pruning the search space, but it is less useful when the support threshold is low, or many long patterns exist, which is shown in our experimental results.

6.5.2 Weighted ZBDD vs ZBDD

We firstly compare between the use of Weighted ZBDD and the non-weighted ZBDD for mining emerging patterns. The weighted ZBDD is shown to be less sensitive to the anti-monotone constraint: the minimum support in the positive class. The non-weighted ZBDD, on the other hand, is more sensitive especially when the minimum support is low.

The main difference between the algorithms which are based on weighted ZBDD and non-weighted ZBDD is that Weighted ZBDD Miner performs ZBDD database projections which represent the negative dataset partitions projected by prefixes of the patterns. The overall cost of performing such database projections is influenced by the data dimensionality and the number of the patterns.

When many patterns exist, such as when the positive support is low, the benefit offered by Weighted ZBDD allows the redundancy of frequency counting in ZBDD Miner to be avoided. In the smaller More specifically, few short patterns exist when the minimum support threshold is high.

The drawback of ZBDD Miner is in its bitmap-based frequency counting whose efficiency is influenced by the size of the input dataset. The larger the input dataset, and the higher the data dimensionality, the more expensive its computation is. Its strong performance compared to the Weighted ZBDD Miner is evident in the smaller lung cancer data set, with high dimensionality and high positive support threshold. On the other hand, its weak performance is evident in the larger colon tumor data set.

When mining the disjunctive emerging patterns, the weighted ZBDD Miner is also more scalable than the non-weighted ZBDD Miner. In most cases, there exist more disjunctive patterns than the non-disjunctive patterns, hence, their search space is larger. In such a scenario, the limitation of ZBDD Miner becomes more apparent, due to the cost in performing frequency counting using its bitmap data representation. The weighted ZBDD miner, on the other hand, allows more efficient frequency counting, resulting in an overall efficient running time as shown in our experimental results.

6.6 Related Work

We have already referred to the general work in the area of ZBDD in Section 3.2. However, we are only aware of one paper [111] where ZBDD is used for pattern mining. They propose a method for finding frequent patterns. Their approach is different from ours in the sense that they explicitly store the support information by constructing multiple shared-ZBDDs which groups itemsets based on their (binary-encoded) supports. It enumerates the complete set of frequent patterns (i.e. the patterns satisfying all support constraints $\alpha > 0$) regardless of the given support threshold, making it inefficient for high values of α or for mining in high dimensionality data since millions of patterns may exist. More details about their algorithm were also presented in Section 3.3.1 in Chapter 3. On the other hand, our proposal stores the input transactions in a single ZBDD, reducing its overall memory consumption, and pushes constraints deep inside the ZBDD operations. We use a secondary structure such as bitmaps for counting support, instead of storing support information inside the ZBDD structure.

Emerging patterns were proposed in [37], and have been successfully used in constructing highly accurate classifiers [39], in particular, for predicting the likelihood of diseases such as leukemia [84] using gene expression data [86]. They are closely related to association rules with large confidence [160] and also to work on detecting group differences [17]. A recent method for mining emerging patterns with zero frequency in the negative dataset appears in [51]. This method is based on modifications to the FP-tree [70]. FP-trees have also been used as the basis for mining contrasts having other types of constraints, such as risk and odds ratio [81]. Work in [83] uses an FP-tree for storing the frequency (in multiple classes) of each itemset. Adopting the similar frequency representation in a BDD is possible, but it would affect the compactness of the structure and the chance of node sharing across the databases.

Connections between the computation of certain kinds of emerging patterns and hypergraph transversals are identified in [16]. Another related notion is version spaces [120, 73], which correspond to emerging patterns with constraints $\alpha = 100\%$ and $\beta = 0$. A disjunctive version space [142] is a disjunction of version spaces, as opposed to the disjunctive emerging patterns presented here, which are a conjunction of disjunctions on attribute values.

Work in [16] identified out a connection between the computation of emerging patterns and the computation of minimal hypergraph transversals (the complements of maximal independent sets). A number of theoretical results for minimal transversals have been developed in papers such as [23, 79, 22, 44]. There, it is shown that a number of problems involving transversals can be computed in quasi-polynomial time (measured according to combined input and output size). This work differs from ours in two principal respects: i) It focuses solely on monotone constraints, whereas the computation of dense maximal independent sets requires simultaneous consideration of both a monotone (negative) and an anti-monotone (positive) constraint, ii) The basis of their approach, the Fredman-Khachiyan algorithm, although having the best known worst case complexity for minimal hypergraph transversals, is unsuitable in practice for use on the kinds of hypergraphs that arise in data mining, often being orders of magnitude slower than bottom-up search methods, since the the volume of output tends to be very large, but the cardinality of patterns small. Work in [16] gives some indicative results in this direction.

In an ordered domain, contiguous disjunctive patterns correspond to quantitative association rules [147], having high confidence and a single item consequent. A quantitative association rule is a conjunction of intervals. Disjunctive and more expressive association rules have been studied [121], but they allow DNF (disjunction of conjunctions) rules, instead of CNF (conjunction of disjunctions) which is the kind of disjunction considered in our work. Thus, disjunctive emerging patterns can effectively be useful for finding a more complex type of quantitative association rule.

Several papers have examined the computation of empty regions or 'holes' in datasets [43, 93]. A contiguous disjunctive emerging pattern with $\beta = 0$ corresponds to a hole in D_n which satisfies the α support constraint. Other work which considers itemsets having disjunctive properties is [140], where similarities to classes of functional dependencies are demonstrated. Interestingly, connections between the discovery of functional dependencies and hypergraph transversals have been explored in other work, see e.g. [65].

A number of recent papers have examined mining of closed patterns from high dimensional datasets using row enumeration instead of column (item) enumeration [125, 136, 96]. The emphasis on closed patterns, as opposed to minimal patterns means this is not directly applicable for finding minimal contrasts. However, alternative variants of emerging patterns based on closure properties can certainly be defined e.g. see [146]. In contrast to the row enumeration work, our paper seeks to investigate the limits of column-wise mining and indeed our results showed that column-wise mining of contrasts in high dimensional datasets is feasible using ZBDDs.

6.7 Summary

In this chapter, we have developed efficient algorithms for mining contrast patterns in high dimensional data. We presented an algorithm based on the use of Zero Suppressed BDD (and its weighted variant) as a data structure. We demonstrated how mining constraints could be integrated with the standard ZBDD library routines. Our experimental results showed the technique scales well for a number of high dimensional biological datasets and allows the computation of both simple contrasts, such as emerging patterns and also more complex contrast patterns which use both disjunction and conjunction. For emerging pattern mining, we showed our method substantially improves on the tree-based technique [51]. Moreover, we are not aware of other work suitable for computing the expressive contrasts considered.

Chapter 7

Using Expressive Contrast Patterns for Classification

In previous chapter, we introduced a highly expressive class of contrasts, namely *disjunctive emerging patterns*, which allow disjunctions as well as conjunctions of attribute values. Emerging patterns [37] have been shown to be useful for building accurate classifiers [42, 39], but the use of the more expressive contrasts for classification remains an open question. This chapter investigates whether disjunctive emerging patterns are helpful for improving the accuracy of classification. We use the term *simple contrasts* to refer to non-disjunctive emerging patterns, and *expressive contrasts* to refer to disjunctive emerging patterns.

7.1 Introduction

A pattern-based classifier builds its model by finding contrast patterns from the given training data. For classifying a test instance T, it weighs the contribution of each pattern according to its occurrence in T, based on its contrast-strength in the training data. Simple contrasts, however, have several limitations which can potentially be overcome by expressive contrasts. For example, consider the following expressive contrast from an *income* data set [71].

Example 28. *Let the following combination be a contrast that differentiates male from female, being true for more than 10% males in the population but not true for any female:* $[age \in [30..39] \wedge (industry = \text{'}manufacturing\text{'} \vee \text{'}transportation\text{'})]$. *Should* *the two industries are considered individually, each of the conjunctive combinations i) [$age \in [30..39] \wedge industry =$ 'manufacturing'], and ii) [$age \in [30..39] \wedge industry =$ 'transportation'] is true for less than 10% males, and they are individually weaker contrasts.*

The issue that occurs in the above example often arises when the data is sparse, or very small, in which strong contrasts are rare. The conjunctive combinations in such a circumstance have low frequency and low contrast strength, but they may be useful for classification.

Another limitation of simple contrasts occurs in the presence of continuous attributes. Emerging patterns assume discrete data. Hence, their usefulness for classification is influenced by the data discretisation. In coarsely discretised data, the simple contrasts may be lacking class-distinguishing ability. In a finely discretised data, on the other hand, the simple contrasts may be lacking frequencies (support). We call this problem the *resolution problem*.

7.1.1 Challenges in Using Expressive Patterns for Classification

Expressive contrasts allow disjunctions (within attributes) as well as conjunctions (across attributes). Their expressiveness is potentially useful for identifying rare contrasts which may help remedy the above-mentioned situations. The disjunctive patterns address the resolution problem by merging multiple intervals through considering their disjunction. Suppose a continuous-valued attribute *age* is finely discretised into equal-length intervals of size 5. A disjunction of [$age \in [20..24) \vee age \in [25..30)$] corresponds to an interval of size 10 (i.e. $age \in [20..30)$, which is found using a coarser discretisation.

Despite their potential, using disjunctive patterns for classification can have some limitations due to two factors:

- A large number of patterns may be available to be used by the classifier
- The patterns may contain disjunctions between data items which are not strongly related, or irrelevant

Our model uses the most expressive (or maximal) disjunctive emerging patterns, i.e. expanding any of its disjunctions results in a constraint violation. Being most expressive, however, those patterns may contain irrelevant disjunctions. Suppose a combination $[(age \in [20..24] \vee age \in [40..44]) \wedge industry =$ 'manufacturing'$]$ is a maximal disjunctive emerging pattern. For example, another combination $age \in [20..24] \wedge$ $industry =$ 'manufacturing' is also a valid disjunctive pattern. It may be the case that the two combinations have the same frequency, if people who are 40-44 years old and working in the manufacturing industry do not exist in the given data samples. Such a circumstance is likely to occur when the data is sparse, or has insufficient training samples. To address this issue, our model adopts a methodology for statistically testing the significance of disjunctive patterns.

7.1.2 Purpose and Contributions

In summary, the purpose of this chapter is to investigate the benefit of using highly expressive contrasts for classification, compared to the simple contrasts. We also analyse circumstances where they are helpful, or not helpful. More specifically, we aim to answer the following questions:

- When should disjunctions be allowed in contrast patterns for building a classifier?
- Which types of contrast patterns are most suitable for various data characteristics?

Our contributions in this chapter are three-fold:

- We propose a classifier based on expressive contrasts, such as disjunctive emerging patterns [97]. As a means to reduce noise and improve classification accuracy, we use a statistical significance method based on the Fisher's Exact Test, similar to that used in [152]. Moreover, to test the significance of each condition in a pattern, we extend the testing methodology by using *negative representations* of the patterns, which are conjunctions of negated attribute values. The use of statistical tests on negative conjunctions has not been previously studied.
- We present experimental results using several real [71] and synthetic data sets, and show the accuracy comparison between our classifier, and a JEP Classifier [39, 51], which is based on emerging patterns with infinite growth rate. It shows that the disjunctive classifier is superior for sparse data, and as good as the other classifier for dense data. Moreover, our results show that the accuracy of our classifier has a low sensitivity to data discretisation.

- Based on our experimental findings, we present a series of recommendations for practitioners, answering the two questions posed earlier, about when disjunctions should be allowed in contrast patterns, and which types of contrast patterns are most suitable for classifying data with particular characteristics.

7.2 Preliminary

We follow the definition of emerging patterns and disjunctive emerging patterns which were defined in previous section. In the remainder of this chapter, we use the term *pattern* to refer to a contrast pattern, either as an emerging pattern or a disjunctive emerging pattern. Let D_p and D_n be the positive and the negative class, respectively. The *support* of a pattern refers to its support in the positive class (i.e. D_p). In relation to association rules, a pattern P is associated with the positive class. It corresponds to a highly confident association rule: $P \rightarrow D_p$ [1].

An overview of the classifiers based on emerging patterns was given in Chapter 2 (Section 2.9). Given a test instance T, if a pattern P occurs in T, P makes a *contribution* to classify T as an instance of D_p. The patterns which contain T can be found from each class, and their *contributions* aggregated. To make the final classification decision, class C is chosen as the label of T if the aggregated patterns which favor C are stronger than those which favor the other class. The classifier based on aggregating emerging patterns, namely CAEP [42], chooses class C if its total score (i.e. contrast strength of the contributing patterns) is maximum: $\mathit{class}(T) = \max_{\mathit{score}(T,C)} C$, and

$$\mathit{score}(T, C) = \sum \mathit{strength}(Q, C, \neg C) \tag{7.1}$$

where Q is a contrast pattern which contains T, i.e. $T \subseteq Q$.

7.2.1 Overview of Disjunctive Emerging Patterns

Disjunctive emerging patterns (Chapter 6) express contrasts as conjunctions of disjunctions. They can be seen as CNF boolean formulae. Disjunctions are only allowed between items within attributes, and conjunctions across attributes. Formally, a **Disjunctive Emerging Pattern (DEP)** is an itemset x s.t. x contains at least one

[1] The confidence of an association rule: $\mathit{confidence}(P \rightarrow D_p) = \frac{\mathit{support}(P, D_p) * |D_p|}{\mathit{support}(P, D) * |D|}$, where $D = D_p \bigcup D_n$

item from the domain of every attribute, and satisfies two support constraints: i) *support*$(x, D_p) \geq \alpha$, and ii) *support*$(x, D_n) \leq \beta$. Itemset x is a **maximal DEP** if there does not exist another DEP y such that $x \subset y$.

Example 29. *Given a dataset having three attribute domains* $\{a_1, a_2, a_3\}$, $\{b_1, b_2, b_3\}$, $\{c_1, c_2, c_3\}$. *Suppose* $x = \{a_1, a_2, a_4, b_1, b_4, c_1, c_2\}$ *is a disjunctive pattern. The association rule that x represents, denoted* $f(x)$, *is* $(a_1 \vee a_2 \vee a_4) \wedge (b_1 \vee b_4) \wedge (c_1 \vee c_2) \rightarrow D_p$.

The dataset projection into multi-dimensional space considers a disjunctive itemset as a subspace. Figure 7.1 shows tha subspace which corresponds to pattern x given in the previous example. Thus, the *support* of a disjunctive itemset is calculated by counting the transactions which it contains.

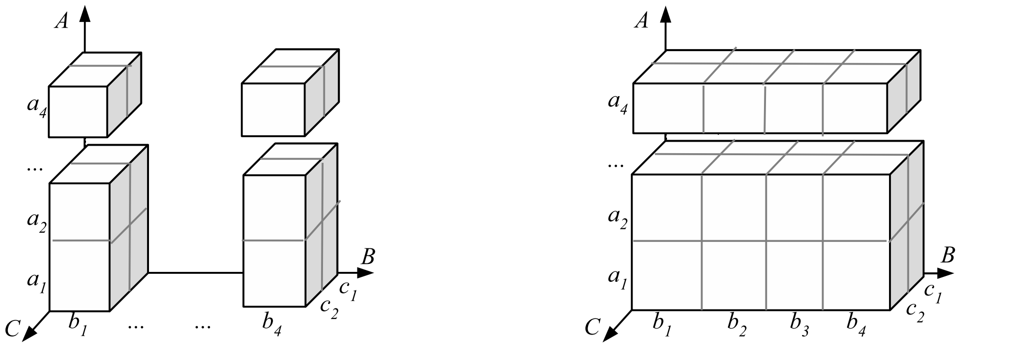
For an attribute over an ordered domain, a set of adjacent items is called a *contiguous* itemset. An itemset is g*-contiguous* if the gap between any two consecutive items is no larger than a given minimum threshold g, where $0 \leq g \leq k-2$, where k is the number of domain items for that attribute. Moreover, a **g-contiguous pattern** is a pattern which does not contain any non g-contiguous subsets in any of its dimensions. Consider Figure 7.1, x is g-contiguous for $g \geq 2$, and y is g-contiguous for $g \geq 1$.

Apart from their expressiveness, another advantage of the non-contiguousness of disjunctive patterns is that they can allow rare contrasts to be identified. However, there is a possible disadvantage. A non-contiguous disjunction geometrically splits a pattern into contiguous sub-patterns, but in real situations, these sub-patterns may be irrelevant. For this reason, we will propose a statistical technique for testing the significance of a gap, which measures whether such gap makes a significant contribution to the contrast strength of the pattern.

7.3 Statistically Significant Disjunctive Emerging Patterns

For the purpose of our study, our classifier uses the existing JEP-classifier framework [39] as a baseline, which is particularly powerful for classifying the challenging multi class, or dense, data sets. A JEP, or Jumping Emerging Pattern, is an emerging pattern that has an infinite growth rate. The JEP-classifier is based on minimal JEPs, i.e. those which are most expressive. For this reason, our classifier is based on maximal disjunctive patterns which have infinite growth rate.

Since disjunctive patterns are relatively longer (i.e. contain more items) than the simple patterns, intuitively not every item makes an equally-high contribution into the



(a) $x = \{a_1, a_2, a_4, b_1, b_4, c_1, c_2\}$ (b) $y = \{a_1, a_2, a_4, b_1, b_2, b_3, b_4, c_1, c_2\}$,

Figure 7.1: Geometric representations of disjunctive patterns x and y

contrast strength of a pattern, which may introduce noise to the classification model. To address this issue, we propose two levels of significance testing:

- **External significance**: tests whether the occurrence of a pattern is highly associated with the class
- **Internal significance**: tests whether each element in a pattern makes a significant contribution into the contrast strength of the pattern.

Work in [159] proposed a multiple test correction method to solve the *multiple tests* problem when testing the significance of multiple rules. Although this method is useful to improve the significance of the rules, we do not use it in our classifier, since the multiple tests problem is more of an issue in knowledge discovery, rather than classification (discussed in [152]).

7.3.1 Statistical Fisher Exact Test and Externally Significant Patterns

Work in [152, 159] has shown that Fisher Exact Test (FET) is useful for finding statistically significant association rules, which makes it potentially useful for finding significant contrast patterns as well. To test the significance of a pattern P, FET uses a 2x2 contingency table containing the support of P and its complemented support in each class (shown in Table 7.1). FET tests the *null-hypothesis* which says that there is no significant association between the rows and the columns of the contingency table. The test returns a probability (i.e. p-value) for whether the hypothesis should be accepted. Given a contingency table $[a, b; c, d]$, and $n = a + b + c + d$. The p-value is computed

Table 7.1: The contingency table for testing the significance of association between pattern P and class C

D	P	$\neg P$
C	$a = support(P, C)$	$b = support(\neg P, C)$
$\neg C$	$c = support(P, \neg C)$	$d = support(\neg P, \neg C)$

by:

$$p([a, b; c, d]) = \sum_{i=0}^{min(b,c)} \frac{(a+b)!(c+d)!(a+c)!(b+d)!}{n!(a+i)!(b-i)!(c-i)!(d+i)!} \tag{7.2}$$

If the p-value of a pattern is below the given *significanceLevel* (typically 0.05), we reject the null-hypothesis and say that the pattern has a significant class-association. In our model, we call such a pattern an **externally significant pattern**.

7.3.2 Internally Significant Disjunctive Emerging Patterns

As has been identified in [152, 159], the inclusion of each condition in a significant association rule should significantly contribute to the rule's associations with the classes. The testing methodology was originally fashioned for purely conjunctive rules. To adapt the method for our needs, we use the *negative representation* of a disjunctive pattern. Based on the Negative Normal Form (NNF) representation of boolean formulae, a pattern can be represented as a conjunction of negative items, and it is significant if each negative item has a significant contribution. This differs from the previous work on significant association rules which are conjunctions of positive items instead of negative items.

Example 30. *Suppose there are two attribute domains, $\{a_1, a_2, a_3, a_4\}$, $\{b_1, b_2, b_3, b_4\}$, and $x = \{a_1, a_4, b_2, b_4\}$ is a disjunctive pattern. The CNF representation of x is $f(x) = (a_1 \vee a_4) \wedge (b_2 \vee b_4)$. The NNF (Negative Normal Form) of $f(x)$ is the conjunction of the non-occurring variables, i.e. $f_N(x) = (\neg a_2 \wedge \neg a_3) \wedge (\neg b_1 \wedge \neg b_3)$.*

The significance of a negated item $\neg z$ in the previous example which occurs in $f_N(x)$ is calculated between x and its generalisation (by inverting $\neg z$ to z). Let $z = a_2$. Let y be the generalisation of x such that $y = x \cup \{a_2\}$. The contingency table for testing the significance of $\neg a_2$ in x consists of the frequency of $\neg a_2$ and a_2 in transactions that contain y. We can calculate the p-value using Equation 7.2 and the contingency table in Table 7.1, by letting $P = \neg z$, $C = D_p|y$, and $\neg C = D_n|y$, where $D_p|y$ refers to the transactions in D_p which contain y, and similarly for $D_n|y$. The p-value gives the significance of $\neg z$ in $f_N(x)$, where $x = \neg z \wedge y$. If each of the negative items a_2, a_3, b_1, and b_3 is significant in $f_N(x)$, then we say that x is an **item-significant pattern**.

Given ordered attribute domains, a disjunctive pattern can be projected into a subspace possibly with some holes in it, where a hole corresponds to a disjunction of non-contiguous items. If there are many small holes, those holes may not be worth retaining if they contain very few data instances from the positive class. On the other hand, big holes may be necessary if they contain many data instances from the negative class. To evaluate the significance of each hole in a disjunctive pattern, we can employ the item-significance test for a set of items, instead of a singular item.

Formally, we call a set of contiguous items which are not contained by a disjunctive pattern as a *gap*. A gap is a *significant gap* if it passes the internal significance test. We call the generalisation of a pattern that is obtained by inverting a gap as the *gap-filled generalisation.* A gap is maximal if it is not a subset of another gap. If each maximal gap g in a x is internally significant, then we say that x is a **gap-significant pattern**.

Example 31. *Consider pattern x in Fig. 7.1a. Given three attribute domains $\{a_1, \dots a_4\}$, $\{b_1 \dots b_4\}$, $\{c_1, \dots c_4\}$, the CNF representation of x is $(a_1 \vee a_2 \vee a_4) \wedge (b_1 \vee b_4) \wedge (c_1 \vee c_2)$. This pattern contains two holes $\{a_3\}$, $\{b_2, b_3\}$. To measure the significance of hole $\{b_2, b_3\}$ in x, we use the contingency table in Table 7.1, by letting $P = \neg(b_2 \vee b_3)$. The p-value measures the significance of excluding interval $\{b_2, b_3\}$ from x, with respect to the gap-filled generalisation y, where $y = x \cup \{b_2, b_3\}$ (shown in Figure 7.1b).*

7.3.3 Classification by Significant Disjunctive Emerging Patterns

Our classifier is built based on significant disjunctive emerging patterns. To obtain patterns with strong contrast strength, we firstly find the maximal disjunctive patterns which have an infinite growth rate. Using only those patterns, however, may overfit the training data, especially when the data is sparse.

In real situations, there may exist patterns which have a significant association with the test instance, but are not identified by the classifier due to the *infinite growth rate* constraint. To eliminate this problem, our classifier allows some constraint violation by filling-in the insignificant gaps, if both of the following conditions hold:

- The gap is not significant in the original pattern
- The resulting gap-filled pattern is externally significant

If a pattern contains an insignificant gap, and the second condition does not hold, then that pattern is removed from the classifier. Thus, all patterns which are used by the classifier are externally and internally significant. We refer to such patterns **significant disjunctive patterns**.

Algorithm 7.1 shows the procedure for filling the insignificant gaps and testing the significance of the resulting pattern. Given a disjunctive pattern x. For each maximal gap g in x, we test whether g is a significant-gap based on the gap-significance test proposed in Section 7.3.2. If it is significant, it means that the discriminating ability of x is highly dependent on the exclusion of g, and we leave the gap as it is (line 5). However, if gap g is not significant in x, then it is a candidate to be filled (line 7). Since x is a maximal pattern, i.e. none of its proper supersets satisfy the support constraints, filling-in gap g violates the support constraints. To maintain the class-discriminating ability of the gap-filled pattern, labeled as x_{filled}, we test its significance (line 10) by the external significance testing described in Section 7.3.1.

We will shortly describe our classifier, `SigDEPClassifier`, as shown in Algorithm 7.2. In the learning phase (line 1-4), it finds the significant disjunctive patterns from each class C_i, which satisfy a pre-defined minimum support α in D_p, and maximum support β in D_n. In the classification phase (line 5-11), given a test instance T, the patterns which contain T are selected to make contribution into the classification (line 7), and their discriminating powers aggregated for each class (line 8), using the scoring function *score* (Equation 7.1). Other alternative scoring functions may be used instead of the simple one that we proposed. Finally, decision is made for T based on the class which has the highest score (line 10).

7.4 Experimental Results

In this section, we evaluate the performance of our classifier described in Section 7.3.3. The aim of our experiments is to analyse the benefits of using statistically significant disjunctive emerging patterns for building an accurate and robust classifier.

7.4.1 Experimental Setup

To evaluate the sensitivity of our classifier to data discretisation, we use the following four data sets [71], which contain continuous attributes: *breast-cancer-w*, *glass*, *wine*, and *horse-colic* data sets. Their characteristics are shown in Table 7.2. We categorise

Algorithm 7.1 `fillGaps`(x)

```
Input: A disjunctive emerging pattern $x$
Output: A gap-filled significant pattern $x_{filled}$ which fills the insignificant gap in $x$
1: $G$ = the set of maximal gaps in $x$
2: $x_{filled} = x$ /* Initialise the gap-filled pattern */
3: for each hole $g$ in $G$ do
4:    if (gap $g$ is a significant gap in $x$) then
5:       $x_{filled} = x_{filled}$ /* Leave gap $g$ as it is in $x$ */
6:    else
7:       $x_{filled} = x_{filled} \cup g$ /* Fill gap $g$ in $x$ */
8:    end if
9: end for
10: if (the gap-filled pattern $x_{filled}$ is externally significant) then
11:    return $x_{filled}$ /* The gap-filled pattern is returned */
12: else
13:    return $\emptyset$ /* No pattern is returned */
14: end if
```

Algorithm 7.2 `SigDEPClassifier`(D_{train}, D_{test})

```
Input: A training dataset $D_{train}$, which contains $k$ classes: $C_1, C_2, \ldots C_k$, and a testing
    dataset $D_{test}$
Output: Assign the class label for each test case in $T$
1: for each class $C_i$ in $D_{train}$ do
2:    $\neg C_i = \bigcup_{j=1..k} C_j - C_i$
      /* Combine the other classes to form the negative data set */
3:    $S_i$ = the set of significant disjunctive patterns in $C_i$ w.r.t $\neg C_i$
4: end for
5: for each test instance $T$ in $D_{test}$ do
6:    for each class $C_i$ do
7:       $C_i^T$ = disjunctive patterns in $C_i$ which contain $T$
8:       $score[i] = \sum strength(Q, C_i, \neg C_i)$, where $Q \in C_i^T$
         /* Calculate the score for class $C_i$ by aggregating the contrast strength of the
         contributing patterns */
9:    end for
10:   $classLabel[T] = \max_{score[i]} i$
      /* Choose the class which has the highest score to be the label of the test instance
      $T$ */
11: end for
```

Table 7.2: Data characteristics

Dataset	C	class sizes
breast-cancer-w	2	458 - 241
horse-colic	2	191 - 109
wine	3	59 - 71 - 48
glass	7	70 - 76 - 17 - 1 - 13 - 9 - 29

C = number of classes

the data sets based on their sparsity/density. The data sets which contain multiple classes, namely *wine* and *glass* data sets, are considered sparser than the binary-class data sets, namely *breast-cancer-w* and *horse-colic* data sets. Moreover, the *glass* data set is considered extremely sparse, since it contains 7 classes with only a few instances in each class.

The classification's accuracies are based on 10-fold stratified cross validation. We will compare the accuracy between our classifier, labeled `CNF`-classifier, and three other classifiers:

- `strictCNF` classifier: which is based on disjunctive emerging patterns without the significance test, strictly imposing the support constraints on the patterns. We compare against this classifier to evaluate the influence of employing the significance test in our `CNF`-classifier.
- `JEP` classifier [39]: which is based on jumping emerging patterns. It has shown to be particularly powerful in large and high dimension databases where the other EP-based classifiers have weak performance.
- `C4.5` classifier [133]: one of the state-of-the-art classifiers (we obtained its implementation from its original author), which is based on a decision tree. It has shown to perform well on large data sets with balanced classes, but not so well for rare-class classification.

7.4.2 Accuracy Comparison with respect to the Contiguousness of Disjunctive Patterns

In this experiment, we study the influence of employing a gap constraint on disjunctive emerging patterns to the classification accuracy. The gap constraint restricts all dimensions of each pattern to not contain a gap which exceeds g (ordered) items. We vary the value of g between 0 to $k-2$, where k is the number of discretised intervals that we chose for each data set.

Figure 7.2a shows the accuracy of the classifiers for the *breast-cancer-w* data set. When the gaps are small, i.e. $g < 8$, `CNF` and `strictCNF` classifiers are weaker than the `JEP` classifier, whilst the `strictCNF` classifier is no weaker than `C4.5`. As g increases, i.e. larger gaps are allowed in the disjunctive patterns, the accuracies of the `CNF` and `strictCNF` classifiers increase. But when $g = 8$, the `CNF` classifier outperforms both the `JEP` and `C4.5` classifiers by achieving 98.6% accuracy.

Figures 7.2b shows the accuracy of the classifiers for the *horse-colic* data set. In this data set, both of the `CNF` and `strictCNF` classifiers are more accurate than the `JEP` classifier, which only has 63%-65% accuracy. Moreover, the `CNF` classifier is the most accurate across all values of g, with its highest accuracy being achieved when $g = 6$.

For the sparse *wine* and *glass* data sets, we chose to use a high support threshold for which the JEPs exist very rarely. Figure 7.2c shows that both the `CNF` and `strictCNF` classifiers have much higher accuracies compared to the `JEP` classifier. In this scenario, they can obtain higher accuracies than `C4.5` given certain values of g. The relative accuracy between the `CNF` classifier and `strictCNF` classifier, however, varies between different values of g. Such a relative behaviour is similarly found in the *glass* data set, as shown in Figure 7.2d, except that they do not exceed the accuracy of either the `JEP` or `C4.5` classifier in this scenario.

7.4.3 Accuracy Comparison with respect to the Discretisation Granularity

For this set of experiments, we varied the number of bins when performing equal-density or equal-length discretisations for each data set. The choice of discretisation was made in such a way so that a reasonable number of patterns exist and mining could complete under 100,000 seconds. Figure7.3 shows the classification accuracy of the different classifiers, with respect to the number of discretisation binnings.

For the *breast-cancer-w* data set, Figure 7.3a shows that the accuracies of both `CNF` and `strictCNF` classifiers are improved when more binnings are used (i.e. finer granularity), with the `CNF` classifier being relatively better. Given a fine granularity with 12 bins, the accuracy of `CNF` classifier is 12% higher than the `JEP` classifier, and 2% higher than `C4.5`.

For the *horse-colic* data set, Figure 7.3b shows that the `CNF` classifier outperforms the other classifiers when a fine granularity is used. With 7 or more bins, the `CNF` classifier can obtain 85% accuracy, whilst the `JEP` classifier only has 60% accuracy, and `C4.5` has 82%. When a coarse granularity is used, i.e. 6 or fewer bins, the accuracy of `CNF` decreases towards the `JEP` classifier. Such a behaviour shows that the significance tests employed by the `CNF` classifier helps improve the accuracy in this data set when the data is finely discretised, but not when the data is coarsely discretised.

Figure 7.3c shows that in the *wine* data set, the accuracy of `CNF` classifier as well as `strictCNF` classifier increases as the number of binnings increases. As the number of bins increases, the accuracy of the `JEP` classifier decreases, whilst the `CNF` and `strictCNF` classifiers have 100% accuracy, which outperforms `C4.5` by 2%. In this data set, the significance test used by the `CNF` classifier does not have much influence in the classification accuracy across various discretisation granularities.

Figure 7.3d shows that in the *glass* data set, the accuracy of the `CNF` classifier decreases with the increase in the number of bins. Results from this experiment show that employing the significance test is not always useful for improving classification accuracy. It is only useful when fine granularity is used, and many patterns exist. More specifically, it is useful when the data is not extremely sparse. In this data set, however, either the `JEP` or `CNF` classifier cannot achieve the accuracy obtained by `C4.5`.

7.4.4 Effects of Inter-dependency Between Multiple Attributes to the Classification Accuracy

This set of experiment investigates the classification accuracies in datasets which have strong inter-dependency between the attributes. Let k be the number of synthetic attributes, we generate random values between 1 and val_{max} for each attribute. For each transaction T, t_n is the value of the n-th attribute. We define a linear function $f(T)$ of the attributes, where $f(T) = \sum_{n=0}^{k-1}(-1)^n t_n$. The class label for T is defined by the value of $f(T)$: if $f(T) \leq c$, $class(T) = 0$ otherwise, $class(T) = 1$, where c is a small positive constant. An illustration of the data set is shown in Figure 7.4a with 20 transactions, $k = 2$, $val_{max} = 40$, where the dotted-lines show a possible data discretisation. We created several synthetic datasets with an increasing number of transactions, which makes the class-distinction more difficult given a strong attribute interdependency.

Figure 7.4b shows the accuracy of each classifier for various number of instances,

N. For each value of N, we generate 10 data sets with $k = 4$, $val_{max} = 1000$, and calculate the average of the accuracy over those 10 data sets. The contrast patterns which are used by the classifier have minimum support α of 40%. Given this scenario, the `JEP` classifier has the lowest accuracy. The accuracies of both the `CNF` classifier and `strictCNF` classifier increase for larger training data. It shows that these two classifiers have a high ability to classify data with inter-dependent attributes, which cannot be handled by the `JEP` classifier.

7.4.5 Effects of Support Threshold to the Classification Accuracy

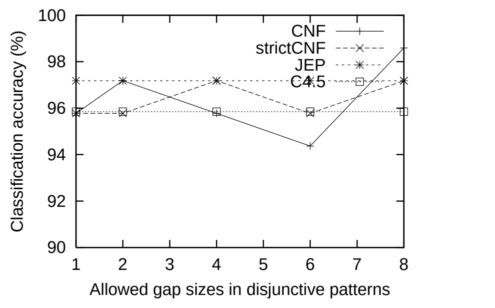
We now compare the sensitivity of the accuracy with respect to the minimum support of the contrast patterns. Figure 7.5 shows the accuracies aggregated over various support thresholds, for an increasing discretisation granularity (measured in terms of the number of bins or discretised intervals). The aggregation is computed as *(mean - 2 st.dev)*, which gives a lower bound estimate of the 95% confidence interval. In the *breast-cancer-w* data set, the `JEP` classifier has a large sensitivity with the support threshold given a fine granularity (i.e. 12 bins), whereas the accuracies of either the `CNF` classifier have lower sensitivity. In the *horse-colic* data set, the `CNF` classifier does not perform as well as `strictCNF` classifier for coarser granularities, which shows that the significance test is less useful when there is a few number of bins.

For the case of multi-class classification, in the *glass* data set, the `strictCNF` classifier without significant testing has similar performance as the `JEP` classifier, but the `CNF` classifier has a high sensitivity with respect to the discretisation granularities as well as the minimum support threshold. In the *wine* data set, which is relatively small and more balanced, the `CNF` classifier outperforms all the other classifiers, showing its suitability for multi-class classification.

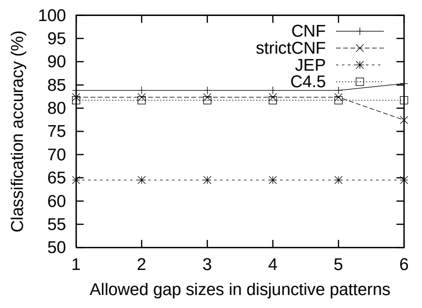
7.5 Discussion

In this section, we will discuss the behaviour of the `CNF` classifier with respect to the characteristics of the input data set, based on the performance results shown in the previous section. Under each data category, either dense binary-class or sparse multi-class data sets, we make a further categorisation based on the class sizes.

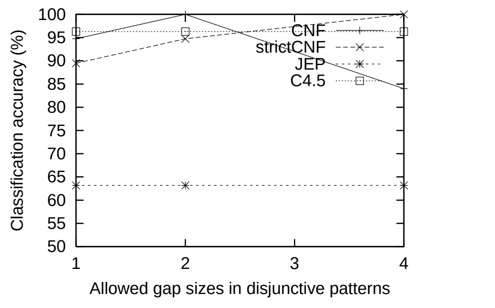
The influence of discretisation granularity: Our performance study showed circumstances where the `CNF` classifier has a high accuracy. In a finely discretised data,



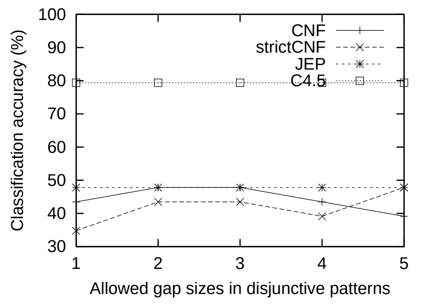
(a) breast-w (dense, $k = 10$, $\alpha = 10\%$)



(b) horse-colic (dense, $k = 8$, $\alpha = 20\%$)



(c) wine (sparse, $k = 6$, $\alpha = 80\%$)



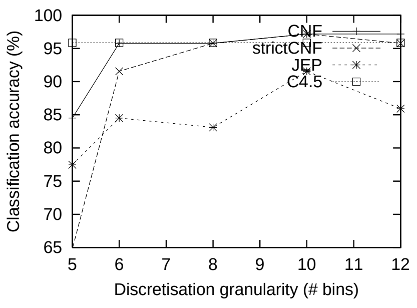
(d) glass (sparse, $k = 7$, $\alpha = 30\%$)

Figure 7.2: Comparison of classification accuracy with respect to the contiguousness of the patterns (in terms of the maximum size of each gap)

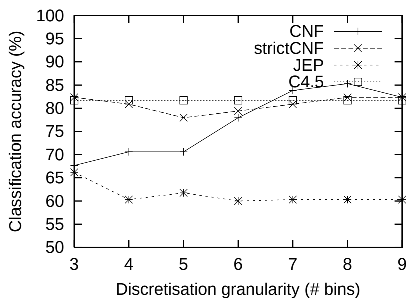
many of the patterns may not be contiguous, and more gaps may occur. Moreover, each bin comprises of a small range of values, which is likely to have low frequency unless the data is very dense. Thus, given a finely discretised data, many gaps in the patterns may be spurious and insignificant, which explain circumstances in our experiments (Section 7.4.3) where the `CNF` classifier outperforms the accuracy of `strictCNF` classifier.

The influence of gap constraint: The gap constraint seems to have stronger influence in the dense data sets compared to the sparse data sets. Applying a very strict gap constraint, i.e. setting g to be very small, may hurt accuracy. In such a circumstance, patterns which contain large gaps are split into smaller patterns. The new patterns have smaller support, thus smaller contrast strength, than the original patterns. This explains circumstances where our `CNF` classifier has a low accuracy for small values of g.

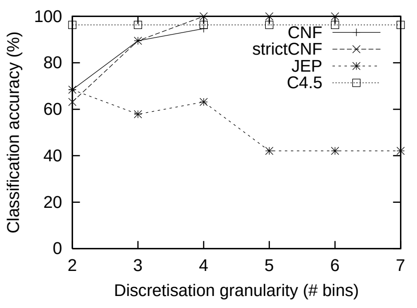
When the data is dense, the gap constraint has an important influence to the score which aggregates the strength of the mined patterns. This explains circumstances in our performance study, where the `CNF` classifier is able to improve the classification



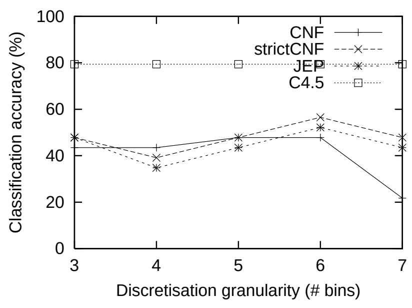
(a) breast-w (dense, $\alpha = 20\%$)



(b) horse-colic (dense, $\alpha = 20\%$)



(c) wine (sparse, $\alpha = 80\%$)



(d) glass (sparse, $\alpha = 40\%$)

Figure 7.3: Comparison of classification accuracy with respect to data discretisation's granularity

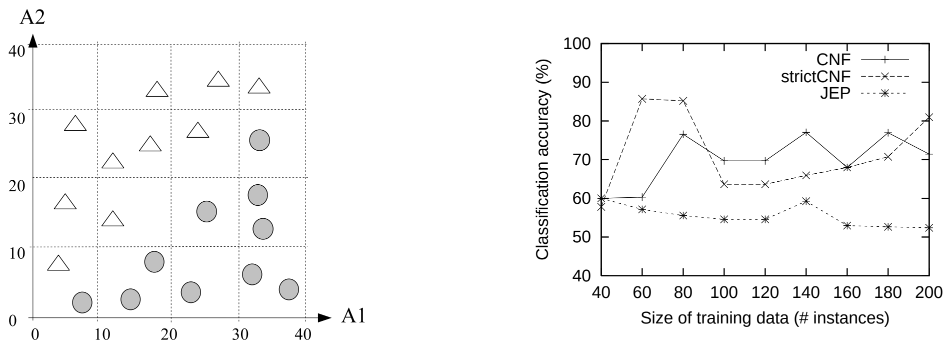
accuracy in dense data sets. It indicates that dense data sets may contain more noise than the sparse data sets, and the significance tests performed by `CNF` classifier are able to handle noise.

7.5.1 Practical Recommendations for Users

Practical recommendations for users: Answering the questions posed at the beginning of this chapter, we now present our recommendations on choosing the appropriate type of patterns for building a highly accurate classifier.

When should disjunctions be allowed in contrast patterns for building a classifier? Disjunctions should be allowed in contrast patterns for building a classifier when the data is sparse, that is when the classes are imbalanced, when the attributes have strong interdependency, or when the data is finely discretised.

Which types of contrast patterns are most suitable for various data characteristics? In a sparse data, expressive contrasts are more appropriate compared to simple contrasts. Whenever continuous-valued or ordered attributes exist, a fine granularity



(a) A diagonal data example (b) *diagonal.l1.h1000.a4.c2*

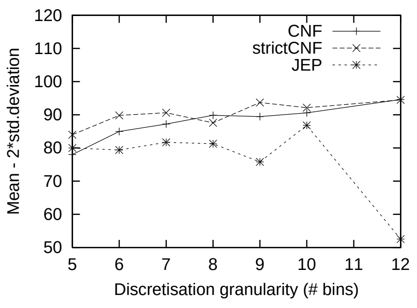
Figure 7.4: Comparison of classification accuracy in synthetic data sets with respect to data sparsity (in terms of # training instances)

discretisation should be used, e.g. 8 bins or finer, and disjunctive patterns are appropriate. The significance test should be performed on expressive patterns, except when the data is greatly imbalanced. The disjunctive patterns without significance test are suitable for a greatly imbalanced data. Simple conjunctive patterns are useful for dense and coarsely discretised data sets.

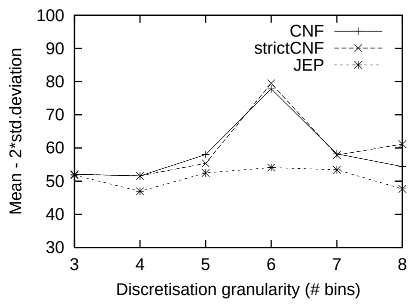
7.6 Related Work

A contrast pattern is similar to a highly confident class association rule [92]. Association rule based classifiers have been studied [92, 163, 152], but none of them use disjunctive rules. Moreover, statistical significance tests on association rules, such as in [159, 152], have only been applied to rules with a simple conjunctive antecedent (i.e. condition). Our testing method for significant disjunctive patterns has not been previously studied, and it could possibly be extended for disjunctive association rules. Other statistical measures, such as optimal rule [90] and odds ratio [83], can also be combined with the significance test presented in this chapter.

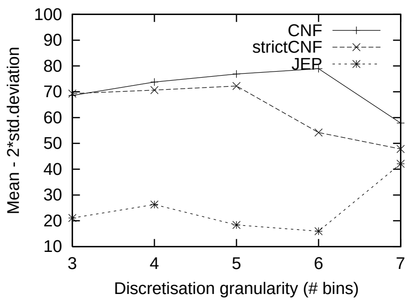
The negative representations of significant disjunctive patterns are similar to negative association rules [10, 155]. Work in [10] also studied their use for classification. However, our pattern representations limit the negation to occur only in the rule's antecedent. Disjunctive patterns are, moreover, more expressive than negative association rules. They also have similar expressions to negative jumping emerging patterns [150], but disallowing any positive item to occur in the patterns. Negative jumping emerging patterns, however, are found from a transformed database (which adds the



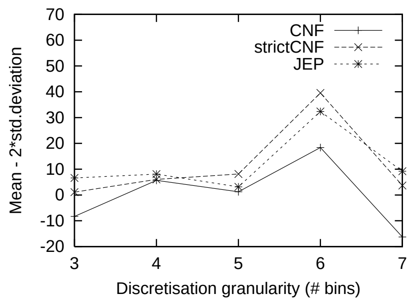
(a) breast-cancer-w (dense)



(b) horse-colic (dense)



(c) wine (sparse)



(d) glass (sparse)

Figure 7.5: Comparison of mean and st.dev of the classification accuracy (over various minimum support of the patterns) with respect to the discretisation granularities

non-occurring negative items to each transaction), which is more expensive than the technique for finding disjunctive emerging patterns [97] from the original database.

7.7 Summary

In this chapter, we have investigated the advantages and disadvantages of using expressive (in the form of CNF combinations) contrast patterns in classification, compared to the simple contrasts. We proposed a statistical testing for finding significant CNF patterns, which can also be adopted for disjunctive association rules or negative association rules. Overall, we found that expressive forms of patterns can be beneficial for classification, since expressive patterns are less sensitive, hence, more robust, with respect to the data discretisation and data sparsity.

Chapter 8

Mining Patterns and Influential Attributes That Capture Class and Group Contrast Behaviour

In Chapter 7, we studied the use of contrast patterns for classification, either the patterns are expressed as the simple emerging patterns [37], or the more expressive disjunctive emerging patterns which we proposed in Chapter 6. When multiple groups of classes are present, however, the existing types of contrast patterns cannot discover more interesting knowledge, such as the *contrasts of contrasts* which are second-order properties. Such information is useful for discovering how differentiating factors can vary across groups of classes, and discovering both the class and group contrast behaviour. Another problem is the overwhelming number of patterns which often exist. For instance, in the census data set [71], millions of (first-order) contrast patterns that differentiate males from females can be discovered, based on only the first ten attributes in the data set. What is needed, is the ability to summarise the meaning of such patterns in a highly compact way. In this chapter, we address the following two research questions: i) '*how do we discover and mine second-order differences?*', ii) '*how do we identify to the user those attributes which have the most impact with respect to a collection of second order-differences?*'.

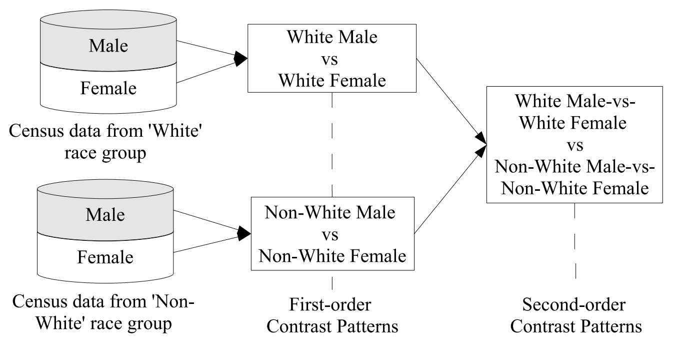

Male
Female
Census data from 'White' race group
Male
Female
Census data from 'Non-White' race group
White Male vs White Female
Non-White Male vs Non-White Female
First-order Contrast Patterns
White Male-vs-White Female vs Non-White Male-vs-Non-White Female
Second-order Contrast Patterns

Figure 8.1: First order differences within groups and second order differences across groups

8.1 Introduction

Second-order differences are meaningful in a number of interesting situations. The following are two examples which motivate our research:

Example 32. *In the census data set [71], one might wish to ask* 'what are the differences between males and females, which are characteristic for one race group, but less characteristic for another race group?' *The first-order contrast mining discovers contrasts between males and females, and the second-order contrast mining discovers how those contrasts differ between two race groups.*

Example 33. *In the domain of plant physiology [137], the biologists would like to discover* 'how does the response to a given treatment differ between the tip and base of a leaf?' *First-order contrast mining discovers treatment contrasts, comparing leaf samples which are given a treatment, against leaf samples which are not given the treatment. Second-order differentiation then compares treatment contrasts with respect to the tip of the leaf, against treatment contrasts with respect to the base of the leaf.*

To answer our research questions, we propose the following two solutions:

- A method that discovers the second-order differences between contrasts for one group of classes, compared to contrasts for some other group of classes. This problem differs from standard contrast mining scenarios, since one needs to be able to compare across groups of classes, as well as between classes.
- A technique for ranking attributes, based on their degree of influence within a collection of second order differences. Such a ranking is far easier to interpret by

a user, compared to returning millions of patterns. It aims to identify the key underlying factors responsible for change across groups.

Firstly, we introduce a class of second-order contrast patterns that we will call the **Group Discriminative Contrast (GDC) patterns**. They correspond to patterns of contrast that strongly differentiate the classes in one group, but whose discriminative power (i.e. ability to differentiate the classes) in the other group is weaker. To explain further, consider an example of a second-order contrast for the census data set [71]. When comparing the differences between male and female across two race groups, i.e. '*White*' and '*Non-White*', some patterns are able to strongly discriminate males and females if the individual belongs to the white race, but not if the individual belongs to the other race group. Figure 8.1 provides a conceptual diagram explaining the relationship between the first-order and second-order contrasts.

Example 34. *In the* census *dataset, 1.5% males and 0.4% females in the 'White' population satisfy the pattern '*older than 60 years and worked in a durable manufacturing industry*'. This shows that the rule consisting of age and the industry can significantly contrast males from females in the 'White' race group, since there are 4 times more males than females for which the rule is true. However, this pattern does not match any individual in the 'Non-white' population and hence it is not a contrast for that group. We say this pattern is a* **group discriminative contrast** *pattern, since it is a class contrast (between male and female) for one group ('White'), but it is not a class contrast for the other group ('Non-white').*

Secondly, we propose a technique for finding attributes which represent the underlying factors behind second-order contrast behaviour. In particular, we identify influential attributes, whose values can be used to find partitions of the original groups, such that these partitions show significant differences in contrast behaviour across the groups. Our work is motivated by the work in [88] which shows that variation in values for certain attributes may increase/decrease the discriminative ability of some contrast patterns. How to assess the degree to which an attribute is responsible in the discriminative ability of contrast patterns has so far been an open question. The number of contrast patterns is usually exponential in the number of attributes, whereas the number of influential attributes is smaller than the number of attributes.

Example 35. *Recall the previous example. Suppose the 'working industry' of the individual is not included in the pattern. In the 'White' population group, 11.5% males and 15.2% females are 60 years old, or older, and in the 'Non-White' population group, 6%*

males and 10% females belong to that age group. This shows that considering the age by itself does not capture a strong contrast in either race group. Moreover, the industry specification attribute has some degree of group discriminative contrast influence, since when combined with age information, it helps the differentiation between males and females in the 'White' group, but does not help the differentiation in the 'Non-White' group. Furthermore, if the industry attribute had a similar effect when combined with many different patterns, we would rank it highly in terms of overall attribute influence.

8.1.1 Challenges

A major challenge in our research is that it is not obvious how one can develop a concept of second order contrast that is simple, intuitive, and useful in practice. Addressing this question is a key aim of the paper.

On the mining side, since we are considering multiple groups of classes, efficient data representation of those classes is a critical factor for efficient mining. To address this issue, we propose to use the Weighted Zero-suppressed Binary Decision Diagrams (WZBDDs) as data structure. In previous chapters of this thesis, WZBDDs have shown their usefulness for mining the other, simpler type, of patterns.

Furthermore, since we are not only discovering the patterns of second order contrast, but also the influence of each attribute in those patterns, our mining task requires repeated and expensive exploration of the pattern space for each possible attribute. To achieve efficiency, thus, it is important to be able to push multiple constraints deep into the mining routine.

8.1.2 An Overview of Feature Ranking Techniques

In regard to ranking how influential an attribute is, existing feature ranking techniques such as entropy or statistical measures, purely focus on the ability of a single attribute to determine a class label. They do not rank an attribute based on consideration of its participation in multi-variate behaviour, or on its ability to find subcategories that exhibit interesting contrast behaviour. This can be very limiting and may result in important attributes being overlooked. e.g. Work in [89] has shown that attributes which are ranked low according to entropy, may still be influential with respect to a set of contrast patterns. Our technique can uncover such attributes, since the influence of an attribute is measured with respect to its behaviour and participation within combinations of contrast patterns.

8.1.3 Contributions

This chapter makes the following important contributions:

- We address two levels of contrasts: first order contrasts between classes within a group, and second order contrasts between the first order contrasts across groups Existing work in contrast mining [37, 168, 17, 38, 160] has only addressed the problem of finding first order contrasts.

  We introduce a formal definition for a novel type of contrast pattern, the *group discriminative contrast* pattern, that differentiates the classes within a group, and at the same time, discriminates between the groups. Furthermore, we introduce a new attribute ranking method that measures the influence of an attribute with respect to its discriminative power for second-order contrast patterns, termed the *group discriminative contrast Influence.*

- We propose a mining technique which can efficiently explore the pattern space and mine the set of second order contrasts, as well as rank the degree of influence for each attribute within this set. Our algorithm is based on the use of Weighted Zero-suppressed Binary Decision Diagrams (WZBDDs) and relies on a novel method for embedding group discriminative constraints within a prefix enumeration style framework. One of the advantages of using WZBDDs is their ability to compactly represent numerous databases, which may be beneficial in second order contrast mining, since multiple groups of classes are considered. We adopt the WZBDD mining framework which we introduced in Section 4, which had been extended for mining contrasts in Section 6.

- We experimentally evaluate our technique on real datasets, and compare our attribute influential scoring method against other classic feature ranking methods, such as entropy and correlational techniques. Our experiments demonstrate the efficiency of our mining technique and also show that our approach is able to discover some intuitively meaningful attributes, representing underlying influential factors that would be difficult or impossible to isolate using standard techniques.

8.2 Preliminaries

Assume we have a data set D defined upon a set of k attributes. For every attribute A_i, $i \in \{1, 2..k\}$, the domain of its values (or items) is denoted by $dom(A_i)$. Let I be the aggregate of the domains items across all the attributes, i.e. $I = \bigcup_{i=1}^{k} dom(A_i)$. An *itemset* is a subset of I. Let p and q be two itemsets. We say p *contains* q if p is a superset of q, i.e. $p \supseteq q$. We require that an itemset can contain at most one item from the domain of any given attribute.

The data set D can be projected to a multi-dimensional space, where each attribute corresponds to a dimension in this space, and an itemset corresponds to a subspace. The *projection of p on dimension A*, denoted p_A, is the item in itemset p which belongs to the domain of attribute A, i.e. $p_A = p \cap dom(A)$. If $p_A \neq \{\}$, then p is called an *A-dependent itemset*, or p *depends* on the value of attribute A. Given an A-dependent itemset p, q is the *A-generalization* of p if q contains all items in p except the item which belongs to the domain of attribute A, i.e. $q = p \setminus p_A$.

Example 36. *Let $p_1 = \{x_0, y_1, z_1\}$ be an itemset that depends on 3 attributes, where* $\mathrm{dom}(A_1) = \{x_0, x_1\}$*,* $\mathrm{dom}(A_2) = \{y_0, y_1\}$*, and* $\mathrm{dom}(A_3) = \{z_0, z_1\}$*. The projection of p_1 in dimension A_3 is $\{z_1\}$, and its A_3-generalization is $\{x_0, y_1\}$.*

A *dataset* is a collection of transactions, where each transaction is an itemset. The *support* of an itemset p in dataset D, i.e. $support(p, D)$, is the fraction of the transactions in D which contain p ($0 \leq support(p, D) \leq 1$). The support function is monotonic, that is, for all itemset q such that $p \supseteq q$, $support(p, D) \leq support(q, D)$.

In the context of first-order contrast mining, a data set contains a positive class, namely D_p and a negative class namely D_n. The *growth rate* of an itemset p, denoted $\mathrm{gr}(p)$, is the ratio between its support in D_p and its support in D_n, i.e. $\mathrm{gr}(p) = \frac{support(p,D_p)}{support(p,D_n)}$. For all itemsets q such that $p \supseteq q$, if $support(p, D_p) = support(q, D_p)$, then $gr(p) \geq gr(q)$, and if $support(p, D_n) = support(q, D_n)$, then $gr(p) \leq gr(q)$. We follow the definition of emerging patterns which has been used through out this thesis. Given α and β threshold values, where $0 \leq \alpha \leq 1$ and $0 \leq \beta \leq 1$, an itemset p is an **emerging pattern (EP) [37]** if $support(p, D_p) \geq \alpha$ and $support(p, D_n) \leq \beta$.

8.3 Group Discriminative Contrast

In this section, we define the second-order contrast characteristics between two groups of classes in terms of **Group Discriminative Contrast (GDC)** patterns and **Group Discriminative Contrast Influential (GDC Influential)** attributes, whose definitions generalise previous work on the simple type of contrasts, emerging patterns (EPs) [37].

Considering the data in a multi-dimensional space, an EP between the positive and the negative class in a particular group corresponds to a subspace that contains at least α positive instances and no more than β negative instances from that group. Such a subspace may have different contrasting ability between the classes in another group though. Hence, before introducing our second-order contrast definitions, we firstly introduce a formula for measuring the contrast strength of a pattern in a particular group, using a function similar to one in [42] as follows.

Definition 22. *Let G_1 and G_2 be two groups of classes. Each group G, $G \in \{G_1, G_2\}$, contains a positive class and a negative class. Given an itemset p and a group G, we refer to the positive and the negative class in G, as D_p and D_n, respectively. Let* $\mathrm{support}_G(p, C)$ *be the support of p in class C in group G, and* $\mathrm{gr}_G(p)$ *be the growth rate of p in group G.*

The **contrast intensity** *of p in group G, denoted* $\mathbf{CI}_G(p, D_p, D_n)$*, is the discriminative power between the positive instances and the negative instances from group G which are contained in subspace p. We define $CI_G(p, D_p, D_n)$ as a function of the support and the growth rate of p:*

$$\mathrm{CI}_G(p, D_p, D_n) = \mathit{support}_G(p, D_p) * \frac{gr_G(p)}{1 + gr_G(p)}$$

Let p and q be two itemsets, such that p contains q, i.e. $p \supseteq q$. In a given group G, the following monotonic properties hold between their contrast intensities:

- if $\mathit{support}_G(p, D_n) = \mathit{support}_G(q, D_n)$, then $\mathrm{CI}_G(p, D_p, D_n) \leq \mathrm{CI}_G(q, D_p, D_n)$
- if $\mathit{support}_G(p, D_p) = \mathit{support}_G(q, D_p)$, then $\mathrm{CI}_G(p, D_p, D_n) \geq \mathrm{CI}_G(q, D_p, D_n)$
- if $\mathit{support}_G(q, D_p) = 0$, then $\mathrm{CI}_G(q, D_p, D_n) = 0$ and $\mathrm{CI}_G(q, D_p, D_n) = \mathrm{CI}_G(p, D_p, D_n)$

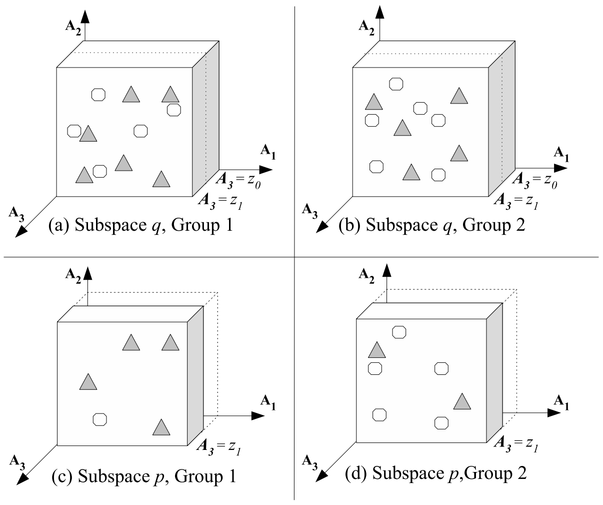


Figure 8.2: Subspace q is the generalization of subspace p in dimension A_3, where $p_{A_3} = z_1$. A triangle represents a positive instance, a circle represents a negative instance, in the specific group

8.3.1 Group Discriminative Contrast Patterns

In this sub-section, we will formally define **Group Discriminative Contrast (GDC) Patterns**, which correspond to subspaces that have strong contrast intensity between classes in one group, but they have relatively weaker contrast intensity in the other group. Firstly though, we define the following measurement for measuring how much stronger a subspace is for differentiating the classes in one group, compared to the other group. We refer to the first group as the *primary group*, and the latter as the *secondary group*.

Definition 23. *Let G_1 and G_2 be two groups of classes, where G_1 is the primary group and G_2 is the secondary group. Let D_{p_i} and D_{n_i} be the positive and the negative class in group G_i, respectively. The* **group-discriminating power** *of a pattern p, denoted* **gCIDiff**(p, G_1, G_2)*, is the difference between the contrast intensity of p in group G_1 and its contrast intensity in group G_2. It is defined as:*

$$\text{gCIDiff}(p, G_1, G_2) = \text{CI}_{G_1}(p, D_{p_i}, D_{n_i}) - \text{CI}_{G_2}(p, D_{p_i}, D_{n_i})$$

Example 37. *Figure 8.2 (a) shows a subspace q in the primary group G_1, which contains 6 positive instances and 5 negative instances. Suppose there are 10 positive and 10 negative instances in each group. Hence, we can calculate* $\text{CI}_{G_1}(q) = 0.6 * \frac{0.6}{0.6+0.5} =$

0.33. *Figure 8.2 (b) shows the same subspace in the secondary group, G_2, which contains 5 positive and 6 negative instances.* $\mathrm{CI}_{G_2}(q) = 0.23$. *The group discriminating power of q, i.e.* gCIDiff, *is 0.10, which shows that the contrast intensity of q between the positive and the negative class in G_1 is larger by 0.10 from its contrast intensity in G_2.*

An itemset with a positive group discriminating power corresponds to a subspace in which the contrast between the positive and the negative class in the primary group is stronger than the class-contrast in the secondary group. If the difference of its contrast strength exceeds a given threshold, then we call that itemset a group discriminative contrast pattern, which is formally defined as follows.

Definition 24. *Let p be a subspace that corresponds to an emerging pattern in group G_1. Given a positive minimum threshold, δ_{gdc}, p is a* **Group Discriminative Contrast (GDC) pattern** *with respect to the primary group G_1, if its group discriminating power is no less than δ_{gdc}, i.e.* $\mathrm{gCIDiff}(p, G_1, G_2) \geq \delta_{\mathrm{gdc}}$

A GDC pattern can be seen as a conditional contrast pattern, since it has a strong contrast strength given data instances from the primary group relative to the secondary group. The following properties hold between GDC patterns and the first-order contrast patterns:

- Not all of the first-order contrast patterns for the primary group are GDC patterns. Only those patterns whose contrast strength for the secondary group is lower by δ_{gdc} or more, are GDC patterns.
- A GDC pattern may be a relatively strong first-order contrast for the secondary group, but its contrast strength for the primary group is higher by δ_{gdc} or more.

8.3.2 Group Discriminative Contrast Influential Attributes

We now introduce our method for measuring responsibility (or influence) of an attribute in the set of group discriminative contrast (GDC) patterns. To give an analogy, a subspace can be seen as a window which captures the contrast intensity between the classes in each group based on the instances which are contained in that subspace. The attributes whose values are specified in the pattern correspond to the dimensions of the frame of that window. As one dimension is removed from, or added to a window, i.e. the value of an attribute is generalized or specified, its contrast intensity may

Table 8.1: Measurement categorisation according to input parameters and group applicability

Input parameter(s)	Group measurement	Between-groups measurement
A pattern	Contrast intensity (CI)	Group discriminating power (gCIDiff)
An attribute and a pattern	Contrast influence (CIGain)	Group discriminating influence (gCIDiffGain); Group discriminating influence ratio (gCIDiffGainR)
An attribute and a set of patterns		GDC influence (globalInfluence)

change due to the increase or decrease in the relative number of positive and negative instances in the new window. Such an increase of contrast intensity can thus be used for measuring the responsibility (influence) of an attribute in a particular pattern, which may vary between different patterns and different groups. Moreover, an attribute has some influence in a pattern only if one of its domain values is contained in the pattern, i.e. the pattern depends on that attribute.

Hence, we formulate the following requirements for defining the scoring function that measures the group discriminative contrast influence of an attribute:

- Aggregates the attribute's influence across all GDC patterns
- For each pattern, measures the attribute's contrast influence in the primary group
- For each pattern, measures the attribute's difference of contrast influence between the groups

We refer to the influence of an attribute in a GDC pattern as its **local influence**, and the group-discriminative contrast influence of an attribute, or the influence of an attribute across all the GDC patterns, as its **global influence**. In the remainder of this section, we use the general term *pattern* for referring to a GDC pattern, unless stated otherwise.

Definition 25. *Given an attribute A, and a pattern p such that p is A-dependent, the* **local influence** *of A in p, denoted* localInfluence*(p, A, G_1, G_2), measures the attribute's group discriminative contrast (GDC) influence locally in subspace p. Given the set of all A-dependent GDC patterns, S_A, the* **global influence** *of A,* globalInfluence*(S_A, A, G_1, G_2), aggregates the GDC influence of A across all patterns in S_A:*

$$\text{globalInfluence}(S_A, A, G_1, G_2) = \sum_{p \in S_A} \text{localInfluence}(p, A, G_1, G_2)$$

If $\text{localInfluence}(p, A, G_1, G_2) < 0$, we say that attribute A has a negative influence in p, as it shows that the inclusion of attribute A weakens the group discriminating power of p. If $\text{globalInfluence}(S_A, A, G_1, G_2) > 0$, then A is a **Group Discriminating Contrast Influential Attribute**, or **GDC Influential Attribute** for short, which means that A-dependent patterns exist, and the inclusion of dimension A strengthens the overall group discriminating power of those patterns.

An attribute's local contrast influence in a group: We now describe the measurement of the local influence of an attribute in a subspace, based on the following definition which measures the gain in the contrast intensity of the subspace, as a result of including that attribute in its dimensions.

Definition 26. *Given group G. Let D_p and D_n be the positive and the negative class in G. Let A be an attribute, p be an A-dependent pattern, and q be its A-generalization. The* **local contrast influence** *of A in p, denoted $\text{CIGain}_G(p, A, D_p, D_n)$, is the contrast intensity which is gained from the A-generalization of p:*

$$\text{CIGain}_G(p, A, D_p, D_n) = \text{CI}_G(p, D_p, D_n) - \text{CI}_G(q, D_p, D_n)$$

In the given group, a positive (resp. negative) CIGain of attribute A in subspace p shows that specifying the value of attribute A in p strengthens (resp. weakens) the class-discriminating ability of subspace p. This can be used for measuring the contrast influence of an attribute in the primary group (satisfying requirement 2 of the scoring function). However, it is a group measurement, which does not tell us about the difference in influence of the attribute with respect to the secondary group (requirement 3 of the attribute's influence scoring function).

An attribute's local contrast influence difference between groups: The following formula measures the relative local influence of attribute A, in terms of how much group-discriminating power is gained as a result of specifying the value of attribute A in a pattern.

Definition 27. *Let G_1 be the primary group, G_2 be the secondary group. Let p be an A-dependent pattern, and q be the A-generalization of p. The* **group-discriminating**

influence *of attribute A locally in p, denoted* $\text{gCIDiffGain}(p, A, G_1, G_2)$*, is the gain in the group discriminating power of p with respect to its A-generalization.*

$$\text{gCIDiffGain}(p, A, G_1, G_2) = \text{gCIDiff}(p, G_1, G_2) - \text{gCIDiff}(q, G_1, G_2)$$

Example 38. *Recall the subspace examples in Figure 8.2. In group G_1,* $\text{CI}_{G_1}(p, D_{p_1}, D_{n_1}) = 0.42$ *and* $\text{CI}_{G_1}(q, D_{p_1}, D_{n_1}) = 0.33$*. Thus, attribute A_3 has a positive contrast influence of 0.09 in p. In group G_2, attribute A_3 has a negative contrast influence of -0.17. Thus, the group-discriminating influence of A_3 is 0.50, i.e.* $\text{gCIDiffGain}(p, A_3, G_1, G_2) = 0.33 - (-0.17) = 0.50$*, which shows that the inclusion of attribute A_3 in p increases the between-groups difference of its ability to capture contrast between the classes.*

Furthermore, $\text{gCIDiffGain}(p, A, G_1, G_2)$ also measures how much larger is the contrast influence of attribute A in the primary group than its contrast influence in the secondary group, locally in subspace p. Re-writing gCIDiffGain() in terms of the contrast intensities of the subspaces, we have:

$$\begin{aligned}\text{gCIDiffGain}(p, A, G_1, G_2) &= \text{CI}_{G_1}(p) - \text{CI}_{G_2}(p) - \text{CI}_{G_1}(p \setminus A) + \text{CI}_{G_2}(p \setminus A)\\ &= \text{CIGain}_{G_1}(p, A) - \text{CIGain}_{G_2}(p, A)\end{aligned}$$

Note: $\text{CI}_{G_i}(p)$ refers to $\text{CI}_{G_i}(p, D_{p_i}, D_{n_i})$, $\text{CIGain}_{G_i}(p, A)$ refers to $\text{CIGain}_{G_i}(p, A, D_{p_i}, D_{n_i})$, where $i \in \{1, 2\}$.

Scoring function formulation: Let A_1 and A_2 be two attributes. If A_1 has a larger (resp. smaller) group discriminating influence than A_2, locally in a given pattern, the contrast influence of A_1 in the primary group is not necessarily larger (resp. smaller) than A_2. Thus, to satisfy both requirement 2 and requirement 3 of the scoring function, we further define the **group-discriminating influence ratio**, denoted gCIDiffGainR, that measures the relative between-groups difference of the influence of attribute A with respect to its influence in the primary group, locally in pattern p:

$$\text{gCIDiffGainR}(p, A, G_1, G_2) = \frac{|\text{gCIDiffGain}(p, A, G_1, G_2)|}{\text{CIGain}_{G_1}(p, A)}$$

The absolute value of the group discriminating influence is used in gCIDiffGainR() to preserve the positive/negative sign of the attribute's influence in the primary group. Using this measurement, attribute A_1 is more influential than attribute A_2 if the between-groups difference of contrast influence of A_1 is larger than A_2, relative to their respective contrast influence in the primary group.

Finally, we can re-write the global group discriminative contrast (GDC) influence, given an attribute A, and a set S_A which contains A-dependent GDC patterns, as:

$$\text{globalInfluence}(S_A, A, G_1, G_2) = \sum_{p \in S_A} \text{gCIDiffGainR}(p, A, G_1, G_2)$$

A positive global influence indicates that an attribute has helped strengthening the overall group discriminating power of the GDC patterns. Hence, such an attribute is a key factor in the contrast behaviour of those patterns. To find a **ranking of GDC influential attributes** we sort the attributes so that the attribute with the largest score of global GDC influence is the most-influential attribute. Some of the GDC patterns used for measuring the global influence may correspond to overlapping subspaces. Overlaps cannot be straightforwardly eliminated, since all GDC patterns may potentially affect the contrast intensity of the subspace, as well as the influence of an attribute in that subspace. It is worth noting that overlaps are not necessarily problematic though, since classifiers based on emerging patterns allow overlaps, but have still proven extremely successful (e.g. [42]). More sophisticated techniques for handling overlaps are beyond the scope of our research.

Table 8.1 shows the characteristics of each measurement defined in this section. The contrast intensity and group discriminating power depend on only a single pattern, the contrast influence and the group discriminating influence depend on a pattern and an attribute. In terms of the group-dependency, the contrast intensity and contrast influence are within-group measurements as they depend on a single group, whilst the group discriminating power and the group discriminating influence are between-groups measurements.

8.4 The Algorithm of `mineGDC` for Mining Group Discriminative Contrasts

This section introduces our algorithm, called `mineGDC`, which finds group-discriminative contrast patterns and their influential attributes. Before we describe our mining algorithm in detail, let us consider a naive algorithm that consists of three independent steps:

1. Find all of the emerging patterns (EPs) in the primary group
2. Apply the GDC constraint on those patterns via postprocessing, i.e. calculate their contrast intensities
3. For each attribute, find the dependent patterns and calculate the attribute's influence in those patterns

The naive mining approach can suffer from significant redundancy, because not all of the EPs satisfy the GDC constraint, and many patterns depend on several attributes. Our technique integrates those three steps and finds the GDC patterns while simultaneously calculating the influence of each attribute.

8.4.1 Mining Challenges

Our mining task is challenging due to three reasons. Firstly, it explores both the pattern space (for finding the GDC patterns) and the feature space (for finding the GDC influential attributes). Since the feature space has $O(n)$ search space, where n is the number of features, and the pattern space has $O(2^n)$ search space, performing the search in both spaces can be space and computationally expensive. None of the existing pattern mining or feature selection techniques deal with both search spaces simultaneously, which is a noteworthy feature of our algorithm.

Secondly, mining patterns with the GDC constraint is challenging because it depends on relative measurements of contrast intensity differences and ratio across groups. Such a constraint cannot be easily handled using the existing contrast pattern mining techniques, such as in [37, 168, 17, 38, 97], as they can only handle one positive and one negative class.

Thirdly, the scoring function for measuring attribute influence is expensive to compute, since it depends on multiple factors: the difference of contrast intensities between each pattern and its generalisation, and the ratio of those differences across groups We address those challenges by using a compact and efficient database representation, namely the Weighted Zero-suppressed Binary Decision Diagram (WZBDD), which is a directed acyclic graph (DAG) data structure and has previously been studied for efficient frequent pattern mining (Section 4), and for first-order contrast mining (Section 6). WZBDDs are useful since they allow compact representation and efficient manipulation of the multiple classes which are considered in second-order contrast mining.

8.4.2 Overview of the `mineGDC` Algorithm

To give a general overview, our mining framework follows a prefix growth mechanism which is typically used in the classical mining framework for finding frequent patterns [70]. It recursively grows prefixes of the patterns and projects conditional databases which contain subsets of the database which are relevant to each prefix, allowing efficient support calculation. The classical *infrequent prefix* pruning strategy for finding EPs prunes a prefix (and its supersets) if its support in the positive class is less than the minimum threshold.

To adopt the prefix growth approach for our problem, our technique projects secondary databases for all of the four classes (i.e. two classes from both groups). Moreover, to efficiently perform the GDC influence calculation for all attributes, the global influence score is computed incrementally. As soon as a GDC pattern is found through out the mining routine, the local influence of each attribute in that pattern is calculated. Not all attributes are needed to be considered for each pattern though, since the attributes whose domain values do not occur in a pattern have zero influence.

Based on the monotonicity of contrast intensity, if the support of an itemset in the positive class is 0, then its contrast intensity and its supersets' are also 0. Therefore, when performing the conditional database projections, we order the classes so that the negative class (from each group) is projected only if the conditional positive class is not empty.

8.4.3 Mining Second-Order Contrasts Using Weighted Zero-suppressed Binary Decision Diagrams

We will shortly describe our mining algorithm, `mineGDC`, as shown in Algorithm 8.1. The input databases correspond to the classes from both groups, which are represented as WZBDDs. The positive class in the primary group serves as the *pattern generator*, since prefixes of the patterns are prefixes of the itemsets in this class. Prefixes are recursively grown using the item in the top node of the pattern generator.

Mining begins by calling `mineGDC`$(prefix, D_{gen}, [D_{n_1}, D_{p_2}, D_{n_2}])$, with an empty itemset $prefix$ as the prefix itemset which is to be grown. Let x be the top-item in D_{gen} which belongs to the domain values of attribute A (line 1). Conditional database projections are performed for each input database, based on the Weighted Zero Suppressed Binary Decision Diagram (WZBDD) routines defined in Section 4 (line 3). The reduced databases are also computed to remove item x (line 4). Those reduced databases will be used to grow the current prefix with the remaining items.

Patterns which contain x are found from the conditional databases (line 6), all of which are A-dependent. Thus, the influence of attribute A can be updated immediately after all patterns that contain x have been found (line 7). This intermediate computation allows the global influence of all attributes to be calculated as the algorithm returns. Detailed explanation about the influence calculation will be given later. When all patterns that contain the top-item have been found, other prefixes are grown from the reduced databases (line 8). The output node contains the GDC patterns found from both the x-conditional databases and from the reduced databases (line 9).

The recursion terminates when the longest prefix for a particular candidate pattern has been found (line 12), or when the pattern generator is empty (line 17). When D_{gen} is an empty itemset, the prefix itemset $prefix$ is a candidate pattern, and its GDC-constraint is checked (line 13) based on its support in each class. The support of $prefix$ in a particular class is represented as the weight of the WZBDD representation of its conditional database. The output value of 1, i.e. sink-1 node (line 14), returns $prefix$ as a GDC pattern, which incrementally builds up the final output WZBDD. On the other hand, the output value of 0, i.e. sink-0 node (line 15,17), discards $prefix$ from the final output.

Updating an attribute's influence: The procedure shown in Algorithm 8.2, called the `calcInfluence`(), calculates the influence of attribute A in a given set of A-dependent patterns, given all of those patterns contain the particular item r which is an item from the domain of A. The inputs are four databases, the first two correspond to the r-conditional databases for finding the contrast intensities of the patterns, the next two databases correspond to the reduced databases which exclude item r for finding the contrast intensities of the generalized patterns. The framework is similar to the pattern growth framework, which recursively projects conditional databases for each class, but the projections are guided by prefixes of the given patterns (line 1-8). The influence calculation is performed when it finds the database projections for the longest prefix of a particular pattern (line 11-20). Using the projected conditional databases represented as WZBDDs, the contrast intensity and the contrast influence of the attribute can be easily computed using the pattern's support values which correspond to the weight of the relevant WZBDDs.

The efficiency of this procedure relies on the use of WZBDD's caching mechanism

Algorithm 8.1 `mineGDC`($prefix$, D_{gen}, $[Dn_1, Dp_2, Dn_2]$)

```
Input: $D_{gen}$ : the generator data set, corresponds to the positive class in group $G_1$
    $Dn_1$: the negative class in group $G_1$
    $Dp_2$, $Dn_2$: the positive and negative classes in group $G_2$.
    All inputs are represented as WZBDDs
Output: The GDC patterns and the GDC influence of each attribute
 1: Let $x$ be the label of the top-node in $D_{gen}$; $prefix_x = prefix \cup \{x\}$
 2: for each $D$ in $[D_{gen}, Dn_1, Dp_2, Dn_2]$ do
 3:     $D_{(x)}$ = Find the $x$-conditional database of $D$
 4:     $D_{(\overline{x})}$ = Remove $x$ from database $D$ /* Compute the reduced database */
 5: end for
 6: $res_x$ = mineGDC($prefix_x$, $D_{gen(x)}$, $[Dn_{1(x)}, Dp_{2(x)}, Dn_{2(x)}]$) /* Grow prefixes which contain
    item $x$ */
 7: calcInfluence($x, res_x, G_{1(x)}, G_{2(x)}, G_1, G_2$) /* Update attribute influence from item $x$ and
    the $x$-conditional patterns GDC patterns */
 8: $res_{\overline{x}}$ = mineGDC($prefix$, $D_{gen(\overline{x})}$, $[Dn_{1(\overline{x})}, Dp_{2(\overline{x})}, Dn_{2(\overline{x})}]$) /* Grow prefixes which do not
    contain item $x$ */
 9: $result$ = node($x, res_x, res_{\overline{x}}$) /* Build the output node */
10: return $result$
11: Terminal cases:
12: Case 1: $D_{gen}$ contains an empty itemset
13:     Check GDC-constraint on $prefix$, using its support* in each class
14:     if ($prefix$ is a GDC pattern) then return 1
15:     else return 0
16:     end if
17: Case 2: $D_{gen}$ is empty: return 0
```

(*): The support of itemset $prefix$ in dataset D can be calculated using the WZBDD routine: weight(D_{prefix}), where D_{prefix} is the conditional database projected by $prefix$.

which allows intermediate computations, such as projecting secondary databases, to be shared across functions. So, the database projections performed in `calcInfluence()` may re-use the cached results from the database projections performed in `mineGDC()`, and vice versa, which avoids redundant computations.

8.5 Performance Study

In this section we evaluate our method and the performance of our algorithm for mining second-order contrasts. Our algorithms were implemented in C++, using the WZDD library routines developed previously for frequent itemset mining (Chapter 4). All experiments were conducted on a 3 GHz CPU, 4 GB RAM, running Solaris. The objectives of our experiments include:

- to compare the volume of GDC patterns with emerging patterns (EPs)

- to evaluate the runtime performance of our mining algorithm.
- to evaluate our proposed GDC based attribute ranking by comparing it against other methods
- to show that meaningful contrast influential attributes can be discovered by our method;

8.5.1 Pattern Volume Comparison

Figure 8.3a shows the number of patterns in the census data set using the first 20 attributes, with GDC-constraint: $\alpha = 1\%$, $\beta = 0.5\%$, and a varying δ_{gdc}, i.e. the minimum group discriminating power. In the 'white' race group, there are 5 million EPs. When δ_{gdc} is very small, almost every EP is a GDC pattern. As δ_{gdc} increases to 0.05, the number of GDC patterns drops by roughly 10% from the EPs. We identify 17 GDC influential attributes in such a scenario (shown in Table 8.3a). The other datasets, i.e. adult, satimage, and the ALL, also have similar trends (shown in Figure 8.3b, 8.3c, and 8.3d). The number of GDC patterns can still be overwhelming, but our technique can find the attributes which help explain the second-order contrast behaviour of those patterns, which we will discuss shortly.

8.5.2 Time Performance of the Mining Algorithm

We measure the runtime performance of our algorithms for mining the GDC patterns and their influential attributes using several data sets, with a varying minimum group discriminating power threshold, δ_{gdc}, where $\delta_{gdc} > 0$. We implemented 2 algorithms: i) *naive*: a two-phase algorithm which finds the GDC patterns, then for each attribute, calculates its influence by projecting the relevant patterns. ii) *mineGDC*: the algorithm described in Section 4.3 which simultaneously finds the influential attributes in the pattern mining phase.

We chose three real UCI data sets [71]: census, adult, and satimage, and a biological data set ALL leukaemia [85] which has a high dimensionality. In the census data set, we choose to find the first-order differences between male (as positive class) and female, and the second-order differences between two race groups: 'White', and the other races (combination of all other races in the data set) which we label as 'Non-White'. In the adult data set, the first-order contrast differentiates individuals with high income (i.e.>50K) from those with low income (i.e. $\leq$50K), grouped by their gender.

The satimage and ALL data sets contain multiple classes. We formulate the groups for those data sets by grouping the classes. Table 8.2 shows the class sizes in each data set.

Figure 8.3 shows that the number of patterns increases as the value of δ_{gdc} decreases. Figure 8.3b, moreover, shows that when δ_{gdc} is very low, all EPs are GDC patterns for the satimage dataset, which indicates that all first-order EPs for the primary group also have strong class-discriminating power.

Figure 8.4 shows the running time comparison between the two algorithms for finding the GDC influential attributes (and the corresponding patterns). Overall, when δ_{gdc} is high, there exist only a few patterns, for which both algorithms have similar runtime. As δ_{gdc} decreases, the discrepancy between the `mineGDC` and the `naive` algorithm increases. More specifically, the `mineGDC` algorithm is 4 times faster when $\delta_{gdc} = 0.01$ in the census data set, or when $\delta_{gdc} = 0.15$ in the satimage data set. This shows that the attribute's influence calculation can be performed efficiently using our technique, since *mineGDC* visits patterns which are shared by numerous attributes only once, and allows multiple re-use of intermediate database projections while finding the patterns and calculating the influence of the attributes.

8.5.3 Attribute Ranking Comparison

The census data set contains several household attributes and income attributes describing census data from the year 1970. With threshold values $\alpha = 1\%$, $\beta = 0.5\%$, and $\delta_{gdc} = 0.05$, we found 17 influential attributes for capturing group discriminative contrasts with the 'white' race group as primary group (Table 8.3a), out of 20 attributes which are included in our experiment. To evaluate our attribute ranking, we compare it against other rankings which are based on entropy measure [52], and the statistical Pearson's correlation measure.

The **entropy-based ranking** is based on the information gain of an attribute, which measures its ability to improve class discrimination. The columns in Table 8.4 show the info gain of each attribute in each group, and the info-gain difference across the groups, labeled *IGDiff*. Attributes *group-quarter-type* (i.e. the type of housing) and *marital-status* appear in the top-5 attributes in our GDC based ranking as well as in this entropy-based ranking. It shows our technique is able to identify such attributes whose male-vs-female discrimination ability is stronger in the 'white' group than their discrimination ability in the other group. *Group quarter* and *farm* attributes, which are highly ranked by entropy, however, are not identified by our method. It suggests that patterns containing those attributes have weak group discriminating contrast influence. If we look closer at their info gain differences, the values are actually very small, which means that there is no significant difference of their class discrimination ability across the groups, explaining why they are not identified by our method. Our attribute ranking identifies other influential attributes which have low ranks in the entropy-based ranking, which shows the ability of the GDC based ranking to identify the interdependency between multiple attributes, whereas an entropy measure treats each attribute independently. Later in this section, we will show a more interesting result regarding those attributes whose entropy-based ranks are lower than their GDC-based ranks.

The top-10 attributes found using a **correlation** measure are listed in Table 8.4. For each attribute, $Corr(g)$ is the Pearson's correlation coefficient between the values of that attribute in the positive class and the values of that attribute in the negative class in group g. A large correlation value indicates that an attribute is a poor class-discriminator in g, because its values vary closely between the classes. The score of correlation difference between groups, denoted $CorrDiff = \text{Corr}(G_2) - \text{Corr}(G_1)$, measures how much an attribute correlates with the classes in the secondary group, but does not correlate with the classes in the primary group. The most influential attribute in our ranking(see Table 8.3a), i.e. *group-quarter-type*, has a negative correlation difference score, meaning that it is a weaker class discriminator in the primary group compared to the secondary group. Like the entropy measure, this result shows that a correlation measure does not to identify the interdependency between multiple attributes, which can be identified by our method.

8.5.4 A Meaningful Discovery

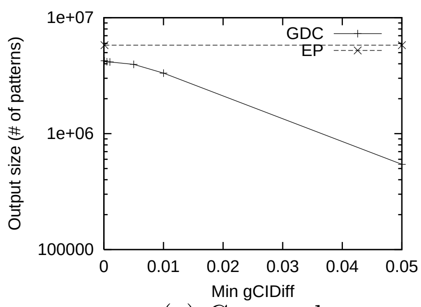
The attributes which are influential for capturing GDC patterns when the 'white' race is chosen as primary group are shown in Table 8.3a. The GDC influential attributes when the 'non-white' race is chosen as primary group are shown in Table 8.3b. Attributes that have positive global GDC influence for one race group but have zero or negative influence in the other group are marked by asterisks (*). Interestingly, the first two attributes, *group-quarter-type*, (i.e. type of housing), and *num-of-families-in-household*, are the top-2 attributes in both groups, suggesting that they have an equally high importance for finding group-discriminative male-vs-female contrast in each group.

Based on the GDC based ranking for each race group, attribute *monthly-rent*

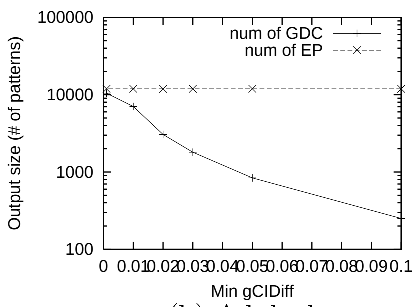
Table 8.2: Class names and sizes in each group in four data sets

	Group 1	
Data set	Positive class	Negative class
Census	White.Male (2957)	White.Female (3089)
Adult	Male.>50K (6662)	Male.≤50K (15128)
Satimage	C1 (1072)	C7 (1038)
ALL	BCR+ABL (9)	E2A+PBX1 (18)

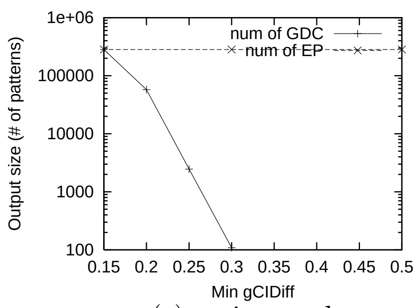
	Group 2	
Data set	Positive class	Negative class
Census	Not-White.Male (448)	Not-White.Female (525)
Adult	Female.>50K (1179)	Female.≤50K (9592)
Satimage	C2+C3 (1440)	C4+C5 (885)
ALL	MLL (14)	T-ALL (28)



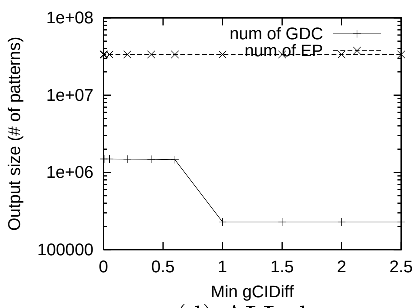
(a) Census data set (nattr = 20)



(b) Adult data set (nattr = 13)

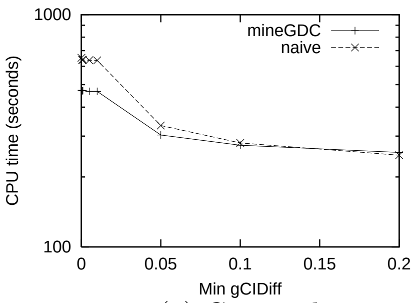


(c) satimage data set (nattr = 36)

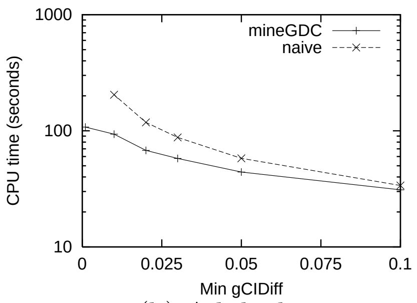


(d) ALL data set (nattr = 50)

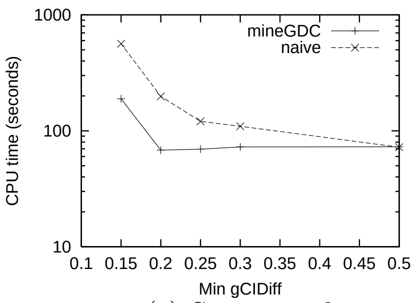
Figure 8.3: Comparison between the number of GDC patterns and emerging patterns



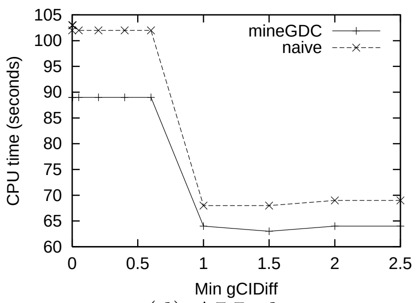
(a) Census data set (nattr = 20)



(b) Adult data set (nattr = 13)



(c) Satimage data set (nattr = 36)



(d) ALL data set (nattr = 50)

Figure 8.4: Runtime comparison between the `mineGDC` algorithm and the `naive` algorithm

Table 8.3: Attribute ranking by GDC (global) influence

Rank	Att.name	GDC influence
1	group-quarter-type	38408.1
2	num-of-families(in-household)	38408.1
3	monthly-rent*	36215.1
4	relationship-to-householder	35590.2
5	marital-status*	24987.3
6	house-ownership*	24971.6
7	mother-location(in-household)*	23022.3
8	father-location(in-household)*	15495.5
9	age-of-eldest-child(in-household)	11012.2
10	family-total-income*	7033.2
11	spouse-location(in-household) *	5756.4
12	age-of-youngest-child(in-household)	4424.0
13	num-of-fathers(in-household)	1171.1
14	family-size*	275.6
15	family-unit	119.8
16	age	119.2
17	house-value*	2.5
18	num-of-couples(in-household)	0.0
19	farm	0.0
20	group-quarter	0.0

(a) 'White' race as primary group; Attributes marked by (*) do not have a GDC influence in the other ranking in Table 8.3b

Rank	Att.name	GDC influence
1	num-of-families(in-household)	872.8
2	group-quarter-type	871.3
3	num-of-fathers(in-household)	315.1
4	father-location(in-household)	313.5
5	relationship-to-householder	300.0
6	num-of-couples(in-household)*	54.9
7	age-of-eldest-child(in-household)	11.6
8	age	4.3
9	family-unit	4.0
10	age-of-youngest-child(in-household)	2.3
11	marital-status	0.0
12	family-size	0.0
13	spouse-location(in-household)	0.0
14	mother-location(in-household)	0.0
15	family-total-income	0.0
16	monthly-rent	0.0
17	house-value	0.0
18	house-ownership	0.0
19	farm	0.0
20	group-quarter	0.0

(b) 'Non-white' race as primary group; The attribute marked by (*) does not have a GDC influence in the other ranking shown in Table 8.3a

Table 8.4: Attribute ranking by entropy and correlation based influence

Rank	Att.name	IG(G1)	IG(G2)	IGDiff
1	marital-status	0.973	0.952	0.021
2	group-quarter-type	0.995	0.983	0.012
3	house-ownership	0.999	0.988	0.010
4	group-quarter	0.999	0.989	0.009
5	farm	0.999	0.995	0.005
6	num-of-fathers	0.999	0.995	0.004
7	num-of-families	0.999	0.995	0.004
8	monthly-rent	0.999	0.995	0.004
9	house-value	0.999	0.995	0.004
10	age-of-youngest-child	0.999	0.995	0.004

(a) Ranking based on group information gain (IG) difference; G1 = 'white' race group, G2 = 'non-white' race group

Rank	Att. name	Corr(G_1)	Corr(G_2)	CorrDiff
1	num-of-families	-0.006	0.158	0.164
2	age	-0.005	0.099	0.104
3	mother-location	-0.007	0.086	0.094
4	family-size	-0.020	0.054	0.075
5	father-location	0.008	0.046	0.039
6	marital-status	-0.009	0.025	0.034
7	house-value	-0.018	-0.004	0.015
8	farm	-0.002	0	0.002
9	family-unit	-0.009	-0.015	-0.006
10	monthly-rent	-0.008	-0.015	-0.007

(b) Ranking based on group correlation difference; G1 = 'white' race group, G2 = 'non-white' race group appears to be influential only for the 'white' race and not for the other race group. Since it does not appear in the top-5 attributes in the other rankings (i.e. entropy-based and correlation-based), it shows that when considered individually, it does not differentiate the male and female differences across groups. However, our ranking suggests that *monthly-rent* can help capture the group differences when it is combined with some other attribute(s) within GDC patterns. To give a specific example, we found that specifying *monthly rent* in the following rule:

'*do not live with a spouse, and monthly rent* $>$ *$125,*'

increases the between-groups difference of this rule's male-vs-female discriminating ability. Based on this rule, someone who does not live with a spouse and pays high monthly rent, is more likely to be a male in the 'white' race group, but without considering *monthly rent*, it is equally likely that the individual is a male or a female, in either race group. This example shows that a GDC influential attribute can help identify important sub-categories (corresponding to GDC patterns) in the population, such as a category of people who do not live with a spouse and pay high monthly rent, in which there is a strong differentiation between male and female in the 'white' race group but there is not a strong differentiation between male and female in the other group [1].

8.6 Related Work

Our method for measuring an attribute's influence is pattern based, which has not been addressed in previous work. Existing feature selection techniques have a common objective of finding attributes which are most relevant to the data classification. Entropy [52] based techniques measure the class discriminating ability of an attribute independently of the other attributes. A recent work [74] proposes a correlation-based technique for finding a set of features which have high inter-correlation among themselves, and low correlation with the other features. Their method for measuring the significance of an attribute-set may be used for measuring the discriminating ability of an attribute in first-order contrast as well as second-order contrast. The difference, however, is that our model can identify subspaces, instead of the entire data space, in which second-order contrast occurs.

[1]This rule corresponds to our intuition since it is likely that there are much more white people who pay high rent than non-white people, hence, the amount of rent is more useful for discriminating male and female in the white population than it is for the non-white population.

Group discriminative contrast patterns correspond to subspaces of high class-contrast in the primary group and low class-contrast in the secondary group. Mining interesting subspaces has been previously studied for solving other data mining problems, such as in [2] for finding outliers in high-dimensional data sets. However, there has not been any work which addresses the problem of second-order differentiation. Work [139] addresses the problem of finding contrast sets, which are first-order contrasts between multiple groups (i.e. classes) of data instances.

Work in [37, 160], proposed techniques for finding (first-order) contrasts between classes. Moreover, work in [153] studies a technique for comparing frequent patterns between classes, which may be extended to comparing contrast patterns between groups. However, their method cannot identify the influence of each attribute that causes the differences, which is a novel aspect of our technique. Previous work in [50] uses χ^2-test to measure the significance of an item in the discriminating power of an emerging pattern. Such a measure may be used for measuring the attribute's local influence in a pattern in a given group of classes, but unlike our method, the χ^2 measure is independent to which class is chosen as the positive (or negative) class.

8.7 Summary

In this chapter, we have introduced a method for finding attributes which are influential for capturing contrast between classes in a group, as well as the (second-order) contrast across groups. Our experiments showed that our method can overcome the limitation of classical attribute selection techniques, which do not take into account the inter-dependency between multiple attributes which may vary between patterns and across groups. Using our method, moreover, an influential attribute can help explain the key underlying factors of the contrast behaviour that is found in certain data sub-categories, instead of considering the entire data as in the classical feature selection techniques. We have shown that Weighted ZBDDs allow complex patterns, such as second order contrasts, to be mined efficiently. This level of complexity has not been addressed by any of the previous pattern mining techniques.

Algorithm 8.2 `calcInfluence`$(A, r, P, [G_{1(r)}, G_{2(r)}, G_1, G_2])$

```
Input: $A$: an attribute
    $r$: a value from the domain of $A$
    $P$: a set of $r$-dependent GDC patterns and their gCIDiff values
    $G_{1(r)} = [Dp_{1(r)}, Dn_{1(r)}]$: $r$-conditional databases from group $G_1$
    $G_{2(r)} = [Dp_{2(r)}, Dn_{2(r)}]$: $r$-conditional databases from group $G_2$
    $G_1 = [Dp_1, Dn_1]$: databases from group $G_1$
    $G_2 = [Dp_2, Dn_2]$: databases from group $G_2$
    $P$, $G_{1(r)}$, $G_{2(r)}$, $G_1$, and $G_2$ are represented as WZBDDs
Output: The GDC influence of item $r$ in $P$
 1: $x$ = the label of the top-node in $P$.
 2: for each $G_i$ in $[G_{1(r)}, G_{2(r)}, G_1, G_2]$ do
 3:    $G_{i(x)}$ = Find the $x$-conditional of databases from $Dp_{G_i}$ and $Dn_{G_i}$
 4:    $G_{i(\overline{x})}$ = Remove $x$ from $Dp_{G_i}$ and $Dn_{G_i}$ in group $G_i$ /* Find the reduced databases */
 5: end for
 6: calcInfluence($A$, $r$, $P_x$, $[G_{1(r.x)}, G_{2(r.x)}, G_{1(x)}, G_{2(x)}]$)
    /* Calculate influence of $A$ in patterns which contain $x$ */
 7: calcInfluence($A$, $r$, $P_{\overline{x}}$, $[G_{1(r.\overline{x})}, G_{2(r.\overline{x})}, G_{1(\overline{x})}, G_{2(\overline{x})}]$))
    /* Calculate influence of $A$ in patterns which do not contain $x$ */
 8: return
 9: Terminal cases:
10: if $P$ contains an empty itemset then
11:    Let $pref_{spec}$ be the itemset that projects $G_{1(r)}$, $G_{2(r)}$
12:    $pref_{gen} = pref_{spec} \setminus r$ /* $pref_{gen}$ is the $A$-generalization of $pref$ */
13:    $gCIDiff_{spec}$ = CI($pref_{spec}, G_1$) - CI($pref_{spec}, G_2$)
       /* Calculate gCIDiff for $pref_{spec}$ */
14:    $gCIDiff_{gen}$ = CI($pref_{gen}, G_1$) - CI($pref_{gen}, G_2$)
       /* Calculate gCIDiff for $pref_{gen}$ */
15:    influence[$A$] += $\frac{gCIDiff_{spec} - gCIDiff_{gen}}{gCIDiff_{spec}}$
       /* Update the global GDC influence of attribute $A$ */
16:    return
17: end if
18: if $P$ is empty then
19:    return
20: end if
```

Note: CI($pref_{spec}$,G_i) is calculated using the weight of the WZBDDs of $D_{p_{i(r)}}$ and $D_{n_{i(r)}}$ which represent the conditional databases projected by $pref_{spec}$. CI($pref_{gen}$,G_i) is calculated using the weight of the WZBDDs of D_{p_i} and D_{n_i} which represent the conditional databases projected by $pref_{gen}$.

Chapter 9

Conclusions

In this thesis, we have shown that Binary Decision Diagrams can be a powerful tool for solving pattern mining problems in a variety of situations. We introduced original variants of Binary Decision Diagrams, namely Weighted Zero-suppressed Binary Decision Diagrams (ZBDDs) and Weighted Sequence Binary Decision Diagrams (SeqBDDs). We studied the problems of mining the fundamental frequent itemsets, and frequent subsequences. We also studied contrast mining, considering the simple types of contrasts, as well as the complex types of contrasts, such as expressive contrasts and second order contrasts.

The Weighted Zero-suppressed Binary Decision Diagrams are useful for mining itemset-based patterns, such as frequent itemsets and contrast patterns, whereas the weighted Sequence Binary Decision Diagrams are useful for mining sequential patterns. These new graph-based data structures are novel in data mining. We believe that our weighted BDDs are promising for solving difficult pattern mining problems, which could not been solved using previous techniques. Although their performance may be limited when the patterns are rare, our weighted BDD based pattern mining algorithms have been shown to be able to outperform the existing tree-based algorithms, especially when there exist a large volume of patterns.

We now give a summary of the thesis, and then a list of interesting future research directions.

9.1 Thesis Summary

In Chapter 4, we proposed an original variant of Zero-suppressed Binary Decision Diagrams, called Weighted Zero-suppressed Binary Decision Diagrams. We proposed a family of algorithms based on Weighted Zero-suppressed Binary Decision Diagrams for mining frequent patterns, such as frequent itemsets and their closed/maximal variants. The algorithms include `FIMiner` for mining frequent itemsets, `MFIMiner` for mining maximal frequent itemsets, `CFIMiner` for mining closed frequent itemsets, and `RowCFIMiner` for mining closed frequent itemsets using the row-wise mining framework.

In previous techniques, prefix trees are the commonly used data structure. On the other hand, ZBDDs and their weighted variant are directed acyclic graphs. We have discussed the advantages and disadvantages of using ZBDDs or trees for mining frequent patterns. ZBDDs can outperform trees, due to the BDD's ability to share and re-use the result of similar computations. Redundancy in performing intermediate computations, such as repeatedly finding patterns from the same conditional databases, can now be avoided through out the mining routine. Moreover, they also allow node sharing across multiple databases, such as the input database, the output, and the conditional databases, which is not allowed in trees. The use of ZBDDs and weighted ZBDDs is proven to be useful for mining a large number of high dimensional patterns.

In Chapter 5, we proposed an original Sequence Binary Decision Diagram (SeqBDD), which is a graph data structure useful for mining sequential patterns. We proposed a SeqBDD-based algorithm for mining frequent subsequences, called `SeqBDDMiner`. Tree data structures are not preferred by previous sequential pattern mining techniques. When mining frequent itemsets, compact trees can be obtained by re-ordering the items. However, this is not possible for the sequences context, since the the ordering of items in sequences must be preserved. Our study showed that a graph data structure, such as Sequential BDDs (SeqBDDs), can be useful. Although a SeqBDD may not be much smaller than a tree, the BDD's feature of re-using nodes and computation results between multiple intermediate databases is powerful. `SeqBDDMiner` allows efficient mining of a large number of long subsequences, such as those in DNA and protein sequences.

In Chapter 6, we studied the problem of mining expressive contrasts. We proposed a novel type of expressive contrast patterns, called disjunctive emerging patterns. Disjunctive emerging patterns allow disjunctions within attributes, and conjunctions across attributes. We proposed a family of algorithms for mining the pure conjunctive emerging patterns (`EPMiner`), and for the disjunctive emerging patterns (`DEPMiner`).

We have discussed the strength and limitation of using ZBDDs, weighted ZBDDs, or trees, for mining contrasts. When a ZBDD is used as the primary data structure, a secondary bitmap data representation is required to perform the frequency calculation. Bitmap-based frequency calculation can be a bottleneck to the algorithm's efficiency. When a Weighted ZBDD is used, the bitmap is no longer needed. When the number of patterns is enormous and the input data is relatively large, the weighted ZBDDs are more efficient than the ZBDDs. However, the ZBDDs may be more compact than weighted ZBDDs when the data is sparse, or not many patterns exist. Compared to previous techniques based on prefix trees, our graph-based algorithms scale well for mining a large number of contrasts in high dimensional data sets. As evidence, we found circumstances where the non-disjunctive EPs are rare, but disjunctive EPs exist in abundance.

In Chapter 7, we studied how useful are disjunctive emerging patterns for building classifiers. Since disjunctions of attribute values may correspond to a non-contiguous range of values in ordered domain, a disjunctive pattern may contain gaps or holes. We employed a significance test on those gaps and on the overall patterns, and investigated how these influence the classification accuracy. We found that the significance tests are beneficial when many patterns exist and they contain many gaps. Our classifier, called `CNFClassifier` (due to the correspondence between a disjunctive pattern and a CNF boolean function) can be more accurate than the simple contrast based classifier, especially when the data is sparse. Allowing disjunctions is shown to be useful for building more robust classifiers, and overcoming the limitation of the EP-based classifiers, whose performance is sensitive to the sparsity/density of the data.

In Chapter 8, we studied the problem of second order contrasts. Finding contrasts of contrasts (i.e. second order contrasts) between groups of classes is an interesting problem because of the multiple class scenario, and the multiple constraints associated with those classes. All of the previous contrast mining techniques have so far only considered first order contrasts. We introduced a novel type of second order contrast patterns, called Group Discriminative Contrast (GDC) patterns, and a method for finding the attributes which have strong influence in those patterns. Our proposed algorithm, based on weighted ZBDDs, can simultaneously mine GDC patterns and the influential attributes. We found influential attributes which otherwise can not be found using traditional feature ranking techniques. The weighted ZBDDs play an important role in our algorithm, due to their ability to compactly represent numerous databases, which allows multiple constraints to be pushed deep inside the routine.

Table 9.1: List of algorithms

Algorithm	Application	Related Chapter
`FIMiner`	Frequent itemset mining	4
`MFIMiner`	Maximal frequent itemset mining	4
`RowCFIMiner`	Row-wise closed frequent itemset mining	4
`SeqBDDMiner`	Frequent subsequence mining	5
`EPMiner`	Emerging pattern mining	6
`DEPMiner`	Disjunctive emerging pattern mining	6
`CNFClassifier`	Disjunctive emerging pattern based classification	7
`GDCMiner`	Group discriminative contrast mining; Second order contrast mining	8

To summarise, Table 9.1 lists the algorithms we have discussed in this thesis and their applications.

9.2 Future Research Directions

Some possible future research directions are listed below:

- It is interesting to study hybrid mining techniques that combine the strength of Binary Decision Diagrams and existing prefix-tree structures. The two data structures may benefit each other in terms of data compression, allowing pruning to take place early during mining, and re-using of intermediate results.
- The types of fundamental patterns being studied in our research include frequent itemsets and frequent subsequences. Exploring other constraints on those patterns, and investigating the behaviour of the BDD-based algorithms for mining them would be interesting.
- Previous work on contrast mining has studied various constraints on emerging patterns, as well as various classification models based on emerging patterns. Extending them to the context of expressive contrasts would be interesting. Whether

they are able to provide solutions to the limitation of simple contrasts is also worth investigating.

- It is interesting to study other applications of second order contrasts, such as multi-group classification, which has not been addressed in any of the previous work.

Bibliography

[1] Ramesh C. Agarwal, Charu C. Aggarwal, and V. V. V. Prasad. Depth first generation of long patterns. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'00)*, pages 108–118, Boston, MA, USA, August 2000.

[2] Charu C. Aggarwal and Philip S. Yu. An effective and efficient algorithm for high-dimensional outlier detection. *The VLDB Journal - The International Journal on Very Large Data Bases*, 14(2):211–221, April 2005.

[3] Rakesh Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the Fifth International Conference on Extending Database Technology (EDBT'96)*, pages 3–17, Avignon, France, March 1996.

[4] Rakesh Agrawal and Tomasz Imielinski. Mining association rules between sets of items in large databases. In *Proceedings of 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington, DC, USA, May 1993.

[5] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB'94)*, pages 487–499, Santiago, Chile, September 1994.

[6] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In *Proceedings of the Eleventh IEEE International Conference on Data Engineering (ICDE'95)*, pages 3–14, Taipei, Taiwan, March 1995.

[7] Sheldon B. Akers. On a theory of boolean functions. *Journal on Applied Mathematics*, 7(4):487–498, December 1959.

[8] Fadi A. Aloul, Igor L. Markov, and Karem A. Sakallah. MINCE: A static global variable ordering for SAT and BDD. *The Journal of Universal Computer Science*, 10(12):1562–1596, 2004.

[9] Fadi A. Aloul, Maher N. Mneimneh, and Kareem A. Sakallah. ZBDD-based backtrack search SAT solver. In *Proceedings of the Eleventh IEEE/ACM International Workshop on Logic and Synthesis (IWLS'02)*, pages 131–136, New Orleans, Louisiana, USA, June 2002.

[10] Maria-Luiza Antonie and Osmar R. Zaïane. An associative classifier based on positive and negative rules. In *Proceedings of the Ninth ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'04)*, pages 64–69, Paris, France, June 2004.

[11] Maria-Luiza Antonie, Osmar R. Zaïane, and Alexandru Coman. Application of data mining techniques for medical image classification. In *Proceedings of the Second International Workshop on Multimedia Data Mining (MDM/KDD 2001) in conjunction with the Sevent ACM SIGKDD*, pages 94–101, San Francisco, California, USA, August 2001.

[12] Musa H. Asyali, Dilek Colak, Omer Demirkaya, and Mehmet S. Inan. Gene expression profile classification: A review. *Current Bioinformatics*, 1(1):55–73, January 2006.

[13] Ricardo A. Baeza-Yates. Searching subsequences. *Theoretical Computer Science*, 78(2):363–376, January 1991.

[14] James Bailey, Thomas Manoukian, and Ramamohanarao Kotagiri. Fast algorithms for mining emerging patterns. In *Proceedings of the Sixth European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'02)*, pages 39–50, Helsinki, Finland, August 2002.

[15] James Bailey, Thomas Manoukian, and Ramamohanarao Kotagiri. Classification using constrained emerging patterns. In *Proceedings of the Fourth International Conference on Web Age Information Management (WAIM'03)*, pages 226–237, Chengdu, China, August 2003.

[16] James Bailey, Thomas Manoukian, and Ramamohanarao Kotagiri. A fast algorithm for computing hypergraph transversals and its application in mining emerging patterns. In *Proceedings of the Third IEEE International Conference*

on Data Mining (ICDM'03), pages 485–488, Washington, DC, USA, November 2003.

[17] Stephen D. Bay and Michael J. Pazzani. Detecting group differences: Mining contrast sets. *Data Mining and Knowledge Discovery*, 5(3):213–246, 2001.

[18] Roberto J. Bayardo. Efficiently mining long patterns from databases. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data (SIGMOD'98)*, pages 85–93, Seattle, Washington, USA, June 1998.

[19] Dennis A. Benson, Ilene Karsch-Mizrachi, David J. Lipman, James Ostell, and David L. Wheeler. GenBank, January 2008. `http://www.ncbi.nlm.nih.gov/Genbank/`.

[20] Francesco Bonchi, Fosca Giannotti, Alessio Mazzanti, and Dino Pedreschi. Ex-Ante: Anticipated data reduction in constrained pattern mining. In *Proceedings of the Seventh European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'03)*, pages 47–58, Cavtat-Dubrovnik, Croatia, September 2003.

[21] Francesco Bonchi and Bart Goethals. FP-Bonsai: The art of growing and pruning small FP-trees. In *Proceedings of the Eighth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'04)*, pages 155–160, Sydney, Australia, May 2004.

[22] Endre Boros, Khaled M. Elbassioni, Vladimir Gurvich, and Leonid Khachiyan. An efficient implementation of a quasi-polynomial algorithm for generating hypergraph transversals. In *Proceedings of The Eleventh Annual European Symposium on Algorithms*, pages 556–567, Budapest, Hungray, September 2003.

[23] Endre Boros, Khaled M. Elbassioni, Vladimir Gurvich, Leonid Khachiyan, and Kazuhisa Makino. An intersection inequality for discrete distributions and related generation problems. In *Proceedings of the Thirtieth International Colloquium on Automata, Languages and Programming (ICALP'03)*, pages 543–555, Eindhoven, The Netherlands, June 2003.

[24] Riccardo Boscolo, James C. Liao, and Vwani P. Roychowdhury. An information theoretic exploratory method for learning patterns of conditional gene coexpression from microarray data. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 5(1):15–24, 2008.

[25] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *Transactions on Computers*, 35(8):677–691, August 1986.

[26] Randal E. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. *ACM Computer Surveys*, 24(3):293–318, 1992.

[27] Randal E. Bryant and Ying-An Chen. Verification of arithmetic circuits with binary moment diagrams. In *Proceedings of the Thirty Second ACM/IEEE Design Automation Conference (DAC'95)*, pages 535–541, San Francisco, California, USA, June 1995.

[28] Cristian Bucilă, Johannes Gehrke, Daniel Kifer, and Walker White. DualMiner: A dual-pruning algorithm for itemsets with constraints. *Data Mining and Knowledge Discovery*, 7(3):241–272, 2003.

[29] Douglas Burdick, Manuel Calimlim, Jason Flannick, Johannes Gehrke, and Tomi Yiu. MAFIA: A maximal frequent itemset algorithm. *IEEE Transactions on Knowledge and Data Engineering*, 17(11):1490–1504, 2005.

[30] Philippe Chatalic and Laurent Simon. Multi-resolution on compressed sets of clauses. In *The Twelfth IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2000)*, pages 2–10, Vancouver, BC, Canada, November 2000.

[31] Hong Cheng, Xifeng Yan, and Jiawei Han. IncSpan: Incremental mining of sequential patterns in large databases. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery in Databases (KDD'04)*, pages 527–532, Seattle, WA, USA, 2004.

[32] Edmund M. Clarke, M. Fujita, and X. Zhao. Multi-terminal decision diagrams and hybrid decision diagrams. *Representations of Discrete Functions*, pages 93–108, 1996.

[33] Gao Cong, Kian-Lee Tan, Anthony K. H. Tung, and Xin Xu. Mining top-k covering rule groups for gene expression data. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, pages 670–681, Baltimore, Maryland, USA, June 2005.

[34] Gao Cong, Anthony K. H. Tung, Xin Xu, Feng Pan, and Jiong Yang. FARMER: Finding interesting rule groups in microarray datasets. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 143–154, Paris, France, June 2004.

[35] Chad Creighton and Samir Hanash. Mining gene expression databases for association rules. *Bioinformatics*, 19(1):79–86, 2003.

[36] Maxime Crochemore and Renaud Vérin. On compact directed acyclic word graphs. *Structures in Logic and Computer Science*, 1261:192–211, 1997.

[37] Guozhu Dong and Jinyan Li. Efficient mining of emerging patterns: Discovering trends and differences. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'99)*, pages 43–52, San Diego, California, USA, August 1999.

[38] Guozhu Dong and Jinyan Li. Mining border descriptions of emerging patterns from dataset pairs. *Knowledge and Information Systems (KAIS)*, 8(2):178–202, August 2005.

[39] Guozhu Dong, Jinyan Li, and Ramamohanarao Kotagiri. Making use of the most expressive jumping emerging patterns for classification. *Knowledge and Information Systems (KAIS)*, 3(2):131–145, May 2001.

[40] Guozhu Dong, Jinyan Li, and Limsoon Wong. The use of emerging patterns in the analysis of gene expression profiles for the diagnosis and understanding of diseases. *New Generation of Data Mining Applications*, pages 331–354, April 2005.

[41] Guozhu Dong, Jinyan Li, and Xiuzhen Zhang. Discovering jumping emerging patterns and experiments on real datasets. In *Proceedings on the Ninth International Conference (IDC'99)*, pages 43–52, Hong Kong, July 1999.

[42] Guozhu Dong, Xiuzhen Zhang, Limsoon Wong, and Jinyan Li. CAEP: Classification by aggregating emerging patterns. In *Proceedings of the Second International Conference on Discovery Science (DS'99)*, pages 30–42, Tokyo, Japan, December 1999.

[43] Jeff Edmonds, Jarek Gryz, Dongming Liang, and Renée J. Miller. Mining for empty spaces in large data sets. *Theoretical Computer Science*, 296(3):435–452, March 2003.

[44] Thomas Eiter and Georg Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6):1278–1304, 1995.

[45] Mohammad El-Hajj and Osmar R. Zaïane. Non recursive generation of frequent k-itemsets from frequent pattern tree representations. In *Proceedings of the Fifth International Conference on Data Warehousing and Knowledge Discovery (DaWak'03)*, September 2003.

[46] Mohammad El-Hajj, Osmar R. Zaïane, and Paul Nalos. Bifold constraint-based mining by simultaneous monotone and anti-monotone checking. In *Proceedings of the Fifth IEEE International Conference on Data Mining (ICDM'05)*, New Orleans, Louisiana, USA, November 2005.

[47] Christie I. Ezeife and Yi Lu. Mining web log sequential patterns with position coded pre-order linked WAP-tree. *Data Mining and Knowledge Discovery*, 10(1):5–38, 2005.

[48] Christie I. Ezeife, Yi Lu, and Yi Liu. PLWAP sequential mining: Open source code. In *Proceedings of the First International Workshop on Open Source Data Mining (OSDM'05)*, pages 26–35, Chicago, Illinois, USA, 2005.

[49] Hongjian Fan and Ramamohanarao Kotagiri. A bayesian approach to use emerging patterns for classification. In *Proceedings of the Fourteenth Australasian Database Conference (ADC'03)*, pages 39–48, Adelaide, Australia, 2003.

[50] Hongjian Fan and Ramamohanarao Kotagiri. Noise tolerant classification by chi emerging patterns. In *Proceedings of the Eighth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'04)*, pages 201–206, Sydney, Australia, May 2004.

[51] Hongjian Fan and Ramamohanarao Kotagiri. Fast discovery and the generalization of strong jumping emerging patterns for building compact and accurate classifiers. *IEEE Transactions on Data Engineering*, 18(6):721–737, 2006.

[52] Usama M. Fayyad and Keki Irani. Multi-interval discretization of continuous-valued attributes for classification. In *Proceedings of Thirteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 1022–1029, Chambery, France, August 1993.

[53] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery: An overview. *Advances in Knowledge Discovery and Data Mining*, pages 1–34, 1996.

[54] Pedro G. Ferreira and Paulo J. Azevedo. Protein sequence classification through relevant sequences and bayes classifiers. In *Proceedings of the Twelfth Portuguese Conference on Progress in Artificial Intelligence (EPIA'05)*, pages 236–247, Covilhã, Portugal, December 2005.

[55] Robert D. Finn, John Tate, Jaina Mistry, Penny C. Coggill, Stephen John Sammut, Hans-Rudolf Hotz, Goran Ceric, Kristoffer Forslund, Sean R. Eddy, Erik L. L. Sonnhammer, and Alex Bateman. The pfam protein families database. *Nucleic Acids Research*, 36:281–288, January 2008. `http://www.sanger.ac.uk/Software/Pfam`.

[56] William J. Frawley, Gregory Piatetsky-Shapiro, and Christopher J. Matheus. Knowledge discovery in databases. *Advances in Knowledge Discovery and Data Mining*, pages 1–30, 1991.

[57] Hiroshige Fujii, Goichi Ootomo, and Chikahiro Hori. Interleaving based variable ordering methods for ordered binary decision diagrams. In *Proceedings of IEEE/ACM International Conference on Computer Aided Design (ICCAD'93)*, pages 38–41, Santa Clara, California, USA, November 1993.

[58] Masahiro Fujita, Hisanori Fujisawa, and Nobuaki Kawato. Evaluation and implementation of boolean comparison method based on binary decision diagrams. In *Proceedings of the IEEE//ACM International Conference on Computer Aided Design (ICCAD'88)*, pages 2–5, Santa Clara, California, USA, November 1988.

[59] Jordan Gergov and Christoph Meinel. Efficient analysis and manipulation of OBDDs can be extended to FBDDs. *IEEE Transactions on Computers*, 43(10):1197–1209, 1994.

[60] Amol Ghoting, Gregory Buehrer, Srinivasan Parthasarathy, Daehyun Kim, Anthony Nguyen, Yen-Kuang Chen, and Pradeep Dubey. Cache-conscious frequent pattern mining on modern and emerging processors. *The International Journal on Very Large Data Bases*, 16(1):77–96, January 2007.

[61] Bart Goethals and Mohammed Javeed Zaki, editors. *FIMI'03: Frequent Itemset Mining Implementation*, volume 90 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2003.

[62] Bart Goethals and Mohammed Javeed Zaki, editors. *FIMI'03, Frequent Itemset Mining Implementations, Proceedings of the ICDM 2003 Workshop on Frequent*

Itemset Mining Implementations, 19 December 2003, Melbourne, Florida, USA, volume 90 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2003.

[63] Karam Gouda and Mohammed Javeed Zaki. Efficiently mining maximal frequent itemsets. In *Proceedings of IEEE International Conference on Data Mining (ICDM'01)*, pages 163–170, San Jose, California, USA, November 2001.

[64] Gösta Grahne and Jianfei Zhu. Efficiently using prefix-trees in mining frequent itemsets. In *Proceedings of the IEEE ICDM 2003 Workshop on Frequent Itemset Mining Implementations (FIMI'03)*, pages 123–132, Melbourne, Florida, USA, December 2003.

[65] Dimitrios Gunopulos, Roni Khardon, and Ram Sewak Sharma. Discovering all most specific sentences. *ACM Transactions on Database Systems*, 28(2):140–174, 2003.

[66] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. Frequent pattern mining: Current status and future directions. *Data Mining and Knowledge Discovery*, 5(1):55–86, August 2007.

[67] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques.* Morgan Kauffman, 2000.

[68] Jiawei Han and Jian Pei. Can we push more constraints into frequent pattern mining? In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery in Databases (KDD'00)*, pages 350–354, Boston, MA, USA, August 2000.

[69] Jiawei Han, Jian Pei, and Laks V. S. Lakshmanan. Mining frequent itemsets with convertible constraints. In *Proceedings of the Seventeenth IEEE International Conference on Data Engineering (ICDE'01)*, pages 433–442, Heidelberg, Germany, April 2001.

[70] Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery*, 8(1):53–87, 2004.

[71] Seth Hettich and Stephen D. Bay. The UCI KDD archive, 1999. `http://kdd.ics.uci.edu`.

[72] Masahiro Hirao, Hiromasa Hoshino, Ayumi Shinohara, Masayuki Takeda, and Setsuo Arikawa. A practical algorithm to find the best subsequence patterns. In *Proceedings of the Third International Conference on Discovery Science (DS'00)*, pages 141–154, Kyoto, Japan, December 2000.

[73] Haym Hirsh. Generalizing version spaces. *Machine Learning*, 17(1):5–45, October 1994.

[74] Michael E. Houle and Nizar Grira. A correlation-based model for unsupervised feature selection. In *Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management (CIKM'07)*, pages 897–900, Lisbon, Portugal, November 2007.

[75] IBM. Synthetic data generation code for association rules and sequential patterns. intelligent information systems, IBM almaden research center. `http://www.almaden.ibm.com/software/quest/resources`.

[76] Haruya Iwasaki, Shin-Ichi Minato, and Thomas Zeugmann. A method of variable ordering for zero-suppressed binary decision diagrams in data mining applications. In *The Third IEEE International Workshop on Databases for Next-Generation Researchers (SWOD'07)*, pages 85–90, Istanbul, Turkey, April 2007.

[77] Xionan Ji, James Bailey, and Guozhu Dong. Mining minimal distinguishing subsequence patterns with gap constraints. *Knolwedge and Information Systems (KAIS)*, 11(3):259–286, April 2007.

[78] Roberto J. Bayardo Jr., Bart Goethals, and Mohammed Javeed Zaki, editors. *FIMI'04, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Brighton, UK, November 1, 2004*, volume 126 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2004.

[79] Leonid Khachiyan, Endre Boros, Khaled M. Elbassioni, and Vladimir Gurvich. Generating all minimal integral solutions to monotone $\wedge$, $\vee$-systems of linear, transversal and polymatroid inequalities. In *Proceedings of the Thirtieth International Symposium on Mathematical Foundations of Computer Science*, pages 556–567, Gdansk, Poland, August 2005.

[80] Ryutaro Kurai, Shin-Ichi Minato, and Thomas Zeugmann. N-gram analysis based on zero-suppressed BDDs. *New Frontiers in Artificial Intelligence*, 4384:289–300, 2007.

[81] Haiquan Li, Jinyan Li, Limsoon Wong, Mengling Feng, and Yap-Peng Tan. Relative risk and odds ratio: A data mining perspective. In *Proceedings of the Twenty-fourth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'05)*, pages 368–377, Baltimore, Maryland, USA, June 2005.

[82] Jinyan Li, Guozhu Dong, and Ramamohanarao Kotagiri. DeEPs: Instance-based classification by emerging patterns. In *Proceedings of the Fourth European Conference on Principles of Data Mining and Knowledge Discovery (PKDD'00)*, pages 191–200, Lyon, France, September 2000.

[83] Jinyan Li, Guimei Liu, and Limsoon Wong. Mining statistically important equivalence classes and delta-discriminative emerging patterns. In *Proceedings of the Thirteenth ACM SIGKDD International Conference on Knowledge Discovery in Databases (KDD'07)*, pages 430–439, San Jose, California, USA, August 2007.

[84] Jinyan Li, Huiqing Liu, James R. Downing, Allen Eng-Juh Yeoh, and Limsoon Wong. Simple rules underlying gene expression profiles of more than six subtypes of acute lymphoblastic leukemia (ALL) patients. *Bioinformatics*, 19:71–78, 2003.

[85] Jinyan Li, Huiqing Liu, and Limsoon Wong. Mean-entropy discretized features are effective for classifying high-dimensional biomedical data. In *Proceedings of the Third ACM SIGKDD Workshop on Data Mining in Bioinformatics (BIOKDD'03)*, pages 17–24, 2003. `http://datam.i2r.a-star.edu.sg/datasets/krbd/index.html`.

[86] Jinyan Li and Limsoon Wong. Emerging patterns and gene expression data. In *Proceedings of the Twelfth Workshop on Genome Informatics*, pages 3–13, December 2001.

[87] Jinyan Li and Limsoon Wong. Geography of differences between two classes of data. In *Proceedings the Sixth European Conference on Principles of Data Mining and Knowledge Discovery (PKDD'02)*, pages 325–337, Helsinki, Finland, August 2002.

[88] Jinyan Li and Limsoon Wong. Identifying good diagnostic gene groups from gene expression profiles using the concept of emerging patterns. *Bioinformatics*, 18(5):725–734, 2002.

[89] Jinyan Li and Qiang Yang. Strong compound-risk factors: Efficient discovery through emerging patterns and contrast sets. *IEEE Transactions on Information Technology in Biomedicine*, 11(5):544–552, 2007.

[90] Jiuyong Li. On optimal rule discovery. *IEEE Transactions on Knowledge and Data Engineering*, 18(4):460–471, 2006.

[91] Wenmin Li, Jiawei Han, and Jian Pei. CMAR: Accurate and efficient classification based on multiple class-association rules. In *Proceedings of the First IEEE International Conference on Data Mining (ICDM'01)*, pages 369–376, San Jose, California, USA, November 2001.

[92] Bing Liu, Wynne Hsu, and Yiming Ma. Integrating classification and association rule mining. In *Proceedings of the Fourth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'98)*, pages 80–86, New York, NY, USA, August 1998.

[93] Bing Liu, Liang ping Ku, and Wynne Hsu. Discovering interesting holes in data. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 930–935, Nagoya, Japan, August 1997.

[94] Guimei Liu, Hongjun Lu, Yabo Xu, and Jeffrey Xu Yu. Ascending frequency ordered prefix-tree: Efficient mining of frequent patterns. In *Proceedings of the Eighth International Conference on Database Systems for Advanced Applications*, pages 65–72, Kyoto, Japan, March 2003.

[95] Guimei Liu, Hongjun Lu, Jeffrey Xu Yu, Wei Wang, and Xiangye Xiao. AFOPT: An efficient implementation of pattern growth approach. In *Proceedings of the IEEE ICDM 2004 Workshop on Frequent Itemset Mining Implementations (FIMI'04)*, Brighton, UK, November 2004.

[96] Hongyan Liu, Jiawei Han, Dong Xin, and Zheng Shao. Top-down mining of interesting patterns from very high dimensional data. In *Proceedings of the Twenty-second International Conference on Data Engineering (ICDE'06)*, page 114, Atlanta, Georgia, USA, April 2006.

[97] Elsa Loekito and James Bailey. Fast mining of high dimensional expressive contrast patterns using ZBDDs. In *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'06)*, pages 307–316, Philadelphia, PA, USA, August 2006.

[98] Elsa Loekito and James Bailey. Are zero-suppressed binary decision diagrams good for mining frequent patterns in high dimensional datasets? In *Proceedings of the Sixth Australasian Data Mining Conference (AusDM'07)*, pages 139–150, Gold Coast, Queensland, Australia, December 2007.

[99] Claudio Lucchese, Salvatore Orlando, Paolo Palmerini, Raffaele Perego, and Fabrizio Silvestri. kDCI: a multi-strategy algorithm for mining frequent sets. In *Proceedings of the IEEE ICDM 2003 Workshop on Frequent Itemset Mining Implementations (FIMI'03)*, Melbourne, Florida, USA, December 2003.

[100] Claudio Lucchese, Salvatore Orlando, and Raffaele Perego. kDCI: on using direct count up to the third iteration. In *Proceedings of the IEEE ICDM 2004 Workshop on Frequent Itemset Mining Implementations (FIMI'04)*, Brighton, UK, November 2004.

[101] Qicheng Ma, J.T.L Wang, D. Sasha, and C.H. Wu. DNA sequence classification via an expectation maximization algorithm and neural networks: A case study. *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews*, 31(4):468–475, November 2001.

[102] Donna Maglott, Jim Ostell, Kim D. Pruitt, and Tatiana Tatusova. Entrez gene: gene-centered information at NCBI. *Nucleic Acids Research*, 35:26–31, January 2007. `http://www.ncbi.nlm.nih.gov/sites/entrez`.

[103] Sharad Malik, Albert R. Wang, Robert K. Brayton, and Alberto Sangiovanni Vintecelli. Logic verification using binary decision diagrams in a logic synthesis environment. In *Proceedings of the IEEE//ACM International Conference on Computer Aided Design (ICCAD'88)*, pages 6–9, Santa Clara, California, USA, November 1988.

[104] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Efficient algorithms for discovering association rules. In *AAAI Workshop on Knowledge Discovery in Databases*, pages 181–192, Seattle, Washington, USA, July 1994.

[105] Florent Masseglia, Fabienne Cathala, and Pascal Poncelet. The PSP approach for mining sequential patterns. In *Proceedings of the Second European Symposium on Principles of Data Mining and Knowledge Discovery (PKDD'98)*, pages 176–184, Nantes, France, September 1998.

[106] Shin-Ichi Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proceedings of the Thirtieth ACM/IEEE Design Automation Conference (DAC'93)*, pages 272–277, Dallas, Texas, USA, June 1993.

[107] Shin-Ichi Minato. *Binary Decision Diagrams and Applications for VLSI CAD*. Kluwer Publishers, Norwell, MA, USA, November 1996.

[108] Shin-Ichi Minato. Zero-suppressed BDDs and their applications. *International Journal on Software Tools for Technology Transfer (STTT)*, 3(2):156–170, 2001.

[109] Shin-Ichi Minato. Finding simple disjoint decompositions in frequent itemset data using zero-suppressed BDD. In *Proceedings of the IEEE ICDM 2005 Workshop on Computational Intelligence in Data Mining*, pages 3–11, Houston, Texas, USA, November 2005.

[110] Shin-Ichi Minato. A theoretical study on variable ordering of zero-suppressed BDDs for representing itemsets. In *Proceedings of the Tenth International Conference on Discovery Science (DS'07)*, pages 139–150, Sendai, Japan, October 2007.

[111] Shin-Ichi Minato and Hiroki Arimura. Efficient method of combinatorial item set analysis based on zero-suppressed BDDs. In *International Workshop on Challenges in Web Information Retrieval and Integration (WIRI'05)*, pages 3–10, Tokyo, Japan, April 2005.

[112] Shin-Ichi Minato and Hiroki Arimura. Frequent pattern mining and knowledge indexing based on zero-suppressed BDDs. In *The Fifth International Workshop on Knowledge Discovery in Inductive Databases (KDID'06)*, pages 152–169, Berlin, Germany, September 2006.

[113] Shin-Ichi Minato and Hiroki Arimura. ZBDD-growth: An efficient method for frequent pattern mining and knowledge indexing. TCS Technical Reports TCS-TR-A-06-12, Hokkaido University, Division of Computer Science, April 2006.

[114] Shin-Ichi Minato and Hiroki Arimura. Frequent closed item set mining based on zero-suppressed BDDs. *Transactions of the Japanese Society of Artificial Intelligence*, 2(1):309–316, February 2007.

[115] Shin-Ichi Minato, Nagisa Ishiura, and Shuzo Yajima. Shared binary decision diagrams with attributed edges for efficient boolean function manipulation. In

Proceedings of the Twenty-seventh ACM/IEEE Design Automation Conference (DAC'90), pages 52–57, Orlando, Florida, USA, June 1990.

[116] Shin-Ichi Minato and Kimihito Ito. Symmetric item set mining method using zero-suppressed BDDs and application to biological data. *Transactions of the Japanese Society of Artificial Intelligence*, 2(1):300–308, February 2007.

[117] Shin-Ichi Minato, Takeaki Uno, and Hiroki Arimura. LCM over ZBDDs: Fast generation of very large-scale frequent itemsets using a compact graph-based representation. In *Proceedings of the Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining (PAKDD'08)*, pages 234–246, Osaka, Japan, May 2008.

[118] Alan Mishchenko. An introduction to zero-suppressed binary decision diagrams. Technical reports, Portland State University, June 2001. `http://www.ee.pdx.edu/\~alanmi/research.htm`.

[119] Ieva Mitasiunaite and Jean-François Boulicaut. Looking for monotonicity properties of a similarity constraint on sequences. In *Proceedings of the ACM Symposium on Applied Computing*, pages 546–552, 2006.

[120] Tom M. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–226, 1982.

[121] Amit A. Nanavati, Krishna P. Chitrapura, Sachindra Joshi, and Raghu Krishnapuram. Mining generalised disjunctive association rules. In *Proceedings of the Tenth ACM Conference on Information and Knowledge Management (CIKM'01)*, pages 482–489, Atlanta, Georgia, USA, 2001.

[122] Arlindo L. Oliveira and Alberto L. Sangiovanni-Vincentelli. Inferring reduced ordered decision graphs of minimal description length. In *Proceedings of the Twelfth International Conference on Machine Learning (ICML'95)*, pages 421–429, Tahoe City, California, USA, July 1995.

[123] Jorn Ossowski and Christel Baier. Symbolic reasoning with weighted and normalized decision diagrams. In *Proceedings of the Twelfth Symposium on the Integration of Symbolic Computation and Mechanized Reasoning*, pages 35–96, March 2006.

[124] Feng Pan, Gao Cong, Anthony K. H. Tung, and Kian-Lee Tan. Mining frequent closed patterns in microarray data. In *Proceedings of the Fourth IEEE International Conference on Data Mining (ICDM'04)*, pages 363–366, Brighton, UK, November 2004.

[125] Feng Pan, Gao Cong, Anthony K. H. Tung, Jiong Yang, and Mohammed Javeed Zaki. CARPENTER: Finding closed patterns in long biological datasets. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03)*, pages 637–642, Washington, DC, USA, August 2003.

[126] Feng Pan, Wei Wang, Anthony K. H. Tung, and Jiong Yang. Finding representative set from massive data. In *Proceedings of the Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 338–345, Houston, Texas, USA, November 2005.

[127] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Discovering frequent closed itemsets for association rules. In *Proceedings of the Seventh International Conference on Database Theory (ICDT'99)*, pages 398–416, Jerusalem, Israel, January 1999.

[128] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Efficient mining of association rules using closed itemset lattices. *Information Systems*, 24:25–46, 1999.

[129] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Jianyong Wang, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. Mining sequential patterns by pattern-growth: The PrefixSpan approach. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1424–1440, November 2004.

[130] Jian Pei, Jiawei Han, Behzad Mortazavi-asl, and Hua Zhu. Mining access patterns efficiently from web logs. In *Proceedings of the Fourth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'00)*, pages 396–407, Kyoto, Japan, April 2000.

[131] Jian Pei, Jiawei Han, and Wei Want. Mining sequential patterns with constraints in large databases. In *Proceedings of the Eleventh ACM Conference on Information and Knowledge Management (CIKM'02)*, pages 18–25, McLean, Virginia, USA, October 2002.

[132] Andrea Pietracaprina and Dario Zandolin. Mining frequent itemsets using patricia tries. In *Proceedings of the IEEE ICDM'03 International Workshop on Frequent Itemset Mining Implementations (FIMI'03)*, Melbourne, Florida, USA, December 2003.

[133] John Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993.

[134] Antoine Rauzy. Mathematical foundations of minimal cutsets. *IEEE Transactions on Reliability*, 50(4):389–396, December 2001.

[135] Isidore Rigoutsos and Aris Floratos. Combinatorial pattern discovery in biological sequences: The TEIRESIAS algorithm. *Bioinformatics*, 14(1):55–67, 1998.

[136] François Rioult, Jean-François Boulicaut, Bruno Crémilleux, and Jérémy Besson. Using transposition for pattern discovery from microarray data. In *Proceedings of the Eighth ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'03)*, pages 73–79, San Diego, California, USA, June 2003.

[137] Utte Roessner, John H. Pattersonand Megan G. Forbesand Geoffrey B. Fincherand Peter Langridge, and Anthony Bacic. An investigation of boron toxicity in barley using metabolomics. *Plant Physiology*, 142(3):1087–1101, September 2006.

[138] Richard L. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of IEEE//ACM International Conference on Computer Aided Design (ICCAD'93)*, pages 42–47, Santa Clara, California, USA, November 1993.

[139] Amit Satsangi and Osmar R. Zaïane. Contrasting the contrast sets: An alternative approach. In *The Eleventh International Database Engineering and Applications Symposium (IDEAS)*, pages 114–119, September 2007.

[140] Bassem Sayrafi and Dirk Van Gucht. Differential constraints. In *Proceedings of the Twenty-fourth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'05)*, pages 348–357, Baltimore, Maryland, USA, June 2005.

[141] Christopher Scholl, Bernd Becker, and Andreas Brogle. The multiple variable order problem for binary decision diagrams: Theory and practical application.

In *Proceedings of the ASP-DAC 2001 Asia and South Pacific Design Automation Conference*, pages 85–90, Yokohama, Japan, January 2001.

[142] Michele Sebag. Delaying the choice of bias: A disjunctive version space approach. In *Proceedings of the Thirteenth International Conference on Machine Learning (ICML'96)*, pages 444–452, Bari, Italy, jun 1996.

[143] Rong She, Fei Chen, Ke Wang, Martin Ester, Jennifer L. Gardy, and Fiona S. L. Brinkman. Frequent-subsequence-based prediction of outer membrane proteins. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03)*, pages 436–445, Washington, DC, USA, aug 2003.

[144] Roslyn M. Sinnamon and John D. Andrews. Quantitative fault tree analysis using binary decision diagrams. *European Journal of Automation*, 30(8):1051–1071, 1996.

[145] Fabio Somenzi. CUDD: CU decision diagram package, 1997. Public software, Colorado University, Boulder.

[146] Arnaud Soulet, Bruno Crémilleux, and François Rioult. Condensed representation of emerging patterns. In *Proceedings of the Eighth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'04)*, pages 127–132, Sydney, Australia, May 2004.

[147] Ramakrishnan Srikant and Rakesh Agrawal. Mining quantitative association rules in large relational tables. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 1–12, Montreal, Quebec, Canada, June 1996.

[148] Radomir S. Stankovic and Tsutomu Sasao. Decision diagrams for discrete functions: Classification and unified interpretation. In *Proceedings of the ASP-DAC'98 Asia and South Pacific Design Automation Conference*, pages 439–446, Yokohama, Japan, February 1998.

[149] Seiichiro Tani, Kiyoharu Hamaguchi, and Shuzo Yajima. The complexity of optimal variable ordering problems of shared binary decision diagrams. In *Proceedings of the Fourth International Symposium on Algorithms and Computation (ISAAC '93)*, pages 389–398, London, UK, 1993. Springer-Verlag.

[150] Pawel Terlecki and Krzysztof Walczak. Jumping emerging patterns with negation in transactions databases - classification and discovery. *Information Sciences*, 177(24):5675–5690, December 2007.

[151] Takeaki Uno, Masashi Kiyomi, and Hiroki Arimura. LCM ver. 3: Collaboration of array, bitmap and prefix tree for frequent itemset mining. In *Proceedings of the First International Workshop on Open Source Data Mining (OSDM'05)*, pages 77–85, Chicago, Illinois, USA, 2005.

[152] Florian Verhein and Sanjay Chawla. Using significant, positively associated and relatively class correlated rules for associative classification of imbalanced datasets. In *Proceedings of the Seventh IEEE International Conference on Data Mining (ICDM'07)*, pages 679–684, Omaha, NE, USA, September 2007.

[153] Jilles Vreeken, Matthijs van Leeuwen, and Arno Siebes. Characterising the difference. In *Proceedings of the Thirteenth ACM SIGKDD International Conference on Knowledge Discovery in Databases (KDD'07)*, pages 765–774, San Jose, California, USA, August 2007.

[154] Sarma B. K. Vrudhula, Massoud Pedram, and Yung-Te Lai. Formal verification using edge-valued binary decision diagram. *IEEE Transactions on Computers*, 45(2):247–255, 1996.

[155] Hao Wang, Xing Zhang, and Guoqing Chen. Mining a complete set of both positive and negative association rules from large databases. In *Proceedings of the Twelfth Pacific-Asia Conference on Advances in Knowledge Discovery (PAKDD'08)*, pages 777–784, Osaka, Japan, May 2008.

[156] Jianyong Wang and Jiawei Han. BIDE: Efficient mining of frequent closed sequences. In *Proceedings of the Twentieth International Conference on Data Engineering (ICDE'04)*, page 79, Boston, MA, USA, mar 2004.

[157] Jianyong Wang, Jiawei Han, Ying Lu, and Petre Tzvetkov. TFP: An efficient algorithm for mining top-K frequent closed itemsets. *IEEE Transactions on Knowledge and Data Engineering*, 17(5):652–664, 2005.

[158] Jianyong Wang, Jiawei Han, and Jian Pei. CLOSET+: Searching for the best strategies for mining frequent closed itemsets. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03)*, pages 236–245, Washington, DC, USA, August 2003.

[159] Geoffrey I. Webb. Discovering significant rules. In *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'06)*, pages 434–443, Philadelphia, PA, USA, August 2006.

[160] Geoffrey I. Webb, Shane Butler, and Douglas Newlands. On detecting differences between groups. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03)*, pages 256–265, Washington, DC, USA, August 2003.

[161] Qiang Yang, Tianyi Li, and Ke Wang. Building association-rule based sequential classifiers for web-document prediction. *Data Mining and Knowledge Discovery*, 3(8):253–273, 2004.

[162] Xifeng Yang, Jiawei Han, and Ramin Afshar. CloSpan: Mining closed sequential patterns in large databases. In *Proceedings of the International Conference on Data Mining (SDM'03)*, pages 166–177, San Francisco, California, USA, November 2003.

[163] Xiaoxin Yin and Jiawei Han. CPAR: Classification based on predictive association rules. In *Proceedings of the Third SIAM International Conference on Data Mining (SDM'03)*, pages 331–335, San Fransisco, California, USA, April 2003.

[164] Osmar R. Zaïane and Maria-Luiza Antonie. Classifying text documents by associating terms with text categories. In *Proceedings of the Thirteenth Australasian Database Conference (ADC'02)*, Melbourne, Australia, January 2002.

[165] Osmar R. Zaïane and Mohammad El-Hajj. Pattern lattice traversal by selective jumps. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'05)*, Chicago, Illinois, USA, August 2005.

[166] Mohammed J. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1-2):31–60, 2001.

[167] Mohammed Javeed Zaki and Ching jui Hsiao. CHARM: an efficient algorithm for closed itemset mining. In *Proceedings of the Second SIAM International Conference on Data Mining (SDM'02)*, pages 457–473, Arlington, VA, USA, April 2002.

[168] Xiuzhen Zhang, Guozhu Dong, and Ramamohanarao Kotagiri. Exploring constraints to efficiently mine emerging patterns from large high-dimensional

datasets. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knolwedge Discvery and Data Mining (KDD'00)*, pages 310–314, Boston, MA, USA, August 2000.