Support-ordered trie for accelerating association rule mining, web usage mining and clustering

Woon, David Yew Kwong

2005

Woon, D. Y. K. (2005). Support-ordered trie for accelerating association rule mining, web usage mining and clustering. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/2532>

<https://doi.org/10.32657/10356/2532>

Nanyang Technological University

Downloaded on 26 Apr 2023 23:09:57 SGT

SUPPORT-ORDERED TRIE FOR ACCELERATING ASSOCIATION RULE MINING, WEB USAGE MINING AND CLUSTERING

WOON YEW KWONG DAVID

SCHOOL OF COMPUTER ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY

2005

Support-Ordered Trie for Accelerating Association Rule Mining, Web Usage Mining and Clustering

Woon Yew Kwong David

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in fulfilment of the requirement for the degree of
Doctor of Philosophy

2005

“The more you recognize and express gratitude for the things you have, the more things you will have to express gratitude for.”

ZIG ZIGLAR, Motivational Writer

Acknowledgements

First and foremost, I want to thank the *School of Computer Engineering* and the *Singapore Institute of Manufacturing Technology* (SIMTech) for granting me the opportunity and scholarship to undertake the Ph.D. (Computer Engineering) programme and for providing a most conducive research environment in the *Center for Advanced Information Systems* (CAIS) for my research activities.

As in any successful endeavor in life, it is unlikely that only one person should deserve all the credit. This thesis is no exception and I wish to thank and honor the following people who have contributed in one way or another, to the success of my research.

I want to first express my heartfelt thanks to my supervisor, Assoc. Prof. Ng Wee Keong, for his constant guidance and encouragement, especially during research *deadlocks* and *dry seasons*. Developing novel insights for a problem is both a tedious and uncertain process. One must work hard to compete globally with other researchers and there is no guarantee that the end result will be revolutionary. Without continued motivation and assurance, one could easily give up in the competitive arena of data mining research.

Next, I also need to thank the Head of the Information Systems Division, Assoc. Prof. Lim Ee Peng, for his advice and help in my work, and for organizing weekly meetings involving postgraduates in similar fields in CAIS for the sharing of findings and ideas. These meetings helped to broaden our knowledge as well as to generate fresh ideas.

My attachment at SIMTech was exciting because I had opportunities to participate in industrial projects as well as interact with people from the manufacturing industry.

My utmost appreciation to Dr Li Xiang and Dr Lu Wen Feng for making my stay in SIMTech a most pleasant and fruitful one!

Of course, I must not forget the unsung heroes of CAIS: the technical staff who worked hard to ensure the smooth operations of our computers, printers and network: Mr Lai Chee Keong and Ms Tan Lay Choo.

My fellow researchers in CAIS certainly deserve much credit too for their friendship, listening ears and sporadic invaluable inputs! A big thank-you to Ong Kok Leong, Sun Aixin, Liu Zehua, Wang Yida, Yin Ming, Khin Myo Win, Myo Myo Naing, Li Wenyuan, Li Zhao, Zhao Qiankun and Erwin Leonardi!

Last but not least, I must really thank God, my wife and my parents for their love, care, support and unwavering confidence in me. There were times when I neglected them in the course of my work and yet they continued to shower me with unconditional love. Their understanding was certainly pivotal in helping me to concentrate in my work without reservations.

In short, without the above key people, this work would never enjoy the current degree of international success and recognition.

Contents

Acknowledgements	iii
List of Figures	viii
List of Tables	ix
List of Algorithms	x
Summary	xii
1 Introduction	1
1.1 Background	1
1.1.1 Data Mining	1
1.1.2 Data Structures	4
1.2 Motivation	5
1.3 Scope	8
1.4 Contributions and Publications	9
1.5 Organization	10
2 Association Rule Mining	11
2.1 Problem Definition	14
2.2 Related Work	15
2.2.1 Algorithms for Speed	15
2.2.2 Algorithms with Novel Data Structures	16
2.2.3 Algorithms for Mining Variants of Association Rules	19
2.2.4 Summary	21
2.3 Data Structure: SOTrieIT	22
2.3.1 Complexity Analysis	26

2.3.2	Strengths and Weaknesses	27
2.4	Algorithms	27
2.5	Experiments	31
2.5.1	Static Databases	33
2.5.2	Dynamic Databases	36
2.5.3	Dynamic Support Threshold	37
2.5.4	Dynamic Universal Itemset	37
2.6	Applications	38
2.7	Summary	40
3	Web Usage Mining	41
3.1	Phases of Web Usage Mining	43
3.1.1	Phase 1: Preprocessing	44
3.1.2	Phase 2: Mining	44
3.1.3	Phase 3: Applying Mining Results	45
3.2	Related Work	46
3.3	Algorithms	52
3.3.1	Sessionization	52
3.3.2	Transactionization	53
3.3.3	Data Structure: WebTrie	56
3.3.4	Mining	59
3.3.5	Optimization Strategies	61
3.3.6	Experiments	63
3.4	Framework for Evaluating Web Usage Mining Algorithms	66
3.4.1	Measuring User Behavior	67
3.4.2	Describing Customer Behavior	69
3.4.3	Describing Web Site Structure	70
3.4.4	Describing Logfile Mining Algorithms	70
3.4.5	Suitability Analysis	72
3.5	Applications	73

3.6	Summary	77
4	Clustering	80
4.1	Problem Definition	81
4.2	Related Work	82
4.2.1	Clustering	82
4.2.2	Outlier Detection	88
4.2.3	Clustering and Association Rule Mining Mergers	90
4.3	Algorithms	91
4.3.1	Key Ideas	91
4.3.2	Algorithm	93
4.3.3	Complexity Analysis	103
4.3.4	Optimization Strategies	104
4.3.5	Evaluation Issues	108
4.3.6	Strengths and Weaknesses	109
4.4	Experiments	111
4.5	Applications	115
4.6	Summary	117
5	Conclusions	118
5.1	Thesis Summary	118
5.2	Future Research Frontiers	119
5.2.1	Association Rule Mining	119
5.2.2	Web Usage Mining	121
5.2.3	Clustering	122
5.3	Concluding Thoughts	122
	References	122

List of Figures

2.1	Resultant SOTrieITs after the arrival of four transactions.	25
2.2	Traversal path of the SOTrieIT at a support threshold of 75%.	30
2.3	Traversal path of the SOTrieIT at a support threshold of 80%.	30
2.4	Execution times and created FP-tree nodes for four synthetic databases at varying support thresholds.	32
2.5	Frequent itemsets found in four synthetic databases at varying support thresholds.	34
3.1	A weighted directed graph.	50
3.2	A WebTrie Example.	58
3.3	Execution times for $\mathcal{D}_1$ at varying minimum support thresholds. . . .	64
3.4	Execution times for $\mathcal{D}_2$ at varying minimum support thresholds. . . .	64
3.5	Execution times for $\mathcal{D}_3$ at varying minimum support thresholds. . . .	65
3.6	Number of frequent k -conses and k -pases in $\mathcal{D}_1, \mathcal{D}_2$ and $\mathcal{D}_3$ at a min- imum support threshold of 0.05%.	66
3.7	Product Development Collaborators	74
3.8	Product Development Processes	76
4.1	Example of Rogue cMaximal itemset regions.	103
4.2	A nSOTrieIT Example.	106
4.3	Real Dataset D_2	111
4.4	Results of executing CLUID on D_1 with $\psi = 50$	113
4.5	Results of executing DENCLUE on D_1 with varying σ, ξ, δ	114

4.6	Effect of ψ on D_1	114
4.7	Results of CLUID2 with $\psi = 600$ on D_3 visualized using multidimensional scaling.	116

List of Tables

2.1	Summary of existing ARM approaches.	21
2.2	A sample transactional database.	24
3.1	A sample clickstream database.	45
3.2	Summary of WUM approaches.	51
3.3	Extracted transactions from Table 3.1.	56
3.4	Real logfiles.	62
3.5	Transactionization and WebTrie Details.	62
3.6	Relationship of browsing and user characteristics.	68
3.7	Characteristics of various types of customers.	69
3.8	Adjustive Characteristics.	70
3.9	Description of logfile mining algorithms using WALMAGE.	70
4.1	Common parameters used in clustering	88
4.2	Characteristics of clustering algorithms	89
4.3	Example of mapping step.	96
4.4	Sample mapping table.	96
4.5	Discovered frequent itemsets.	101
4.6	Synthetic and real datasets used in the experiments.	111

List of Algorithms

1	Construction Algorithm of the SOTrieIT	23
2	Universal Itemset Update Algorithm for the SOTrieIT	24
3	FOLDARM Algorithm	28
4	FOLD-growth Algorithm	29
5	Transactionization Algorithm	55
6	Construction Algorithm of the WebTrie	57
7	Web Traversal Blueprint Discovery Algorithm	78
8	Web Traversal Blueprint Discovery 2 Algorithm	79
9	MapDB (CLUID Algorithm)	95
10	MineDB (CLUID Algorithm)	99
11	Construction Algorithm of the nSOTrieIT	105
12	MineDB2 (CLUID Algorithm)	107

“Out of clutter, find SIMPLICITY. From discord, find
HARMONY. In the middle of difficulty lies
OPPORTUNITY.”

ALBERT EINSTEIN (1879-1955) Physicist

Summary

Data mining becomes increasingly important with the advent of powerful data collection and storage tools; raw data is so abundant that it is no longer feasible to use statistics to manually analyze the data for useful patterns/trends. However, data mining problems are usually difficult to solve and this necessitates the use of novel data structures to improve efficiency. In this work, we have critically analyzed existing data structures used in association rule mining, an important and popular data mining technique. Association rule mining discovers correlations among items in a transactional database and is useful in many areas including electronic commerce, customer relationship management and bioinformatics. Our analyses culminate in a dual-level support-ordered trie structure called the *SOTrieIT* (Support-Ordered Trie Itemset).

The *SOTrieIT* greatly speeds up the discovery of frequent itemsets by storing ordered information about itemsets of lengths one and two. Further research revealed that web usage mining can also benefit from a data structure like the *SOTrieIT*. Hence, a variant of the *SOTrieIT*, called the *WebTrie*, was proposed to accelerate web usage mining. Together with appropriate segregation and mining algorithms, the *WebTrie* proved to be invaluable in the fast discovery of knowledge about the behavior of web users. With such knowledge, web sites can be improved and web page prefetching and caching can be carried out efficiently.

Clustering identifies natural groupings of objects without human supervision and is very useful in situations where there is little a priori knowledge. In bioinformatics, clustering has been shown to be exceptionally good at analyzing gene expression data and helping biologists understand the functions of unknown genes. By transforming

the problem of clustering to an itemset discovery problem, we are able to seamlessly plug in the power of association rule mining to speed up clustering. Using another variant of the SOTrieIT, called the *nSOTrieIT* (n-level SOTrieIT), we have developed an algorithm to rapidly discover clusters of varying sizes and shapes with minimal supervision amidst much noise.

Finally, extensive experiments involving a wide range of both synthetic and real datasets revealed that our techniques outperform (in terms of speed, scalability and quality) recent techniques (eg. FP-growth, RL method, DENCLUE) of the three data mining tasks by a significant margin.

“The distance is nothing; it’s only the first step that is difficult.”

MARQUISE DU DEFFAND (1697-1780) French
Woman of Letters

“The most important motive for work in school and in life is pleasure in work, pleasure in its result, and the knowledge of the value of the result to the community.”

ALBERT EINSTEIN (1879-1955) Physicist

1

Introduction

1.1 Background

1.1.1 Data Mining

Since the late 1980s, our abilities in the generation and collection of data have been greatly enhanced with the maturing of bar-coding, database, remote sensing and Internet technologies. As a result, manual inspection and analysis of raw data for patterns/trends are virtually impossible because of the sheer size of the datasets. This prompted keen interest in automated data analysis tools which in turn, catapulted the rise of one of the most important information technologies of the 1990s: DATA MINING.

Data mining is defined as *“the analysis of (often large) observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner.”* [76]. One school of thought defines data mining as only a step in the *Knowledge Discovery* (KDD) process while another defines both KDD and data mining to be synonymous because of the significance and complexity of the data mining step; the latter school of thought is adopted here. The following are the seven stages of KDD [74]:

Chapter 1. Introduction

1. *Data Cleaning*: Handling of missing data and removal of noise and inconsistent data
2. *Data Integration*: Merging of data from multiple heterogeneous sources into a coherent whole
3. *Data Selection*: Reducing size of data through aggregating, eliminating redundancy, reducing dimensions or sampling
4. *Data Transformation*: Transforming data into an appropriate form for mining by normalization, scaling or summary operations
5. *Data Mining*: Extracting patterns/rules from the data
6. *Pattern Evaluation*: Identifying interesting or important patterns/rules
7. *Knowledge Representation*: Presenting discovered knowledge through appropriate knowledge representation and visualization techniques

Many data mining techniques have been researched and applied in almost every imaginable scenario where data is abundant. The following are three main data mining techniques [74]:

1. *Association Rule Mining (ARM)*: ARM is the systematic discovery of groups of items in a transactional database that are correlated to one another. It is commonly known as *market basket analysis* because a good analogy of ARM is the analysis of items that are frequently put together in a basket by shoppers in a market. It is widely used in a myriad of applications including recommender systems [101, 100], promotional bundling [149], customer relationship management (CRM) [58] and cross-selling [26, 27]. In addition, its concepts/ideas are also integrated into other mining tasks for increased efficiency and effectiveness: web usage mining [156], clustering [164, 151], classification [54] and text mining [83, 16].
2. *Clustering*: Clustering or cluster analysis is the process of grouping data into previously unknown clusters (classes) so that objects within the same cluster

Chapter 1. Introduction

are similar according to some specified similarity measure. It is considered an unsupervised technique because users need not supply class-labelled data for training. It is important in data mining in general because it can discover hidden patterns with little human supervision. Some applications include customer segmentation [58], spatial data mining [118], web usage mining [143], manufacturing [65], personalization of web pages [111], and digital libraries [144].

3. *Classification*: Classification is the process of deciding which predefined classes new objects should be assigned to. Unlike clustering, the classification process requires human intervention and is thus a supervised technique. Users must supply a classifier with training data so that the classifier can generalize from the data and understand how new data should be classified. Its major applications include web document categorization [109], clinical diagnosis [163], pattern recognition [56] and security informatics [145] .

As the *World Wide Web* (Web) matures and rises in popularity and importance, the above techniques are adapted to work with web data, giving birth to three main web mining tasks [104]:

1. *Web Content Mining*: Web content mining is the analysis of data contents found on the Web, which includes text, images, audio, video and documents. It can be employed for information retrieval [7] (finding information from digital document collections) as well as web data modelling for facilitating database queries [61].
2. *Web Usage Mining (WUM)*: Web servers store lots of potentially useful data in the form of web access logs where every single HTTP request is recorded religiously. WUM analyzes such logs to sieve out patterns of user behavior that is valuable for CRM [58] (eg. customer retention, sales boosting), web personalization [112, 110] (customizing web experience of a user according to his/her interest/taste), web site structuring [106, 139, 141] (eg. grouping related pages together), marketing [29], user profiling [111, 77] (for better web personalization), caching and prefetching [162] (for faster navigation).

3. *Web Structure Mining*: Web structure mining is concerned about the underlying model of both the inter-page and intra-page link structures in the Web. Such a model is useful for the study of social networks [33] (study of connectivity and distances in graphs for identifying authoritative pages and for citation indexing etc.), web site categorization [34] (for better structured search) as well as for locating important authoritative and good summary web sites [97].

Finally, bioinformatics is also a fast emerging field because of recent advances in large-scale genome analysis, microarray and storage technologies [12]. Bioinformatics leverages on mathematics and information technology to advance biological research. At present, there is increasing use of data mining techniques to analyze complex biological data [19]. Both ARM and clustering have been used to study proteins and gene expression data [135, 15, 57, 146].

1.1.2 Data Structures

Data structures first appeared when programming became increasingly complex during the 1960s. With his classic book, *The Art of Computer Programming*, Knuth masterfully reviewed and analyzed all major algorithms and data structures that are essential for program efficiency [90]. Since then, the traditional data structures have been extended and new algorithms introduced to manipulate and traverse them. Though computing power has increased tremendously over the years, efficient algorithms with customized data structures are still necessary to obtain timely and accurate results. This is specially true for data mining which is a computationally-intensive process. Data mining problems are either *NP-Complete* or require exponential time to complete [9, 47]. Hence, in data mining, data structures play important roles such as storing pertinent data in memory and accelerating data access in order to ensure the timely delivery of results.

Several novel data structures have been introduced to improve the efficiency of data mining tasks but these data structures are created in an ad-hoc manner for specific problems. There is none to date that can be used effectively in a range of data mining problems. One highly promising candidate is the *trie* data structure

Chapter 1. Introduction

because of its compactness as well as the possibility of designing fast search algorithms for it [63]. The trie data structure is also our choice of focus in this work as it is widely applicable and yields tremendous improvement in speed when appropriately customized and used in many data mining problems.

A trie is a k -ary tree whose organization is based on a *key space* decomposition. In key space decomposition, the key range is equally subdivided and the splitting position within the key range for each node is predefined. The following are three types of tries [91]:

1. *Alphabet Trie*: The *alphabet trie* stores a dictionary of words and thus its nodes are labelled with alphabetical keys. It allows words to be quickly searched and conserves space because words with the same prefixes share the same nodes. The inspiration for our successful trie structure is drawn from the alphabet trie.
2. *Binary Trie*: The *binary trie* is introduced by Coffman *et al.* [42]. It is similar to an alphabet trie except that its keys are binary. Hence, the bits 0 or 1 is used to decide if the left or right branch is to be taken at each traversal step.
3. *Patricia Trie*: The *Patricia* (Practical Algorithm To Retrieve Information Coded in Alphanumeric) trie was proposed by Morrison to compress binary tries by avoiding one-way branches [113]. This is accomplished by including in each node, the number of bits to skip over before making the next branching decision. The Patricia Trie is the most compact of the three tries and is the fastest access method of all. However, it is very difficult to update and thus unsuitable for fast-changing data.

1.2 Motivation

The driving force for this thesis stems from the increasingly urgent need for *speed*, *scalability* and *reusability* (SSR) of three important data mining techniques that are highly relevant in today's hot areas of business and/or biology: ARM, WUM and clustering. WUM may be considered an application area but as it requires special and complex techniques to adapt classic data mining techniques like ARM, clustering

Chapter 1. Introduction

and classification, to its unique needs, we shall henceforth term it as a data mining technique in view of the nontrivial challenges it presents. We chose to focus on these 3 mining tasks because they all deal with data that is very similar in nature: streaming in fast, changing rapidly and increasing in size and complexity. Such data has recently been termed *data streams* [78, 11]. Other important data mining techniques like classification and spatial mining are not considered due to a lack of time and that they cannot derive benefits directly from our proposed data structures.

We reckon intuitively that these mining problems will certainly benefit tremendously from a generic data structure that can help bring about SSR with customized algorithms. Speed is necessary to give timely results while scalability is essential as data streams are unbounded in size. Reusability promotes speed and scalability by reusing past knowledge. In addition, the mining tasks have one thing in common: the need to discover patterns, trends or natural groupings with little or no prior knowledge. This implies that it is necessary to adjust parameters with many runs of the mining algorithm so as to obtain good results. Here are examples of scenarios of the three mining techniques that motivate this work:

- *An Association Rule Mining Scenario:* Consider an online store like AMAZON.COM, which sells thousands of items 24 hours a day globally. Many customers can arrive concurrently and thus, its transactional database grows very quickly. Such a store would certainly benefit much from ARM as ARM can identify items that are frequently purchased together and hence, suggest items to the customers based on the current items in their shopping cart. However, to be effective, ARM must be performed fast on the current state of the database, which is probably very huge. Moreover, as the database is constantly changing, it is impossible to determine suitable support and confidence thresholds a priori. Therefore, it is necessary to run the ARM algorithms a few times at various thresholds to yield the desired number of rules. In short, without algorithms with excellent SSR, it is difficult for online stores to recommend items to their customers in real time.

Chapter 1. Introduction

- *A Web Usage Mining Scenario:* One of the main success factors of web sites is fast page loading. Nobody would want to visit a web site where pages take minutes to load. WUM is useful here as it can discover frequently-accessed page sequences from the web server logs and this knowledge can be used to perform page prefetching and caching to reduce/eliminate page loading time. Like the previous scenario, it is not possible to set thresholds to remove uninteresting or infrequent sequences before the mining process. Coupled with the fact that server logs are usually very huge, it is clear that WUM algorithms need to have good SSR characteristics in order to take into consideration the latest data in the prediction of pages for prefetching or caching.
- *A Clustering Scenario:* A recent and important use of clustering is in the analysis of gene expression data, which is critical in the fight against cancer. By grouping genes with similar expression level trends, it is possible to identify cancer causing/inhibiting genes, which will eventually help in the accurate diagnosis/prognosis of cancer. However, gene expression data has very high dimensionality (number of genes) in the order of thousands and most clustering algorithms cannot cope with such dimensionality efficiently. Gene expression data is also steadily growing in size because of the ease of conducting experiments using advanced microarray technologies. Hence, SSR is certainly highly desired here though it is equally important to develop new clustering techniques that are more attuned to the idiosyncrasies of gene expression data. This work addresses the SSR issues only.

There does not exist any single data structure that can fulfill all the above SSR requirements; some allow fast access but are not reusable (eg. FP-tree [75], adjacency lattice [2]) while others are scalable but cannot be quickly updated with new data (eg. binary Patricia trie [161]). Hence, we decided to embark on a three-year quest for such a data structure that not only works well in these three tasks, but also possibly in other mining tasks.

1.3 Scope

Each of the three data mining techniques that we have chosen to focus on, has a scope broad enough for a Ph.D. thesis. In addition, the data stream phenomenon is a recent one with several nontrivial challenges due to its four main characteristics [11]:

1. Its data elements arrive online.
2. The order of arrival of its data elements cannot be controlled.
3. It is potentially unbounded in size.
4. Once its data element is processed, it is discarded — only a small number of elements are stored in memory.

Hence, it is necessary for us to choose to tackle specific issues in each of the three data mining techniques and to address only the first three characteristics (SSR issues) of the data stream model. There is a host of information on mining data streams in recent prominent conferences [35, 64, 60].

The following problems are the focus points of this thesis:

- *Association Rule Mining*: Since 1993, ARM has received intense attention from the research community. Most researchers focus on improving its SSR and this would be our focus as well. Recent issues like mining for evolving rules in data streams, mining variants of association rules and mining with only a single database scan are not addressed here.
- *Web Usage Mining*: WUM takes on an important role as the Web becomes a popular portal for conducting business, research and product development. We not only address SSR issues here but also propose new ways of segregating web server logs so as to obtain meaningful and usable knowledge. However, we only deal with traditional HTML web pages as other dynamically-generated web pages are not explicitly captured in logfiles and hence could not be easily analyzed. We have also proposed a framework for assessing the suitability of WUM algorithms for specific scenarios. However, before the framework can be

deployed, it still requires the integration of cognitive science concepts, which is beyond the scope of this work.

- *Clustering*: For over three decades, researchers have been searching for the ultimate clustering method that is truly unsupervised (no input parameters needed) and can discover clusters of arbitrary shapes and sizes from a huge dataset amidst much random noise. Till today, the search has not ended but we have helped make significant progress in this search by enhancing the SSR of clustering and designing algorithms to discover arbitrary clusters with minimal human intervention. The current issue of adapting clustering techniques to discover evolving clusters as well as to work with limited memory in data streams are not addressed here.

1.4 Contributions and Publications

This work resulted in eleven papers (ten accepted, one under review) over the past three years. The following is a succinct summary of the contributions of this thesis:

- The study of ARM was previously undertaken during the author's undergraduate candidature in his final-year project. In this thesis, the breadth and depth of the prior work was expanded with enhanced theoretical and empirical studies. We firstly studied the problem of ARM in depth and noted its limitations [154]. We then designed a trie-like data structure called the *SOTrieIT* (**S**upport **O**rdered **T**rie **I**Temset) and accompanying algorithms to speed up ARM [51, 152]. Our work in ARM culminate with an algorithm that can perform faster than a prominent existing algorithm by up to two orders of magnitude and this technique is published in the IEEE journal *Transactions on Knowledge and Data Engineering* [157] and the *Encyclopedia of Data Warehousing and Mining* [158] (see Chapter 2).
- After understanding the issues involving WUM, we proposed a new way of segregating web server logs as well as a variant of the SOTrieIT, called the

WebTrie [156]. We also realized that a framework for comparing various WUM algorithms is needed so as to guide the choice of algorithms for specific scenarios. Hence, we have created a basic framework called *WALMAGE* (**W**eb **A**ccess **L**og **M**ining **A**l**G**orithm **E**valuator) for this purpose [155]. With the proliferation of web-based product development solutions, we further examined and applied WUM in the product design context [153]. Our comprehensive work in WUM is published as a book chapter in *Web Mining: Applications and Techniques* [159] (see Chapter 3).

- After analyzing the plethora of clustering methods, we first designed a noise-filtering technique called *FLUID* (**F**i**L**tering **U**sing **I**temset **D**iscovery) [150]. Next, based on the ideas of FLUID, we moved on to develop a clustering algorithm called *CLUID* (**C**Lustering **U**sing **I**temset **D**iscovery). CLUID transforms the original dataset into a transactional database so that the power of ARM can be tapped to enhance the SSR of clustering as well as to discover clusters with arbitrary shapes and sizes with just one input parameter. We have also introduced another variant of the *SOTrieIT*, called the *nSOTrieIT* (*n*-level *SOTrieIT*) to work together with an optimized algorithm to further stretch the SSR limits of clustering. Our work on CLUID has been submitted to the IEEE journal *Transactions on Knowledge and Data Engineering* [151] (see Chapter 4).

1.5 Organization

This thesis is organized into five chapters. Chapters 2, 3 and 4 are self-contained studies of association rule mining, web usage mining and clustering respectively. In the three chapters, existing works are critically reviewed, possible solutions are proposed and empirical results shown to verify the viability of using our novel trie structures with their customized algorithms. Some useful applications for the three mining tasks are also discussed at the end of each chapter. Finally, Chapter 5 concludes the thesis and recommends exciting uncharted areas for future research.

“Inspirations never go in for long engagements; they demand immediate marriage to action.”

BRENDAN FRANCIS (1923-1964) Writer

“Restlessness is discontent, and discontent is the first necessity of progress.”

THOMAS EDISON (1847-1931) Inventor

2

Association Rule Mining

Association Rule Mining (ARM) discovers associations/correlations among items in a database of transactions. Obviously and naturally, ARM is often associated with the analysis of sales data. Indeed, ARM works well with sales data and it was the business domain that brought ARM into the limelight in its early years [154]. However, its applicability and popularity in other domains are also undisputed now with many prominent ARM works in data mining tasks such as classification [102] and clustering [95] as well as application areas like web mining [156] and bioinformatics [135]. As electronic commerce is the main impetus for the general advancement of ARM techniques as well as the basic motivation for this work, it will be discussed here in detail. Electronic commerce data has some characteristics of data streams and thus has challenging SSR requirements for ARM algorithms. However, as mentioned in the previous chapter, not all unique challenges of data streams are addressed; the restriction on memory usage and constraint of mining only with a single database scan will be addressed in our future work.

The importance of electronic commerce is beyond doubt now as online transactions become increasingly secure and more widely accepted by the general public. In a recent Forrester Research article, *Global eCommerce Approaches Hypergrowth*, it is

Chapter 2. Association Rule Mining

estimated that global Internet trade would reach US\$6.8 trillion in 2004, amounting to 8.6% of the global sales of goods and services [133]. In addition, according to an Ernst & Young survey, by 2005 online shopping could account for 10% to 12% of sales of clothing, accessories and toys, 20% to 25% of sales of books, music, software and consumer electronics [165]. Hence, it is no longer an option but a requirement for companies to realign their business strategies with the Internet. In electronic commerce, thousands of transactions can easily take place in a single day. Hidden within these data is valuable knowledge about the behavior of customers which could be unraveled with data mining techniques. Such knowledge is of utmost importance to companies in a highly-competitive business arena like the Internet. To know and understand the customer is the key to winning customer loyalty, which leads to market dominance. ARM examines the buying habits of customers and helps to decide how products are to be laid out in a virtual store or how promotions are to be bundled. In addition, it can be used to improve sales by suggesting additional products for the customer to purchase.

In electronic commerce where a huge number of transactions can arrive at a virtual store from all over the world 24 hours a day, transaction databases are expected to be updated frequently. Thus, existing algorithms that are designed to mine static databases (databases that are not expected to change) will not be able to perform well with databases that are constantly changing. There are some algorithms that can perform incremental mining, which means that they can improve mining speed by reusing past mined information [39, 40, 134]. However, such algorithms cannot cope with databases with many frequent updates because these algorithms capitalize on the fact that databases do not usually change much (less than 40%) and that many of the previously mined results are still valid (see more details in Section 2.2.1).

In addition, in a highly competitive setting like electronic commerce, companies will need to constantly introduce new products and remove unpopular products to satisfy the increasingly demanding needs of the now empowered customer. This means that, in most cases, the number and types of unique items in the database will change very often. For example, an online bookstore would constantly need

Chapter 2. Association Rule Mining

to update its database with the latest available books and this would increase the number of unique items. Moreover, since the books are new items, there would be new types of unique items. Another good example is an online movie ticketing system where available movies for booking change very frequently. Unfortunately, existing algorithms assume that unique items are fixed and thus, each time items are added or removed, the algorithms must mine the database from scratch and discard valuable past mined results [6, 122, 39, 40, 134, 166, 2, 80, 8, 161, 75].

Finally, given the dynamism and volatility of transactional data in electronic commerce, companies cannot predict a suitable support threshold to set for the mining process. Using too high a threshold may result in too many unimportant rules while too low a threshold may result in certain important rules being passed over. Therefore, there is a need to mine the database with several different support thresholds before an *optimal* threshold (suitable threshold that yields association rules of good quality and quantity for competitive intelligence) can be determined. This critical need has not been effectively tackled by current algorithms.

Several data structures have been proposed to make ARM more efficient but there is not one that can meet *ALL* of the above needs [166, 2, 80, 8, 161, 75] (see Section 2.2.4). Here, we shall critically examine existing preprocessing data structures used in ARM in an attempt to understand their strengths and weaknesses. Our analyses culminate in a practical structure called the *SOTrieIT* (**S**upport-**O**rdered **T**rie **I**Temset—pronounced as “so try it”) and three synergistic association rule mining algorithms to accompany it. Experiments involving a wide range of synthetic datasets reveal that the algorithms outperform *FP-growth* [75], a recent association rule mining algorithm with excellent performance, by up to two orders of magnitude and thus, verifying its efficiency and viability.

The rest of the chapter is organized as follows. The next section defines the ARM problem formally. Section 2.2 surveys and scrutinizes existing ARM algorithms. Our novel data structure is presented in Section 2.3 while algorithms to complement it are proposed in Section 2.4. Section 2.5 shows experiments involving a variety of synthetic datasets. Useful ARM applications are discussed in Section 2.6 and finally,

the chapter is summarized in Section 2.7.

2.1 Problem Definition

The following is a formal statement of the problem of mining association rules [6]:

- Let the *universal itemset*, $I = \{a_1, a_2, \dots, a_U\}$ be a set of literals called *items* and $|I|$ be its size.
- Let D_t be a database of transactions, where each transaction T contains a set of items such that $T \subseteq I$ and $|D_t|$ be its size.
- A k -itemset is an unordered set of k unique items.
- For a given *itemset* $X \subseteq I$ and a given transaction T , T contains X if and only if $X \subseteq T$.
- Let σ_x be the *support count* of an itemset X , which is the number of transactions in D_t that contain X .
- Let s be the *support threshold*.
- An itemset X is *large* or *frequent* if $\sigma_X \geq |D_t| \times s\%$.
- A candidate itemset is an itemset that is potentially a frequent itemset.
- Let a set of candidate k -itemsets and frequent k -itemsets be C_k and L_k respectively.
- An *association rule* is an implication of the form $X \implies Y$, where $X \subseteq I$, $Y \subseteq I$ and $X \cap Y = \emptyset$.
- The association rule $X \implies Y$ holds in the database D_t with *confidence* $c\%$ if no less than $c\%$ of the transactions in D_t that contain X also contain Y .
- The association rule $X \implies Y$ has *support* $s\%$ in D_t if $\sigma_{X \cup Y} \geq |D_t| \times s\%$.

For a given pair of confidence and support thresholds, the problem of mining association rules is to discover all rules that have confidence and support greater than the corresponding thresholds. For example, in a computer hardware shop, the

association rule $CD_Writer \implies Lens_Cleaner$ means that whenever customers buy CD writers, they also buy lens cleaners at least $c\%$ of the time and this trend occurs at least $s\%$ of the time. The ARM problem can be decomposed into two sub-problems [6]:

1. Discovery of frequent itemsets
2. Generation of association rules from frequent itemsets

Researchers usually tackle the first sub-problem only because it is more computationally expensive and less straightforward [74]. Hence, most algorithms are designed to efficiently discover frequent itemsets.

2.2 Related Work

A myriad of ARM works is reviewed here. For clarity, this section is divided into three parts. The first part deals with algorithms that are designed for speed. The second shows works on mining variants of association rules and the last focuses on techniques that rely on customized data structures.

2.2.1 Algorithms for Speed

The Apriori algorithm is the first successful algorithm for mining association rules [6]. Since its introduction, it has popularized the task of mining association rules and sparked off many research papers to improve mining efficiency. It introduces a method to generate candidate itemsets C_k in a pass k using only frequent itemsets L_{k-1} in the previous pass. The idea rests on the fact that any subset of a frequent itemset must be frequent as well. This idea is known as the *downward closure* or *apriori* property. Hence, C_k can be generated by joining L_{k-1} and deleting those that contain any subset that is not large. By using *only* the previous frequent itemsets to generate candidate itemsets, Apriori generates much fewer candidate itemsets than its predecessors, the AIS [5] and SETM [84] algorithms, which generate candidate itemsets by extending previous frequent itemsets with items in the transactions.

Chapter 2. Association Rule Mining

The *Direct Hashing and Pruning* (DHP) algorithm is another efficient algorithm for the mining of association rules [122]. It employs a hash technique to reduce the size of candidate itemsets and the database. This amounts to significant speed-ups because the dominating factor in the generation of frequent itemsets using the apriori property is the size of the candidate itemsets. DHP has significant speed improvements over Apriori due to the reduced size of the candidate itemsets generated. However, it incurs additional overheads because of the need to do hashing and maintain a hash table. After some experiments, it is concluded that the hash technique should only be applied during the generation of candidate 2-itemsets to achieve speed-ups of up to three times against Apriori.

The *Fast Update* (FUP) algorithm is an incremental algorithm which makes use of past mining results to speed up the mining process [39]. Its successor, the *Fast Update Two* (FUP2) algorithm, is a generalization of it [40]. By setting bounds for the support counts of candidate itemsets, it is able to reduce the size of C_k and improve its efficiency. By re-using past mining information, FUP2 reduces the number of candidate sets and hence achieves a speed improvement of up to three times over Apriori. However, when the size of the updates exceeds 40% of the original database, Apriori performs better. Incremental mining is brought to another new level when the Adaptive algorithm is introduced [134]. This algorithm is not only incremental but also *adaptive* in nature; by inferring the nature of the incremental database, it adapts itself for improved performance by avoiding unnecessary database scans. Experiments have shown that it can perform up to seven times faster than Apriori in a database of about 70K transactions and in incremental databases with up to 10K transactions.

2.2.2 Algorithms with Novel Data Structures

The use of *lattice theory* [52] was pioneered by Zaki *et al.* [166]. Lattice theory allows the vast search space to be decomposed into smaller segments that can be tackled independently in memory or even in other machines and thus promotes parallelism. To complement the use of lattices, Zaki uses a vertical database format where each itemset is associated with a list of transactions known as a *tid-list* (transac-

Chapter 2. Association Rule Mining

tion identifier-list). This format is useful for fast frequency counting of itemsets but generates additional overheads because most databases have a horizontal format and would need to be first converted. Six new algorithms are introduced and the best one, *MaxClique*, performs more than 300 times faster than Apriori at support thresholds of 0.75% and lower.

Unlike all the discussed algorithms, the *Continuous Association Rule Mining Algorithm* (CARMA) allows the user to change the support threshold and continuously displays the resulting association rules with support and confidence bounds during the first scan/phase [80]. During the second phase, it determines the precise support of each itemset and extracts out all the frequent itemsets. With the support lattice, CARMA can readily compute frequent itemsets for varying support thresholds. However, experiments reveal that CARMA only performs faster than Apriori at support thresholds of 0.25% and below because of the tremendous overheads involved in constructing the support lattice.

The adjacency lattice was introduced by Aggarwal *et al.* [2, 3]. It is similar to Zaki's boolean powerset lattice except that the authors introduced the notion of adjacency among itemsets and it does not rely on a vertical database format. Two itemsets are said to be *adjacent* to each other if one of them can be transformed to the other with the addition of a single item. To address the problem of heavy memory requirements, the authors defined a *primary threshold* which is the minimum support threshold possible to fit all qualified itemsets into the adjacency lattice in main memory. However, this approach disallows the mining of frequent itemsets at support thresholds lower than the primary threshold.

Amir *et al.* presented a completely new way of mining association rules using a trie to preprocess the database [8]. This method is the first to eliminate the need for candidate itemset generation, which is the main bottleneck in all Apriori-based algorithms. Here are its basic ideas:

1. All transactions are mapped onto a trie structure. This mapping involves the extraction of the powerset of the transaction items and the updating of the trie

Chapter 2. Association Rule Mining

structure.

2. Once built, there is no longer a need to scan the database to obtain support counts of itemsets because the trie structure contains all their support counts.
3. To find frequent itemsets, the structure is traversed using depth-first search and itemsets with support counts satisfying the minimum support threshold are added to the set of frequent itemsets.

However, this method requires a great deal of processing time and memory space to construct the trie and therefore has limited scalability, especially for databases with long transactions.

Drawing upon the work of Amir *et al.*, Yang *et al.* introduced a binary Patricia trie to reduce the heavy memory requirements of the preprocessing trie [161]. To support faster support queries, the authors also added a set of horizontal pointers to index nodes. Yang *et al.* also advocated the use of some form of primary threshold to further prune the structure. As experiments are not conducted to compare their algorithm with existing ones, we do not have a feel of its performance. Intuitively, we can appreciate the immense space savings reaped by the Patricia trie. However, this compression comes at a hefty price: it greatly complicates the horizontal pointer index and this is a severe overhead. In addition, after compression, it will be difficult for the Patricia trie to be updated whenever the database is altered. Hence, this approach works well only for *static databases* (databases that do not change with time) with a small universal itemset.

The *Frequent Pattern-growth* (FP-growth) algorithm is a recent association rule mining algorithm that achieves impressive results [75]. It uses a compact tree structure called a *Frequent Pattern-tree* (FP-tree) to store information about large 1-itemsets. This compact structure removes the need for multiple database scans and it is constructed using only 2 scans. In the first database scan, large 1-itemsets L_1 are obtained and sorted in support descending order. In the second scan, items in the transactions are first sorted according to the order of L_1 . These sorted items are used to construct the FP-tree. FP-growth then proceeds to recursively mine FP-trees

Chapter 2. Association Rule Mining

of decreasing size to generate frequent itemsets without candidate generation and database scans. It does so by examining all the *conditional pattern bases* of the FP-tree, which consists of the set of frequent itemsets occurring with the suffix pattern. Conditional FP-trees are constructed from these conditional pattern bases and mining is carried out recursively with such trees to discover frequent itemsets of various sizes. However, since the construction and use of the FP-trees are complex, the performance of FP-growth is reduced to be on par with Apriori at support thresholds of 3% and above for datasets with up to 100K transactions with an average of 25 items per transaction [75]. It only achieves significant speed-ups at support thresholds of 1.5% and below. Moreover, it is only incremental to a certain extent depending on the FP-tree *watermark* (validity support threshold). As new transactions arrive, the support counts of items increase but their relative support frequency may decrease too. But suppose the new transactions cause too many previously infrequent itemsets to become frequent, i.e., the watermark is raised too high (in order to make such itemsets infrequent) according to a user-defined level, then the FP-tree must be reconstructed.

Kuramochi *et al.* proposed the use of graphs to model database objects in an attempt to allow ARM to be applied in non-transactional domains more easily [99]. With such a model, ARM can be reduced to simply finding frequent subgraphs in a set of graphs regardless of its domain. However, as this technique involves the need to solve NP-Complete problems of *canonical labelling* and *graph isomorphism discovery*, it is very slow and will not work for large databases with large universal itemsets.

2.2.3 Algorithms for Mining Variants of Association Rules

Besides the above algorithms for improving mining speed and flexibility, there are several algorithms that are designed for mining variants of association rules in a multitude of situations. The mining of multi-level associations rules is useful in situations where concept hierarchies [72] are available [73]. By extracting rules for data at different levels of abstractions, more interesting rules can be found. For example, the association rule *Iomega_CD_Writer* $\implies$ *3M_Lens_Cleaner* is more informative than

Chapter 2. Association Rule Mining

CD_Writer $\implies$ *Lens_Cleaner* because it is at a lower concept level and is thus more specific.

Traditional statistics is integrated into ARM in where the classic *chi-squared* test is used to test for correlations between itemsets [28]; this allows additional pruning to be done on top of the pruning achieved by support and confidence thresholds because rules must contain correlated itemsets. Quantitative association rules are introduced because quantitative variables are very common and interesting rules can be derived from them [142]. However, quantitative variables are usually partitioned into fixed intervals which may result in some rules being overlooked. This problem is addressed with fuzzy logic where variables are assigned by membership functions [98].

As databases and universal itemsets grow, the number of frequent itemsets grows exponentially. Hence, the idea of only extracting *maximal itemsets* (frequent itemsets that are not subsets of any other itemsets) is being explored [167, 14]. This results in major speed-ups but at the expense of completeness because the support counts of all frequent itemsets are not available and the database must be scanned again to derive the final association rules. The idea of mining only *frequent closed itemsets* (frequent itemsets that do not have supersets with equal support counts) is proposed as a better alternative to mining maximal itemsets [123, 124]. Its advantage lies in its ability to generate complete association rules without any loss of information and the number of frequent closed itemsets generated is between that of frequent and maximal itemsets. However, its performance suffers when the database contains weakly-correlated and sparse data.

Recently, Seno *et al.* argued that having a constant support threshold in FP-growth will result in too many small itemsets while obtaining only a few large-sized itemsets, especially at lower thresholds [136]. To overcome this problem, the authors suggested using support thresholds that decrease as a function of the size of an itemset; a smaller itemset must have a much higher support count than a larger itemset to qualify as frequent. With this new support constraint, the authors designed a new algorithm called *LPMiner* that is faster than FP-growth (see next section) by up to two orders of magnitude. This speed-up is achieved by using the support constraint

Chapter 2. Association Rule Mining

Algorithm	Maximum speed-up factor against Apriori	Special Features
Apriori	1	Nil
DHP	3	Use of hash tables
FUP	7	Incremental
FUP2	3.4	Generalized FUP
Adaptive	7	Incremental and adaptive
MaxClique	300	Use of lattices
CARMA	2.5	Ability to handle multiple support thresholds online
FP-growth	10	Use of FP-trees
FOLD-growth	1000	Use of SOTrieIT; incremental, dynamic and handles mutiple support thresholds

Table 2.1: Summary of existing ARM approaches.

to prune transactions, FP-tree nodes and FP-tree paths. This approach may not be practical because it is difficult to determine a suitable function to compute the optimal support thresholds.

2.2.4 Summary

Table 2.1 summarizes the various features of existing approaches for mining traditional association rules (those meant for mining variants of association rules are excluded) and compares it with our flagship algorithm, FOLD-growth (to be discussed in the next section). To address the SSR requirements of mining fast-changing data for association rules, ARM algorithms must have the following features:

- *Incremental*: Being incremental means there is no need to restart mining from scratch as information from past mining runs can be exploited for maximum speed gains. This is especially true for large datasets like data streams.
- *Dynamic*: Being dynamic in this context means the ability to accommodate changes in the universal itemset without the need to restart mining from scratch. The universal itemset changes frequently in today's datasets, especially those of electronic commerce sites where products become obsolete very quickly.

Chapter 2. Association Rule Mining

- *Handles multiple support thresholds:* For fast-changing data, it would take a few runs to determine a suitable support threshold. Algorithms without this feature would need to restart mining from scratch when the support threshold is changed.

Note that there is not one algorithm that has all the above features except FOLD-growth. In addition, FOLD-growth far outperforms all existing approaches in terms of speed.

All in all, the discussed ARM algorithms and data structures are not capable of addressing the SSR requirements of mining today's fast-changing datasets effectively. In the next section, we shall unveil our novel trie data structure that can be used together with customized algorithms to meet such requirements.

2.3 Data Structure: SOTrieIT

Here, we introduce the *SOTrieIT* (**Support-Ordered Trie I**temset), a dual-level support-ordered trie data structure, which is used to store pertinent itemset information to speed up the discovery of frequent itemsets.

Definition 1 (SOTrieIT) *A SOTrieIT consists of two levels of tree nodes such that every tree node w has a label ℓ (represents an item) and an integer j (represents support count). (As every tree node corresponds to some item $a_i \in I$, for brevity, we use w_i to refer to a tree node that corresponds to $a_i \in I$.) Let $C(w_i)$ be the set of child nodes of node w_i and $S(w_i)$ be the support count of an itemset represented by w_i . If w_i is a first-level node, $S(w_i)$ is the support count of 1-itemset $\{a_i\}$. If w_i is a second-level node, $S(w_i)$ is the support count of 2-itemset $\{a_h, a_i\}$ where w_h is its parent node. If $C(w_i) \neq \emptyset$, then $C(w_i) \subseteq \{w_j, \dots, w_N\}$ where $j > i \wedge S(w_j) \geq S(w_{j+1})$, i.e., the child nodes are sorted by decreasing support counts. ■*

A set of SOTrieITs, possibly rooted at $w_1, w_2, \dots, w_N$, is built from a database to store support counts of all 1-itemsets and 2-itemsets. We use a special node called *ROOT* to link all these SOTrieITs together and keep them ordered by support count

Chapter 2. Association Rule Mining

Algorithm 1 Construction Algorithm of the SOTrieITInput: A database transaction T and a SOTrieIT Y Output: An updated Y

```

1: for ( $k = 1; k \leq 2; k++$ ) do
2:   Obtain all  $k$ -itemsets of  $T$  and store them in  $C_k$ , a set of candidate  $k$ -itemsets
3:   for each itemset  $X \in C_k$  do
4:     Traverse  $Y$  to locate nodes  $N$  along the path that represents  $X$ 
5:     if  $N \in Y$  then
6:       Increment or decrement support count of the leaf node depending on the nature
       of update
7:       if its support count falls to 0 then
8:         Remove node and its child nodes (if any)
9:       end if
10:      Sort the updated node according to its new support count in descending order
11:    else
12:      Create a new set of nodes with support counts of 1 that represent a path to  $X$ 
13:      Insert nodes into  $Y$  according to their support counts in descending order from
      the left
14:    end if
15:  end for
16: end for

```

in a way similar to their second-level nodes. For brevity, we shall refer to a set of SOTrieITs as simply SOTrieIT henceforth.

Algorithm 1 shows the steps taken to update the SOTrieIT whenever a transaction is added or deleted. Note that these steps are carried before actual mining is carried out and they can be viewed as preprocessing steps. For every transaction that arrives, 1-itemsets and 2-itemsets are first extracted from it (step 2). For each itemset, the SOTrieIT will be traversed in order to locate the node that stores its support count (step 4). Support counts of 1-itemsets and 2-itemsets are stored in first-level and second-level nodes respectively. The traversal of the SOTrieIT thus requires at most two redirections, which makes it very fast. At any point in time, the SOTrieIT contains the support counts of all 1-itemsets and 2-itemsets that appear in all the transactions. It will then be sorted level-wise from left to right according to the

Chapter 2. Association Rule Mining

Algorithm 2 Universal Itemset Update Algorithm for the SOTrieITInput: An item i and a SOTrieIT Y Output: An updated Y

- 1: **if** item i is added to the universal itemset **then**
- 2: Do nothing because the SOTrieIT will be updated the moment a transaction with i arrives
- 3: **else if** item i is removed from the universal itemset **then**
- 4: Traverse the SOTrieIT to remove all nodes and their child nodes (if any) that contain i
- 5: **end if**

TID	Items
100	AC
200	BC
300	ABC
400	$ABCD$

Table 2.2: A sample transactional database.

support counts of the nodes in descending order to enable fast pruning during mining (see Section 2.4). The sorting can be done easily (if necessary) because it involves shifting a single node (the node that has just been updated) left or right depending on its new support count. If such a node does not exist, it will be created and inserted into the SOTrieIT accordingly. Similarly, the SOTrieIT is then sorted after such an insertion. For deletions, the steps are similar except that the support counts of the affected nodes are decremented and nodes are deleted if their support counts fall to zero.

Algorithm 2 shows how the SOTrieIT is updated when the universal itemset is changed. In algorithms like FP-growth that use a similar data structure to store itemset information, the structure must be rebuilt to accommodate updates to the universal itemset. In this case, the SOTrieIT can be easily updated to accommodate the new changes.

Example Figures 2.1(a), (b), (c) and (d) show how the SOTrieIT is constructed from the four transactions of a sample database in Table 2.2. The bracketed number

Chapter 2. Association Rule Mining

Figure 2.1: Resultant SOTrieITs after the arrival of four transactions.

beside a node's label denotes the support count. Note that its nodes are support-ordered and has 2 levels of nodes (excluding the special ROOT node). To illustrate how nodes are created, let us examine what happens when a new transaction arrives. Note that only 1-itemsets and 2-itemsets are extracted from the transactions. When transaction 100 arrives, only three itemsets, $\{A\}$, $\{C\}$, $\{A, C\}$, are extracted and three nodes are created to hold their support counts, which have a value of one at this moment as seen in Figure 2.1(a). When transaction 200 arrives, the nodes created are shown in Figure 2.1(b). Notice that in Figure 2.1(b), the node w_C under the ROOT node comes before the node w_A . This is because the nodes are sorted according to their support counts and w_C has a higher support count than w_A . When transaction 300 arrives, the following itemsets are extracted and used to update the SOTrieIT:

$$\{A\}, \{B\}, \{C\}, \{A, B\}, \{A, C\}, \{B, C\}$$

Figure 2.1(c) shows the resultant SOTrieIT when this transaction is processed. When transaction 400 arrives, the following itemsets are extracted:

$$\{A\}, \{B\}, \{C\}, \{D\}, \{A, B\}, \{A, C\}, \{A, D\}, \{B, C\}, \{B, D\}, \{C, D\}$$

The SOTrieITs are updated in a similar fashion for transaction 400 as seen in Figure 2.1(d).

2.3.1 Complexity Analysis

In this section, the time complexity of constructing the SOTrieIT and space complexity of storing it are analyzed. These measurements determine the scalability of the SOTrieIT with respect to the size of the database as well as the size of the universal itemset.

The amount of time to preprocess a single transaction is the time needed to extract 1-itemsets and 2-itemsets from the transaction, traverse the SOTrieIT to increment the support counts of the respective nodes, and create new nodes in the SOTrieIT for items that are not encountered yet. For a transaction of size l , only $({}^lC_1 + {}^lC_2)$ itemsets are preprocessed. As the SOTrieIT is only two levels deep, it takes at most two links to reach the desired node. Suppose it also takes one unit of time to move over one link, it will take a maximum of $2 \times ({}^lC_1 + {}^lC_2)$ units of time to move to all the nodes required by a transaction of size l . For a database D_t , it takes a total of $|D_t| \times 2 \times ({}^lC_1 + {}^lC_2)$ units of time. Hence, the time complexity of constructing the SOTrieIT is $O(l^2)$.

In a database with a universal itemset of size N , there will be N first-level nodes in the SOTrieIT. For each first-level node, since the SOTrieIT is created in a trie-like manner, it will contain only items that are lexicographically larger than itself. The first-level node which has the largest number of child nodes is the one that has the first position in a set of lexicons. It will have $N - 1$ child nodes. Subsequent first-level nodes will have one fewer child node than the previous one. Therefore, for N unique items, a maximum of only $\sum_{x=1}^N x$ nodes, inclusive of both first-level and second-level nodes, are needed to store the entire preprocessing information. Hence, its space complexity is $O(N^2)$.

2.3.2 Strengths and Weaknesses

Unlike the trie structure of Amir *et al.*, the SOTrieIT is ordered by support count (speeds up mining) and does not require the powersets of transactions (reduces construction time) [8]. The main weakness of the SOTrieIT is that it can only discover L_1 and L_2 while its main strength lies in its speed in discovering L_1 and L_2 . Both L_1 and L_2 can be found promptly because there is no need to scan the database. In addition, the search (depth-first) can be stopped at a particular level the moment a node representing a non-frequent itemset is found because the nodes are all support-ordered.

Another advantage of the SOTrieIT, compared with all previously-discussed structures, is that it can be constructed *online*, meaning that each time a new transaction arrives, the SOTrieIT can be incrementally updated. This is possible because the SOTrieIT is constructed without the need to know the support threshold; it is support-independent. All 1-itemsets and 2-itemsets in the database are used to update the SOTrieIT regardless of their support counts. To conserve storage space, existing trie structures have to use thresholds to keep their sizes manageable and thus, when new transactions arrive, they have to be re-constructed because the support counts of itemsets would have changed.

Finally, the SOTrieIT requires far less storage space than a trie or Patricia trie because it is only two levels deep and can be easily stored in both memory and files. The number of level 2 nodes $\mathcal{N}_2$ far exceeds that of level 1 nodes $\mathcal{N}_1$ and thus, level 2 nodes are stored in files. Although this cause some I/O overheads, it is insignificant as seen in the experiments in Section 2.5. $\mathcal{N}_1$ is in the order of the size of the universal itemset $|I|$ while $\mathcal{N}_2$ is $\sum_{i=1}^{|I|} i$.

2.4 Algorithms

In this section, three algorithms are proposed to work hand in hand with the SOTrieIT to enable the fast discovery of frequent itemsets.

The first algorithm is termed *FOLDARM* (**F**ast **O**nLine **D**ynamic **A**ssociation

Chapter 2. Association Rule Mining

Algorithm 3 FOLDARM AlgorithmInput: A database D , a support threshold s and a SOTrieIT Y

Output: A set of frequent itemsets

```

1: Let  $N_q^p$  be the  $q^{th}$  child node of parent node  $p$  in  $Y$ 
2: Let  $NC_p$  be number of child nodes under  $p$ 
3: Let  $I_n$  be the itemset represented by node  $n$ 
4: for ( $x=1$ ;  $x \leq NC_{ROOT}$ ;  $x++$ ) do
5:   Let  $X = N_x^{ROOT}$ 
6:   if  $\sigma_X \geq |D| \times s\%$  then
7:     Add  $I_X$  to  $L_1$ 
8:     for ( $y=1$ ;  $y \leq NC_X$ ;  $y++$ ) do
9:       if  $\sigma_{N_y^X} \geq |D| \times s\%$  then
10:        Add  $I_{N_y^X}$  to  $L_2$ 
11:       end if
12:     end for
13:   end if
14: end for
15: Run Apriori from its third iteration to find the rest of the frequent itemsets

```

Rule Mining); it is *fast* because it leverages on the speed of the SOTrieIT; it is *online* because users can vary the support threshold while maintaining the same performance; it is *dynamic* because the universal itemset, can be changed and be easily accommodated in the SOTrieIT. Algorithm 3 shows the steps of FOLDARM. The SOTrieIT Y is first traversed to discover frequent 1-itemsets L_1 and 2-itemsets L_2 . In our approach, depth-first search is used, starting from the leftmost first-level node. As Y is sorted according to support counts, the traversal can be stopped the moment a node is found not to satisfy the minimum support threshold. After L_1 and L_2 are found, FOLDARM proceeds to discover other larger frequent itemsets using the Apriori algorithm. Since the generation of candidate 2-itemsets has been shown to be the main bottleneck in Apriori's candidate generation strategy, FOLDARM is able to achieve much performance improvement over Apriori [152].

The second algorithm is called *FOLD-growth* (**F**ast **O**nLine **D**ynamic-growth), which is an enhanced hybrid version of FOLDARM and FP-growth [75]. Past ex-

Chapter 2. Association Rule Mining

Algorithm 4 FOLD-growth Algorithm

Input: A database D , a support threshold s and a SOTrieIT Y

Output: A set of frequent itemsets

- 1: Use the Y to quickly discover L_1 and L_2
 - 2: **if** $L_1 = \emptyset \vee L_2 = \emptyset$ **then**
 - 3: Terminate algorithm
 - 4: **end if**
 - 5: **for** transaction $T \in D$ **do**
 - 6: Remove items that will not contribute to L_k , where $k > 2$, using L_1 and L_2
 - 7: Sort items in support descending order
 - 8: Construct/Update the FP-tree with trimmed and sorted T
 - 9: **end for**
 - 10: Run FP-growth algorithm on constructed FP-tree
-

periments reveal that FOLDARM is extremely fast when the size k_{max} of the largest frequent itemset is small (≤ 10) while FP-growth excels in situations where $k_{max} > 10$ and thus, FOLD-growth is an attempt to amalgamate their strengths [152].

Algorithm 4 shows how FOLD-growth utilizes the SOTrieIT. With the SOTrieIT, L_1 and L_2 can be quickly found and they can be used to prune the transactions that are used to construct the FP-tree (step 6). Hence, only one database scan is needed to start building the FP-tree. Note that if L_2 is not found, we can terminate the algorithm immediately because all possibly frequent itemsets, in this case L_1 , are already found (step 3). Another point to note is that there is no need to store the trimmed transactions because once they are used to update the FP-tree, they can be discarded. The SOTrieIT can be easily and incrementally updated when transactions are added or deleted but the FP-tree must always be reconstructed whenever there are updates to the database because of its dependence on support thresholds; an update to the database affects the relative frequency of items, causing them to become frequent or infrequent. Therefore, since FOLD-growth uses SOTrieIT, it can be said to be more incremental than FP-growth to a certain extent even though FOLD-growth itself is not incremental.

We apply such pruning to the FOLDARM algorithm to create our third algorithm:

Chapter 2. Association Rule Mining

Figure 2.2: Traversal path of the SOTrieIT at a support threshold of 75%.

Figure 2.3: Traversal path of the SOTrieIT at a support threshold of 80%.

FOLDARM2. FOLDARM2 is similar to FOLDARM except for an additional step to reduce the database size before applying Apriori. Like FOLD-growth, L_1 and L_2 are used to prune transactions so that only items that may contribute to the support count of frequent itemsets of size three and above remain. As Apriori scans the database iteratively, this reduction of database size translates into huge savings in database scanning time.

Example To illustrate how the SOTrieIT is efficiently traversed to obtain the support counts of L_1 and L_2 , we use the same transaction database found in Table 2.2 and the SOTrieIT structure in Figure 2.1(d). Suppose the support threshold is set at 75%. Then the minimum support count to qualify an itemset to be large is 3. Figure 2.2 shows the traversal path taken in obtaining L_1 and L_2 . The bold numbers on the arrows denote the sequence with which the SOTrieIT is traversed. During the generation of the first two frequent itemsets, the moment a first-level node with a support count lower than 3 is encountered, the rest of its siblings and subtrees are not scanned. But when a second-level node is found not to have satisfied the minimum support count, only its subsequent siblings will be ignored. In this case, at the fifth traversal, when the node that represents itemset $\{A, B\}$ is found to have a support

Chapter 2. Association Rule Mining

count of less than 3, the node that represents itemset $\{A, D\}$ will not be explored. The final frequent 1-itemsets and 2-itemsets found are $L_1 = \{\{A\}, \{B\}, \{C\}\}$ and $L_2 = \{\{A, C\}, \{B, C\}\}$ and the total number of traversals is 9 (out of a maximum 10). The scenario changes favorably when the support threshold is increased slightly. The next example demonstrates more clearly the advantage of ordering the SOTrieIT by the support counts of nodes.

For a minimum support threshold of 80%, the minimum support count needed is 4. Figure 2.3 shows the traversal path taken in obtaining the large 1-itemsets and 2-itemsets. Traversal stops at the third step when the node containing item A , which has a support count of 3, is found. This is because all other nodes that come after the first-level node A will have a support count of 3 or less. Therefore, there will not be any more frequent itemsets in the rest of the SOTrieIT. The algorithm terminates after only 3 traversals and the only frequent itemset is $\{C\}$.

The above examples illustrates the usefulness of the SOTrieIT: once constructed, it can be used for varying support thresholds. In addition, in a best case scenario where the minimum support threshold is high, it may only need one traversal to discover all frequent itemsets while in a worst case scenario, it may only need a number of traversals whose cost is definitely lesser than that of scanning a large database.

2.5 Experiments

This section evaluates and compares the relative performance of FP-growth, FOLD-growth and FOLDARM2 by conducting experiments on a Pentium-IV machine with a CPU clock rate of 2.4 GHz and 1 GB of main memory. The section is divided into four sections where the applicability of the algorithms is considered in four scenarios: scenarios involving static and dynamic databases as well as scenarios requiring support for dynamic support thresholds and dynamic universal itemsets. All algorithms are implemented in Java and the implementations of FP-growth and the synthetic database generator are taken from *ARMiner* [120]. To describe a database, we use the notation $Tw.Ix.Ny.Dz$ where w is the average size of transactions, x is the average

Chapter 2. Association Rule Mining

Figure 2.4: Execution times and created FP-tree nodes for four synthetic databases at varying support thresholds.

size of maximal potentially frequent itemsets, y is the number of unique items and z is the size of the database.

2.5.1 Static Databases

A static database here is defined as a database that does not change frequently. The databases used here are T25.I20.N10K.D100K (11 MB), T2.I2.N32K.D640K (10 MB), T25.I20.N75.D1K (0.1 MB) and T25.I20.N32K.D640K (73 MB) denoted as $\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_3$ and $\mathcal{D}_4$ respectively. The first database is similar to the one used by Han *et al.* [75]. The second database $\mathcal{D}_2$ is based on a survey done by the *National Association of Recording Merchandisers* which discovered that, on average, customers do not buy more than two items in a transaction [121]. Since online stores like **Amazon.com** usually carry millions of products, we want to use a universal itemset that is as large as possible but due to memory limitation, we only use 32,000 unique items for $\mathcal{D}_2$ and $\mathcal{D}_4$. The third database $\mathcal{D}_3$ is modeled after survey-based datasets. It is the *densest* database here as about a third of the universal itemset are in each transaction; a dense database here refers to a database where transactions contain a substantial ($\geq 10\%$) portion of the universal itemset. Finally, $\mathcal{D}_4$ is the largest database and is used to test the scalability of the algorithms. Figure 2.4 shows the processing time and number of FP-tree nodes needed for the databases while Figure 2.5 displays the number of frequent itemsets found. Note that in some of the graphs, the computation times of FOLDARM2 are not plotted as it exceeds 1,000 s.

The preprocessing step to construct the SOTrieIT is not reflected in the graphs because the SOTrieIT is incrementally constructed/updated for each transaction as it arrives; the construction phase takes place over a long period of time before the actual mining process and the SOTrieIT is ready for use in mining at any one point in time. Hence, the effect of incremental updates on the data is not mentioned since the focus here is the performance comparison of algorithms in the mining phase. An average of only 1 s, 4 ms, 15 ms and 1.25 s are spent on preprocessing a single transaction found in $\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_3$ and $\mathcal{D}_4$ respectively. In addition, since it is built only once and re-used many times (during the initial adjustment of support thresholds), its construction

Chapter 2. Association Rule Mining

Figure 2.5: Frequent itemsets found in four synthetic databases at varying support thresholds.

cost, which is spread out over a period of time, is trivial due to amortization. However, the same cannot be said for the FP-tree because it is support-dependent and must thus be constructed from scratch for different support thresholds. One interesting idea would be to construct an FP-tree with a support threshold of 0%. However, experiments reveal that much computation time (up to 370 s for $\mathcal{D}_4$) and storage space (more than 9 million nodes for $\mathcal{D}_4$) are needed for such a FP-tree. Unlike the SOTrieIT, its construction time must be considered because it cannot be built incrementally over time.

It is clear from Figure 2.4 that FOLD-growth outperforms FP-growth by a huge margin under all circumstances except in Figure 2.4(D1A) where it performs only slightly faster than FP-growth for low support thresholds and in Figure 2.4(D3A) where it exhibits similar performance as FP-growth. This can be explained by Figure 2.5(D1) and (D3) where the length k_{max} of the biggest frequent itemset increases

Chapter 2. Association Rule Mining

exponentially as the support threshold decreases. With a higher k_{max} value, the major overhead of the FP-growth approach shifts from the creation of FP-tree nodes to the recursive mining of conditional pattern trees and this nullifies FOLD-growth's early pruning feature, which is its key edge against FP-growth. In Figure 2.4(D1B) and (D3B), it can be observed that k_{max} also affects the number of FP-tree nodes that is being reduced by FOLD-growth through the pruning of the database using the SOTrieIT. In short, FOLD-growth and FP-growth have similar performance in dense databases like $\mathcal{D}_1$ and $\mathcal{D}_3$ where k_{max} is high (≥ 10).

Both FOLD-growth and FOLDARM2 perform exceptionally well for higher support thresholds. Notice that in many cases, there is a sudden drop in execution time for both algorithms, eg. Figures 2.4(D2A) and (D4A) at support thresholds of 0.25% and 2% respectively. This is because all the frequent itemsets have at most 2 items and thus can be discovered simply by traversing the SOTrieIT without additional mining steps. This allows the user to quickly try out a range of values for the support threshold (to find a value which yield good results) with minimal waste of time during the initial exploratory mining operations. For example, in $\mathcal{D}_1$, suppose the user tries out support thresholds of 5%, 4%, 3% and 2%. It takes FP-growth a total of about 140s for the 4 mining operations while FOLD-growth takes only 4s.

For FOLD-growth, another important reason for its good performance at high support thresholds is found in Figures 2.4(D1B), (D2B), (D3B) and (D4B) where we can see that the number of FP-tree nodes created at higher support thresholds is minimal. There is no need to create so many nodes as the SOTrieIT allows L_1 and L_2 to be discovered without much computation and without database scans. On the other hand, regardless of the support threshold, FP-growth needs two database scans; one to discover L_1 and another to sort and prune transactions according to L_1 . FOLD-growth only needs to scan a smaller version of the database (due to the additional pruning of transactions using L_2) only once when L_2 exists in order to construct the FP-tree. This characteristic of FOLD-growth is particularly useful in large databases as observed in Figure 2.4(D4A) where FOLD-growth outperforms FP-growth by two orders of magnitude at support thresholds of 2% and above. Hence, FOLD-growth

is much more scalable than FP-growth. However, FOLDARM2 performs badly for low support thresholds because it relies on Apriori that needs to generate many more candidate itemsets when k_{max} is large. In conclusion, FOLD-growth is the best algorithm here.

2.5.2 Dynamic Databases

A dynamic database is one where transactions are added and removed frequently. The FUP2 algorithm is proposed to make use of past mining results to mine new transactional updates more efficiently [40]. Due to time constraints, FUP2 is not implemented for comparison studies with FOLD-growth. Fortunately, with the results in its performance analysis section, we can easily assess its performance as compared to FOLD-growth with the performance of Apriori as our reference point.

In a database of the type T10.I4.N1K.D100K with an addition and deletion of 5,000 transactions, it is found that FUP2 performs only about twice as fast as Apriori. This is its best performance against Apriori through the use of past mined knowledge. Both Apriori and FOLD-growth mine a dynamic database as if it were a static database because they do not make use of past results. Since FOLD-growth outperforms FP-growth by up to two orders of magnitude and FP-growth outperforms Apriori by up to an order of magnitude, it can be concluded that FOLD-growth outperforms Apriori by at least two orders of magnitude, far greater than what FUP2 can achieve. Hence, FOLD-growth clearly outperforms FUP2.

Explanation FUP2 performs poorly against FOLD-growth mainly because it is fundamentally based on the Apriori approach. In fact, it is a generalization of Apriori [40]. It outperforms Apriori due to its ability to reuse mined results and reduce the number of candidate itemsets generated. However, it has limited scalability because of the need to generate candidate itemsets (see Section 2.2.1). Moreover, in situations when the updates are high, the performance of FUP2 drops because the updated database becomes so different from the original one that past mining results are not helpful in determining new frequent itemsets and computation time is wasted on

processing useless old information.

FOLD-growth maintains its good performance for dynamic databases because it always mines the database from scratch regardless of the amount of updates. The main difference here as compared to static databases is that the SOTrieIT is updated more often and this does not affect the mining phase.

2.5.3 Dynamic Support Threshold

CARMA is currently the only algorithm that allows the user to modify the support threshold on the fly. We will once again use the results presented in its experiment section as a form of comparison [80]. In a database of the form T10.I4.N10K.D100K, it is found that Apriori outperforms CARMA for support thresholds of 0.5% and above. It is only when the support thresholds are at 0.25% and below that CARMA reaps the rewards of its approach to outperform Apriori by about 1.5 times. As seen in the previous section, FOLD-growth outperforms Apriori by at least two orders of magnitude and thus obviously outperforms CARMA as well.

Explanation The poor performance of CARMA is attributed to its need to maintain a lattice of potentially frequent itemsets to mine at different support thresholds. It will be faster only when the user does not need a precise set of frequent itemsets because in this case, CARMA does not need to re-scan the database. FOLD-growth performs much faster because the SOTrieIT stores threshold-independent information and thus its performance will not be affected even if the user changes the support thresholds frequently to obtain an optimal threshold.

2.5.4 Dynamic Universal Itemset

As none of the existing algorithms discussed takes into consideration of the fact that unique items in the database will vary, all the discussed algorithms would have to mine from scratch a database whose universal itemset has been modified. Therefore, their performance against FOLD-growth in such a scenario can be deduced from the previous sections. When the universal itemset is changed, FOLD-growth's SOTrieIT

can be easily updated using Algorithm 2. Hence, FOLD-growth retains its performance edge with minimal additional operations.

2.6 Applications

In this section, some successful ARM applications are presented to demonstrate the wide applicability and usefulness of ARM. These applications will benefit tremendously from FOLD-growth because of the fast-changing and unbounded nature of their data; electronic commerce data changes rapidly due to intense competition and the changing needs of customers; text databases grow and change as documents increase and their relationships become more interlinked and complex in the information age; biological databases are changing rapidly as knowledge about newly-understood/discovered genes and proteins becomes readily available through advances in biological techniques.

Together with collaborative filtering, ARM has been used to analyze shopping carts and preferences of e-shoppers [101]. To determine associations among shoppers, a product is mapped as a transaction with shoppers that like it as the items. To determine associations among products, a shopper is mapped as a transaction with products that he/she likes as the items. Therefore, after the mappings, it is possible to understand shoppers better by running ARM algorithms to determine the above-mentioned associations. In addition, promotional bundling can be carried out for specific customers with similar biodata or buying habits using an approach called *Profit Mining* [149]. Given a collection of past sales transactions, profit mining recommends certain products that would generate as much profit as possible on certain target products over future transactions.

Customer relationship management (CRM) also benefits greatly from ARM as it helps to understand customer behavior [58]. Marketing managers can use association rules of products to develop joint-marketing and cross-selling campaigns to acquire new customers. The application of ARM for the cross-selling of supermarket products has been successfully attempted in many cases [26, 27]. In one particular study

Chapter 2. Association Rule Mining

involving the personalization of supermarket product recommendations, ARM has been applied with much success [100]. ARM is used to determine relationships among products. Together with customer clustering, ARM achieved an increase of 1.8% in revenue.

ARM has recently been adapted for text databases by Holt *et al.* [83]. The authors mention two possible uses of ARM in text mining:

1. *Statistical Thesaurus*: ARM is useful in building up a statistical thesaurus where words that are frequently associated with one another are stored to improve document searches.
2. *Latent Semantic Indexing (LSI)*: LSI is an approach to retrieve documents that do not have any terms in common with the query expression by appending query results with additional documents that are *similar* to the documents in the original query results [67]. ARM can assist in finding such similar documents by mining association rules with words as transactional items in the documents.

In the biology domain, ARM is used to discover associations concerned with the amino acid sequence, structure and function of proteins [135]. It well-known that the above three aspects of proteins are closely related and ARM can be used to discover their associations. Such association rules are useful in predicting protein structure and designing proteins with desired functions. In this study, the discovered association rules match biological knowledge and this implies the applicability of ARM in future protein research.

In a study involving the analysis of gene expression data, ARM has shown potential as well [15]. ARM was applied to a freely available human serial analysis of gene expression data. The association rules revealed that a very strong co-regulation of mRNA encoding ribosomal proteins occurred in the dataset. Several rules associating proteins involved in the signal transduction were analyzed and a wrongly labeled tag was discovered and reassigned. In addition, the authors were able to propose a function for an expressed sequence tag encoding a protein with unknown function.

2.7 Summary

In this chapter, we have examined the problem of mining association and analyzed its diversified algorithms. We have designed a novel trie data structure called the SOTrieIT and used it effectively to achieve immense speed-ups as verified by rigorous experiments. In fact, our FOLD-growth algorithm outperforms a prominent algorithm, the FP-growth, by up to two orders of magnitude. Finally, practical applications of ARM are highlighted to illustrate its versatility and viability as well as the applicability of FOLD-growth. In the next two chapters, the usefulness of ARM is further exemplified as we show how ARM concepts can be integrated into WUM and clustering to enhance the SSR of their algorithms.

“The time for extracting a lesson from history is ever at hand for those who are wise.”

DEMOSTHENES (384-322 B.C.) Orator

“Nine-tenths of wisdom is being wise in time.”

THEODORE ROOSEVELT (1858-1919) 26th U.S.
President

3

Web Usage Mining

The rate of growth of the *World Wide Web* (Web) may be slowing down but *Online Computer Library Center* researchers concluded that the Web would continue to grow rapidly in their annual review of the Web [53]. In addition, Forrester Research affirms the continued popularity of electronic commerce through its prediction that global online trade will expand to \$12.8 trillion by 2006 [137]. To stay competitive and profitable in a fast-paced environment like the Web, online companies must be able to exploit and extract customer-related knowledge from their web server access logs for the successful implementation of *Customer Relationship Management* (CRM) [18]. In addition, much valuable knowledge about the trends and patterns of product development processes can be found by applying data mining techniques on logfiles because of the growing popularity of online product development solutions [153] (see Section 3.5). Such knowledge is *hidden* and *previously-unknown* and thus may well become the keen competitive edge that manufacturers are urgently seeking. This knowledge can be put to great use by streamlining the product development processes or simply improve the speed and convenience of web accesses. These motivating examples bring us to the focus of this chapter: *Web Usage Mining* (WUM)

We define WUM as the *extraction of meaningful user behavioral patterns from web*

Chapter 3. Web Usage Mining

server access logs using data mining techniques. WUM is primarily interested in the web user and seeks to understand him/her in a way that will contribute to business competitiveness. Several data mining methods can be used to reach this goal but we choose to focus only on association and sequential rule mining because other methods are more difficult to apply (see Section 3.1.2). For brevity, we shall henceforth refer to web server access logs as *logfiles*. Unfortunately, as logfiles are originally meant for debugging purposes, they are non-ideal candidates for WUM raw data [92]. However, due to their wide adoption by many existing web servers, we postulate that they will not be replaced as the *de facto* web data sources in the near future. Sources of logfiles include web servers, web clients, proxy servers and application servers [143, 92]. A standard logfile has the following format [43]: [*remotehost logname username date request status bytes*] where

- *remotehost* is the remote hostname or its IP address,
- *logname* is the remote logname of the user,
- *username* is the username as which the user has authenticated himself,
- *date* is the date and time of the request,
- *request* is the exact request line as it came from the client,
- *status* is the HTTP status code returned to the client, and
- *bytes* is the content-length of the document transferred.

The following is a fragment of a common logfile:

```
ntu.edu.sg - - [30/May/2003:00:01:15 -0400] "GET /html/faq.html HTTP/1.0" 200 4855
155.69.181.254 - - [1/Jun/2003:00:03:22 -0400] "GET /pub/tile/home.html HTTP/1.0" 200 165
```

As observed in the example, the fields *logname* and *username* are usually not recorded. Therefore, it is difficult to identify the activities of individual users. An extended format is also available to capture demographic data and session identifiers but we shall only focus on the standard common logfile format because of its wider usage and lesser privacy/security concerns [70].

Chapter 3. Web Usage Mining

There are several existing work on logfile mining and but they deal separately on specific issues of mining by making assumptions without taking a *holistic* view and thus, they could yield good results only in certain scenarios [37, 38, 44, 86, 125, 66, 115, 116]. Some take the web site structure into consideration; some focus only on traversal patterns; some consider the amount of time spent on a page. In this chapter, we shall examine such work and then introduce a more holistic version (which takes into consideration all of the above issues) of WUM termed *TRALOM* (**TR**Ansactionized **LO**gfile **M**ining) to effectively and correctly identify transactions as well as to mine useful knowledge from logfiles. Useful knowledge is knowledge that can be translated into useful actionable tasks mentioned in Section 3.1.3. TRALOM can discover more useful knowledge than existing approaches because it does not make many baseless assumptions and it exploits all the information stored in logfiles.

We also introduce a data structure called the *WebTrie*, a variant of the *SOTrieIT*, to efficiently hold useful preprocessed data so that TRALOM can be done in an *online* and *incremental* fashion; online means being able to mine the latest data with little overheads while incremental means being able to re-use past mining data. Our approach has been shown to be useful and viable by experiments conducted on a variety of real logfiles.

The rest of the chapter is organized as follows. The next section defines the phases of WUM. Section 3.2 reviews existing WUM algorithms. Our novel *SOTrieIT* variant and its accompanying algorithms are presented in Section 3.3. A framework for comparing WUM algorithms cognitively is unveiled in Section 3.4. WUM applications are discussed in Section 3.5 and lastly, the chapter is summarized in Section 3.6.

3.1 Phases of Web Usage Mining

In this section, we discuss the phases of WUM and define the mining problem.

3.1.1 Phase 1: Preprocessing

Several problems exist during the preprocessing phase where logfiles are transformed into a form that is suitable for mining. The following are preprocessing tasks that have been identified [46]:

1. *Data Cleaning*: The logfile is first examined to remove irrelevant entries such as those that represent multimedia data and scripts or uninteresting entries such as those belonging to top/bottom frames.
2. *User Identification*: Since several users may share a single machine name, certain heuristics are used to identify users [129].
3. *Session Identification*: After a user is identified, his page accesses must be divided (sessionized) into individual sessions [17]. A collection of such sessions is known as *clickstream data*.
4. *Path Completion*: Some accesses are not captured by the logfile and this may result in *incomplete paths* (requests made to a page not directly linked to the last requested page) in the log data. This is probably due to the use of local caches or proxy servers. Some solutions include *cache busting* (using cache specific headers to stop page caching) or the use of navigation history [129].
5. *Transaction Identification*: Once a session of completed paths is determined, page references must be grouped into logical units representing web transactions before any mining can be carried out.

3.1.2 Phase 2: Mining

There are four main mining techniques that can be applied to web access logs to extract knowledge:

1. *Sequential-pattern-mining-based* [37]: Allows the discovery of temporally-ordered web access patterns
2. *Association-rule-mining-based* [21]: Finds correlations among web pages
3. *Clustering-based* [110]: Groups users with similar characteristics

Session ID	Web Page Access Sequence	Avg Time
100	$A(4)B(3)C(21)B(5)D(4)E$	7.4
200	$B(4)D(2)B(5)C(17)E$	7
300	$C(4)D(5)E$	4.5
400	$A(6)B(24)D$	15

Table 3.1: A sample clickstream database.

4. *Classification-based* [23]: Groups users into predefined classes based on their characteristics

The above techniques help in the design of better web sites as well as in the development of effective marketing strategies. However, generally, users are not willing to disclose personal information and may tend to give false information [45]. Hence, it is more practical to assume the anonymity of users for WUM in general, especially for non-commercial sites where user registration is not required. Moreover, such user data cannot be found in standard logfiles which are the focus of our work. Therefore, we will not discuss algorithms based on clustering and classification where user information is needed. Here, we define the WUM problem as one that is sequential-pattern-mining-based and association-rule-mining-based.

3.1.3 Phase 3: Applying Mining Results

The last phase of WUM involves the analysis and translation of mining results into useful actionable tasks such as the following:

- Re-design web sites so that correlated pages are found together
- Improve access time by prefetching pages frequently accessed sequentially
- Improve caching by storing pages frequently revisited
- Enhance surfing experience by relocating pages in such a way that users need not visit unnecessary pages to get to their desired pages

3.2 Related Work

The focus of this section is on various existing approaches of identifying and analyzing transactions from sessions. In our subsequent discussions, we shall use the sample clickstream database in Table 3.1 for illustration purposes. At this point in time, we assume that user sessions can be accurately determined. This database contains a total of five web pages A, B, C, D and E and the bracketed numbers are their respective access times in seconds. The access time of a page is obtained by taking the difference between the time it is requested and the time the next page is requested. Note that it is not possible to determine the access time of the last accessed page of every session because logfiles do not contain enough information to determine when a user actually finishes reading a page. The last column contains the average access time of pages in a session.

Chen *et al.* pioneered work in path traversal pattern mining and popularized research in this area [37]. The authors coined the term, *mining traversal patterns*, which means the capturing of clickstream patterns in distributed information providing environments such as the Web. The procedure for mining traversal patterns consists of three steps:

1. *Determine transactions*: The authors assumed that backward references are used only for ease of travelling and focused on the discovery of forward reference patterns. A backward reference is defined as revisiting an object by the same user. The moment a backward reference occurs, a forward reference path is considered terminated and is termed a *maximal forward reference* (MFR) which represents a transaction. Another MFR is obtained by tracing back to the starting point of the previous MFR and ignoring unique references of the previous MFR. For example, given a traversal path $\{A, B, A, C, D\}$, the set of MFRs is $\{AB, ACD\}$. This step is repeated until the entire traversal log for a particular user is searched.
2. *Determine large reference sequences*: A *large reference sequence* (LRS) is a MFR that appears frequent enough to satisfy a minimum support threshold.

Chapter 3. Web Usage Mining

From the set of MFRs, LRSs are determined using the following two proposed algorithms:

- (a) *Full Scan (FS)*: FS follows the ideas of the DHP algorithm for mining association rules with one slight modification for the self-join procedure during candidate generation [122]. The following modification is necessary because unlike an itemset, a MFR has consecutive references: For any two distinct reference sequences, $\{a_1, \dots, a_{k-1}\}, \{b_1, \dots, b_{k-1}\} \in L_{k-1}$, they are combined to form a k -reference sequence only if either $\{a_1, \dots, a_{k-1}\}$ contains $\{b_1, \dots, b_{k-2}\}$ or $\{b_1, \dots, b_{k-1}\}$ contains $\{a_1, \dots, a_{k-2}\}$.
- (b) *Selective Scan (SS)*: SS is similar to FS except that it can avoid certain database scans by utilizing information in candidate references. The idea is to use $C_k * C_k$ to generate C_{k+1} instead of using $L_k * L_k$ which saves one scan. This idea works only for situations when the C_{k+1} generated in this way is not too large for the main memory. Experiments show that SS outperforms FS by up to two times for a database with 200K reference paths and a longest path of 10 references [38].

3. *Determine maximal reference sequences*: A *maximal reference sequence* (MRS) is a LRS that is not contained in any other MRS. From the set of LRSs, MRSs are determined. For example, given a set of LRSs $\{AB, BE, BC, ABE\}$, the resulting MRSs are BC and ABE .

Chen *et al.* do not make any distinction between references used for various purposes and may discover too many MRSs from the transactions identified using the MFR technique. Obviously, the MFR technique *over-evaluates* subsequences found at the beginning of a sequence; such subsequences are given higher importance. For example, consider session 100 of Table 3.1. For brevity, we shall use the term, S_X , to denote session X of Table 3.1. The resultant set of MFRs for S_{100} is $\{ABC, ABDE\}$. Although the subsequence $\{AB\}$ appears only once in S_{100} , it appears twice in the set of MFRs. In addition, the assumption that backward references are used only for

Chapter 3. Web Usage Mining

ease of travelling is only applicable in certain cases only. For example, in S_{200} , page D is only accessed for 2 seconds which suggests that the user may think that D is an irrelevant page and travels back quickly to B . However, the incorrect subsequence $\{BD\}$ would be discovered by the MFR technique. Finally, using the apriori-like technique to discover LRSs means limited scalability as proven in several ARM papers [75, 14, 166].

To group web page references into more meaningful transactions, Cooley *et al.* proposed a novel way of identifying transactions based on the assumption that a user uses a page for one of the following two subjective purposes [44]:

1. *Navigation*: Navigation pages are pages containing links to desired content.
2. *Content*: Content pages are pages containing the desired content.

With the above assumptions, a transaction can be defined in the following two ways:

1. *Navigation-Content*: A navigation-content transaction contains all navigation references leading to content references and the content references themselves for a given user. Mining such transactions yields frequent traversal paths to content pages.
2. *Content*: A content-only transaction contains only content references for a given user. Mining such transactions yields associations or correlations among content pages.

Assuming that the amount of time a user spends on a page is indicative of the type of the page, the authors introduced a *reference length* (RL) (duration of viewing reference) transaction identification method. In this method, the user must supply the percentage n of navigation references in a log so that a cutoff time c can be calculated to decide if a reference is for navigation or content purposes. This technique uses a chi-squared distribution to calculate the time c such that $n\%$ of the references in the log have a time length of less than c with a confidence of 95%. Experiments are

Chapter 3. Web Usage Mining

conducted to mine association rules from transactions identified by both the reference length and MFR techniques. In a MFR, the last reference is assumed to be a content reference while the rest are navigation references. The experiments revealed that the reference length method performed better for content-only transactions. Navigation-content transactions lead to too many rules and are deemed less useful. The RL method is clearly more practical than MFR because it takes into consideration the dimension of time. However, it suffers from two main weaknesses: n can only be determined accurately by an expert and there is a need to scan the entire logfile once first before c can be computed. In addition, since users may be accessing a site using heterogeneous systems with varying bandwidth connections and at different times with varying network traffic, using c may mistake content pages for navigation-content pages for systems with broadband connections used during off-peak periods.

Instead of preprocessing log data into a suitable form for ARM, Borges *et al.* proposed the reverse: generalizing ARM for graph-like or web data [21]. This is to retain the original semantics of a user's navigation and avoid over-evaluating references during transaction identification. The basic idea is to first build a weighted directed graph and then search the graph using depth-first search methods to find frequently traversed paths. Hence, the problem of ARM is generalized to finding frequent paths in the graph. Figure 3.1 shows the graph constructed from the database in Table 3.1. The numbers on the links are obtained by counting the number of occurrences of all consecutive sequences of length 2. From manual inspection, we will find that the longest and most frequent path is $\{ABDE\}$, which has a support of at least 2 along its links. However, this particular path does not even occur once in the database and is thus misleading. This problem is caused by the assumption that the path a user takes depends only on the previously accessed page. The authors also proposed another similar approach that is based on probabilistic grammars [22]. Instead of using weights, probabilities are used on the directed graphs. In this approach, generated strings with higher probabilities represent frequent paths.

Nanopoulos *et al.* argued that the technique of Chen *et al.* to find MRSs is not resistant to *noise* [117]. Noise in this context means random page accesses which are

Figure 3.1: A weighted directed graph.

not strictly intended by a user. To counter the effect of noise, the authors proposed a technique which takes into consideration, the structure of a site as well as *subpath* containment in obtaining support counts [117]. For example, the path $\{ACD\}$ is a subpath of S_{100} and is thus supported by S_{100} . The authors assumed that the graph structure of a site is available to determine the validity of candidate paths and designed an apriori-like algorithm similar to that of Chen *et al.* to mine frequent paths. Pei *et al.* followed up on this technique by proposing a faster scalable mining technique which is based on the *FP-tree* [75] approach [125]. However, we contend that this notion of noise is farfetched because its occurrence is minimal and should not affect mining. Even if noise consistently exists, it would be advantageous to detect it and capitalize on it; it may mean poor website design because most users cannot find what they want without navigating to unrelated pages. Moreover, by using subpath containment, too many frequent paths may be found and it is difficult to identify the more useful ones or differentiate which ones are due to noise.

There are also many other miscellaneous works that attempt to improve WUM in terms of query-processing and efficiency and we shall discuss some of them here for completeness. The use of *wildcards* in navigation paths is first proposed by Spiliopoulou *et al.* [140] and explored further by Gaul *et al.* [66]. Such paths are known as *generalized subsequences* and contain an additional wildcard symbol $*$ which denotes arbitrary subsequences. Generalized subsequences are effectively used only by experts in building specific queries using a mining language like MINT [140]. Spiliopoulou *et al.* also introduced *aggregate trees* which are similar to *FP-trees* [75].

Chapter 3. Web Usage Mining

Approach	Assumptions	Special Features	Weaknesses
<i>MFR</i>	Backward references used for ease of travelling	Nil	Over-evaluates subsequences
<i>RL</i>	Time spent on a page is indicative of type	Cutoff time to find navigation/content pages	Needs expert input; need full pre-scan; finds too many rules
<i>Weighted Directed Graph</i>	Path taken depends on previous page only	Generalised ARM to retain original semantics	Discovered paths are misleading
<i>Subpath Containment</i>	Noise is present and undesirable	Noise-resistant	Finds too many rules; noise may mean something important
<i>TRALOM</i>	Access time determines user's perception of page type	Image content proportion value γ ; Dual set of rules; incremental	Needs user to input support threshold and γ

Table 3.2: Summary of WUM approaches.

An aggregate tree is a trie which stores navigation paths and the number of times a particular path is visited. It is useful for obtaining support counts of paths but grows exponentially with respect to the number of pages on a site. Finally, Massealia *et al.* enabled the discovery of frequent patterns in real time through the use of distributed computing and heuristics [105].

In summary, there is not one approach that is suitable for use in all situations; the assumptions made by each approach are true only in certain scenarios. Table 3.2 shows a summary of WUM approaches, including TRALOM, our proposed approach. It may not be clear now how TRALOM compares with the rest but it can be observed from the table that it has more special features (through utilization of logfile data) to address the needs and issues of today's WUM problems. Its strengths will become more evident with the detailed analysis of each of its features as they are being introduced in the next section.

3.3 Algorithms

In this section, new techniques for sessionization, transactionization and mining of logfiles are introduced. Collectively, these techniques are termed *TRALOM* (**TR**Ansactionized **LO**gfile **M**ining). TRALOM is more holistic than existing approaches as it takes into consideration all the available information from logfiles and avoids making too many assumptions.

3.3.1 Sessionization

Firstly, all entries are removed from the logfile except successful HTTP requests for HTML files and image files. Unlike all existing work, we retain image files because they may contain important content (maps, product images etc.) and affect the calculation of access times of entries. On the one hand, a navigation page with many images to load would give the impression that the user is spending a lot of time viewing the content in it. On the other hand, the images themselves may be the actual desired content. Therefore, we use an image content proportion value γ to allow users to set the proportion of images with content. We postulate that being the creator of the site, a user should have a rough idea of how much content is stored in the images on his pages. Moreover, it is not important and also not possible to have an exact value as this is a subjective measure and will not greatly affect mining results. Hence, it is not difficult to set an appropriate γ . Of course, it would be good to compute γ automatically but this is a nontrivial challenge and would be addressed only in our future work.

The image entries will not be included in the final sessions but they are used to compute the access times of HTML pages that appear after the image entries in the following way: HTML page access time = $\gamma \times$ (sum of access times of all images after previous HTML page and before current HTML page). Setting $\gamma = 0$ means there is no content in images and while setting $\gamma = 1$ means all images contain content; the closer γ is to 1, the greater the proportion of images with content. All entries are then mapped to unique integers. Next, as Berendt *et al.* discovered that a 30-minute

session duration heuristic delivers the best results, it will be used here to group entries into sessions [17].

3.3.2 Transactionization

Once a session has been identified, it is important to determine whether a page is used for content or navigation purposes [44]. A content page is a page perceived by a user to contain content information while a navigation page is one perceived to contain link information to reach a content page. Before we describe the *Transactionization Algorithm* (TA), here are formulation preliminaries: Let the *universal pageset*, $U = \langle p_1, p_2, \dots, p_n \rangle$ be a set of *pages* found in a web site and n be the total number. A *k-consset* is an unordered set of k unique content pages. A *k-traset* is an ordered set of $(k - 1)$ unique navigation pages and a content page as its last page. Let $D_c \subseteq \mathcal{P}(U)$ (powerset of U) be a set of consets. Let $D_t \subseteq \mathcal{P}(U)$ be a set of traset. Let $s_m = \langle p_1, p_2, \dots, p_m \rangle$ be a session identified from the logfile using a 30-minute window with m pages. Let a_p be the access time of page p . Let A_{s_m} be the average access time of a page given by $\frac{1}{m-1} \sum_{p=1}^{m-1} a_p$ in a session s_m . Let $\uplus$ be an operator that appends a page to a traset.

A consset contains only pages that are accessed for their content rather than their links. A traset contains a sequence of navigation pages and a content page as its last page. A page is considered to be a content page if its access time is greater than the average access time of all the pages in a session. We contend that this is appropriate as it is logical that users spend more time on a content page than a navigation page. A study on tasks undertaken by users on the Web revealed that users spend an average time of about 75 seconds and 10 seconds on content and navigation pages respectively [32]. Recently, Cockburn *et al.* conducted an empirical analysis of web use and showed that the most frequently occurring time gap between subsequent page accesses was just about a second which meant that navigation pages are accessed most frequently [41]. Therefore, since users usually spend much more time on content pages and navigation pages are accessed more often than content pages, it is reasonable to conclude that this approach is feasible. Of course, we could

Chapter 3. Web Usage Mining

never be 100% sure that time spent on a page is fully spent on reading the content on the page as a user can be doing something else. But we can safely conclude that if a sufficient number of users spend a long time on a particular page, there is a high chance that it is a content page. This is done using a threshold to extract out only frequent cases of a page being perceived as a content page (see Section 3.3.4).

Our method differs from that of Cooley *et al.* as we do not require the user to make a good guess of the percentage of navigation pages and we save one logfile scan [44]. In addition, TA can also be applied on a session the moment it arrives because there is no need to calculate a suitable cutoff time based on all the sessions. Finally, since we use the average access time of pages in each session, our technique is not affected by varying bandwidth and traffic conditions across sessions. Algorithm 5 shows the steps of TA.

The basic idea of TA is to determine the nature (content or navigation) of a page using its access time and then assign it to meaningful sets. For each page in a session, we first check if it is the last page (step 7). If it is and since we cannot determine the access time of the last page, we assume it is a content page and add it to both conset X and traset Y . For every other page, we check if its access time is greater than or equal to the average access time of pages in that particular session (step 11). If it is, then we conclude that it is a content page and append it to Y and add Y to D_t . This is because every traset ends with one content page. We will only add this content page to X if it does not exist in X (step 15) as we are only interested in finding correlations between different pages with consets. If a page p is a navigation page and already exists in Y , it is assumed that since a user has returned to a previously traversed page without reaching a content page, the user has not found what he/she wants and thus, the intermediate navigation pages are actually mistakes and thus should not be included (step 18).

Example Both TA and MFR are applied on the logfile in Table 3.1 and Table 3.3 shows the extracted content, traversal and MFR transactions. An interesting observation is that a page can be both a content and a navigation page. This is due

Chapter 3. Web Usage Mining

Algorithm 5 Transactionization AlgorithmInput: A clickstream database D Output: A set of consets D_c and a set of traset D_t

```

1: Initialize  $D_c = D_t = \emptyset$ 
2: for each session  $s_n \in D$  do
3:   Initialize conset  $X = \emptyset$ 
4:   Initialize traset  $Y = \emptyset$ 
5:   Compute  $A_{s_n}$ 
6:   for each page  $p_i \in s_n$  where  $i = 0 \dots n$  do
7:     if  $i = n$  then
8:        $X = X \cup \{p_i\}$ 
9:        $Y = Y \uplus \{p_i\}$ 
10:    else
11:      if  $a_{p_i} \geq A_{s_n}$  then
12:         $Y = Y \uplus \{p_i\}$ 
13:         $D_t = D_t \cup Y$ 
14:        Initialize traset  $Y = \emptyset$ 
15:        if  $p_i \notin X$  then
16:           $X = X \cup \{p_i\}$ 
17:        end if
18:      else if  $p_i \in Y$  at position  $m$  then
19:        Remove all pages in  $Y$  from position  $m + 1$  onwards
20:      else
21:         $Y = Y \uplus \{p_i\}$ 
22:      end if
23:    end if
24:  end for
25:   $D_c = D_c \cup X$ 
26:   $D_t = D_t \cup Y$ 
27: end for

```

to different user perceptions. Rare perceptions are weeded out with the use of the support threshold. In the example, page B can be identified as both a content and navigation page. However, it is clear that it will not contribute to a frequent conset as it occurs only once in D_c but will more likely contribute to a frequent traset as it appears 3 times as a navigation page in D_t . Since page B is perceived as a navigation

D_c	D_t	MFR
Transactions		
$\{CE\}$	$\langle ABC \rangle$	$\langle ABC \rangle$
$\{CE\}$	$\langle BDE \rangle$	$\langle ABDE \rangle$
$\{DE\}$	$\langle BC \rangle$	$\langle BD \rangle$
$\{BD\}$	$\langle E \rangle$	$\langle BCE \rangle$
	$\langle CD \rangle$	$\langle CDE \rangle$
	$\langle E \rangle$	$\langle ABD \rangle$
	$\langle AB \rangle$	
	$\langle D \rangle$	

Table 3.3: Extracted transactions from Table 3.1.

page, it could be prefetched or cached with other pages in the same trasets as itself in order to give users a faster response time during navigation. Another observation is that transactions identified by MFR have longer lengths. This is undesirable as such longer transactions will result in several rules being mined and since such transactions contain both content and navigation pages, mined rules may be misleading. For example, if the minimal support count is 2, $\langle ABD \rangle$ will be a frequent sequence. In S_{100} , it is a series of navigation pages while in S_{400} , it has more content pages. Hence, there is no way of deciding whether this sequence should be used to aid in the design/layout of navigation or content pages.

3.3.3 Data Structure: WebTrie

To cope with the challenges of data streams, we have introduced the SOTrieIT to hold preprocessed data so that ARM can be done quickly, incrementally and dynamically (see Chapter 2). In this section, we shall adapt its ideas to design a new data structure called a *WebTrie* for storing pertinent logfile data. We need to first define a set called a *paset*: A *k-paset* is an ordered set of *k* unique pages and may contain a mixture of content and navigation pages. The WebTrie is specifically designed to mine pasets. Paset mining reveals frequent paths taken by users to reach their desired content pages.

Definition 2 (WebTrie) A *WebTrie* consists of a tree node *w* which has a label

Chapter 3. Web Usage Mining

Algorithm 6 Construction Algorithm of the WebTrieInput: A set of traset D_t and an empty WebTrie $\mathcal{W}$ Output: An updated $\mathcal{W}$

```

1: for each  $n$ -traset  $T_n \in D_t$  do
2:   for each page  $p_i \in T_n$  where  $i = 0 \dots n$  do
3:     if  $w_{p_i} \in \mathcal{W}$  then
4:        $\beta_{w_{p_i}}++$ 
5:     else
6:       Create new node  $w_{p_i}$  under root node  $R$  with  $\beta_{w_{p_i}} = 1$ 
7:     end if
8:     if  $i \neq n$  then
9:       if  $w_{p_{i+1}} \in C(w_{p_i})$  then
10:         $\beta_{w_{p_{i+1}}}++$ 
11:      else
12:        Create child node  $w_{p_{i+1}}$  under parent node  $w_{p_i}$  with  $\beta_{w_{p_{i+1}}} = 1$ 
13:      end if
14:    end if
15:  end for
16: end for

```

representing a navigation page and an integer representing its support. We use w_{p_i} to refer to a tree node that corresponds to a page $p_i \in U$ and $\beta_{w_{p_i}}$ to represent its support count. Let $C(w_{p_i})$ be the support-ordered set of child nodes of node w_{p_i} . If $C(w_{p_i}) \neq \emptyset$, then $C(w_{p_i}) \subseteq \{w_m, \dots, w_n\}$ where $1 \leq m \leq n \wedge \beta_{w_m} \geq \beta_{w_{m+1}} \wedge m \neq i$. ■

Note that an implicit virtual root node R is needed to link all the WebTries together. The WebTries under R are sorted in support-descending order where the leftmost WebTrie has the highest support count. The WebTrie $\mathcal{W}$ is constructed using Algorithm 6.

For every page p in a traset, we first check if there is a first-level tree node representing it (step 3). If there is, we simply increment its support count by 1. If not, we will create a node under R to represent it. If p is not the last page of the traset, we next check the page that comes immediately after it to see if a child node is needed to be created or we can simply increment its support count (step 8). With this con-

Figure 3.2: A WebTrie Example.

struction algorithm, information about every 2-paset that is supported by the traset in D_t is stored in the WebTrie. Though not explicitly mentioned above, all tree nodes and their child nodes are sorted in a support-descending order and thus, every change to a WebTrie may require a shifting of nodes to maintain the order. Note that $\mathcal{W}$ is created/updated on-the-fly each time TA identifies a traset; the WebTrie is considered to be constructed *online* because at any point in time, it reflects the actual current state of the database. Like the SOTrieIT, the WebTrie is *support-independent* and thus, does not require any user input to begin construction. In addition, it is also *incremental* because it need not be rebuilt from scratch whenever new log entries arrive or when different support thresholds are used. Hence, using the WebTrie will make our mining algorithm both online and incremental. Figure 3.2 shows the WebTrie constructed from D_t found in Table 3.3. It has a space complexity of $O(n^2)$ while the weighted directed graph has a space complexity of $O(n)$ where n is the size of the universal pageset [21]. We will see that the advantages reaped at the expense of space are worthwhile in the next section where we present our mining algorithms.

3.3.4 Mining

In this section, the algorithms for discovering knowledge from consets and traset efficiently are discussed. Here are formulation preliminaries: A consset $T \in D_c$ is said to support a consset X iff $X \subseteq T$. Let σ_X be the *support count* of an consset X , which is the number of conssets in D_c that support X . A paset $X = \langle x_1, x_2, \dots, x_j \rangle$ is said to be a *consecutive subsequence* of traset $Y = \langle y_1, y_2, \dots, y_k \rangle$, $X \triangleright Y$, iff $\exists i$ s.t. $y_{i+m} = x_m$ for $1 \leq m \leq j$. A traset $T \in D_t$ is said to support a paset X iff $X \triangleright T$. Let α_X be the *support count* of an paset X , which is the number of traset in D_t that support X . Let $\mathcal{S}$ be the *support threshold* and $|D|$ be the number of transactions in a transaction database D . A consset X is *large* or *frequent* if $\sigma_X \geq |D_c| \times \mathcal{S}\%$. A paset X is *large* or *frequent* if $\alpha_X \geq |D_t| \times \mathcal{S}\%$. A **Web Usage Rule** (WUR) is an implication of the form $X \implies Y$, where X and Y are sets of frequent conssets and $X \cap Y = \emptyset$. The WUR $X \implies Y$ holds in the database D_c with *confidence* $c\%$ if no less than $c\%$ of the content transactions in D_c that contain X also contain Y . A **Web Traversal Blueprint** (WTB) is a set of all frequent pasets.

A WUR is analogous to an association rule and thus, all the advancements of ARM can be brought to bear on mining WURs. A WUR shows how content pages are correlated and helps web site designers to decide which pages should be placed near to each other. In an electronic commerce context, WURs will discover which product pages are usually assessed together and the site owner can then place such pages close together within a link away to maximize sales. We will not elaborate more on WUR mining because it is a straightforward task since we have modelled the problem as an ARM problem.

The WTB reveals most frequently accessed traversal paths to particular content pages and thus, web designers can use the WTB to prefetch pages or modify the site so that users need minimal traversals to reach their desired content pages. By making navigation easy and fast, WTB can help improve the popularity of electronic commerce sites. WTB discovery is performed using the WebTrie $\mathcal{W}$ constructed from D_t given a support threshold $\mathcal{S}$ and its steps are detailed in Algorithm 7 (see page

78).

The main idea of our algorithm is to traverse $\mathcal{W}$ in a depth-first search manner and generate larger-sized pasets by linking child nodes to parent nodes of similar labels. We begin by checking tree nodes under R from the left branch (step 4). If a node has enough support, we shall add the page it is representing to the set of frequent pasets L_j and invoke procedure **FindTrail** on it (step 7). **FindTrail** recursively tries to discover larger-sized frequent pasets by checking the child nodes of first-level nodes and then using these child nodes to link to other first-level nodes under R (step 11 of FindTrail). Since we know that trasetes do not contain duplicate pages, we will skip nodes that represent pages that are already in the paset X (step 2 of FindTrail). If a child node represents a page p with sufficient support, we shall add p to L_j and invoke **FindTrail** on it (step 10 of FindTrail). Note that we only add a paset to the set of candidate pasets C_j only when its size is greater than 2 (step 7 of FindTrail). This is because pasets of sizes 1 and 2 have their full support counts stored in the WebTrie. The support counts of candidate pasets have to be obtained by scanning D_t later on (step 13) because the WebTries are previously updated with only 2-pasets and their support counts. As all tree nodes are sorted in descending order of their support counts, our search algorithm is faster than that of [21]; the moment a tree node is found to be infrequent, its siblings need not be checked. This pruning strategy is applied to both the main algorithm (step 9) and the procedure **FindTrail** (step 13 of FindTrail).

Example Given a support threshold of 25%, applying our WTB mining algorithm on the WebTrie in Figure 3.2 yields the following results: $C_k = \{\langle ABC \rangle\}$, $WTB = \{\langle B \rangle, \langle C \rangle, \langle D \rangle, \langle E \rangle, \langle A \rangle, \langle BC \rangle, \langle AB \rangle\}$. The paset $\langle ABC \rangle$ has a support count of 2 according to $\mathcal{W}$ but when checked against D_t , it has a support count of only 1 and thus, it is an infrequent paset. We run a standard ARM algorithm on the consets found in Table 2 and the following are the frequent consets found:

$$\{B\}, \{C\}, \{D\}, \{E\}, \{BD\}, \{CE\}, \{DE\}.$$

With the same support, applying the FS algorithm on the database found in Table 3.1 yields the following LRSs:

$$\langle A \rangle, \langle B \rangle, \langle C \rangle, \langle D \rangle, \langle E \rangle, \langle AB \rangle, \langle BC \rangle, \langle BD \rangle, \langle DE \rangle, \langle ABD \rangle.$$

It is clear that FS discover more sequence patterns but since it does not differentiate between consets and paset, the mined knowledge is difficult to apply; do the patterns suggest bringing pages close together because they are correlated content pages or prefetching pages for faster access because they are navigation pages? In addition, it has also missed out one frequent consset $\{CE\}$. The reason is because pages C and E are not consecutive pages as seen in S_{100} and S_{300} and thus, they cannot be found using FS. However, we know that correlated content pages need not necessarily be consecutive pages although navigation pages must be consecutive. On the other hand, if we check the results of our algorithms against the original database found in Table 3.1, we would discover useful insights. For example, the frequent paset $\langle BC \rangle$ tells us that BC is an important path to a content page. We can see from S_{100} and S_{200} that this insight is meaningful because to reach the content page C , we must go through navigation page B . The frequent consset $\{CE\}$ tells us that content pages C and E are often accessed in a session and are thus correlated. A check from S_{100} and S_{200} once again confirms this.

3.3.5 Optimization Strategies

We are interested only in page sequences that occur frequently because they represent interesting and important trends. We have previously introduced the WTB algorithm to mine sequences together with the WebTrie but it is not efficient at lower support thresholds where extremely long patterns are discovered. This is due to the large number of *candidate* (potentially-frequent) patterns generated and the need to scan the logfile to obtain the actual support counts of each of them.

Formerly, we have presented an algorithm, which we now term WTBD1 (WTB Discovery 1), to mine for the WTB. WTBD1 uses the WebTrie constructed from D_t given a support threshold $\mathcal{S}$. From past experiments, it was observed that WTBD1

Chapter 3. Web Usage Mining

Symbol	Duration	Size (MB)	Number of Unique URLs	Source
$\mathcal{D}_1$	1 week	168	8436	Internet access provider for the Metro Baltimore
$\mathcal{D}_2$	1 month	200	941	NASA Kennedy Space Center
$\mathcal{D}_3$	7 months	228	2981	University of Saskatchewan

Table 3.4: Real logfiles.

	$ D_c $	$ D_t $	Average size of consets	Average size of traset	Time to process one traset for WebTrie (ms)
$\mathcal{D}_1$	49K(97K)	171K(209K)	2.4(2)	2.1(1.3)	104(0.28)
$\mathcal{D}_2$	38K(88K)	121K(299K)	3.2(2.7)	2.7(1.3)	746(0.41)
$\mathcal{D}_3$	78K(46K)	132K(241K)	2.1(3.4)	1.8(1.3)	13(0.23)

Table 3.5: Transactionization and WebTrie Details.

performed badly whenever there are many frequent paset sets having a size greater than two [156]. This is due to the huge number of candidate paset sets being generated by the recursive function **FindTrail** and the need to scan the database to confirm their actual support counts. To eliminate this candidate generation problem, we introduce an improved version of WTBD1, called the WTBD2 algorithm (Algorithm 8; see page 79).

The main idea is to reduce the number of database scans by earlier database scans to eliminate the need for candidate paset generation. Unlike WTBD1, WTBD2 only invokes the new procedure **FindTrail** recursively on frequent paset sets (steps 7-10 of FindTrail). This is intuitive because if a paset has been found to be infrequent, it will still be infrequent even if pages are appended to it. Therefore, by scanning the database earlier (step 7 of FindTrail), unnecessary calls to **FindTrail** as well as candidate paset generation can be avoided. The next section shows the effectiveness of this approach.

3.3.6 Experiments

This section evaluates the performance and practicality of the TRALOM techniques by conducting experiments on a Pentium-4 machine with a CPU clock rate of 2.4 GHz, 1 GB of main memory and running on a Windows 2000 platform. The algorithms are implemented in Java and the real logfiles can be freely downloaded [50]. These logfiles were also used in a workload characterization study for web servers [10]. They are useful as they are obtained from diversified environments as seen in Table 3.4. Table 3.5 shows the details of the transactionized logfiles and the construction time of WebTries. We have set the content proportion value γ of images to 0.25 as we assume that images at the three sites do not contain much content; this is speculative but it suffices to show the effect of γ . The bracketed numbers are the details of the transactionized logfiles in our previous work where images are totally disregarded and thus removed even before sessionization, i.e., $\gamma = 1$ [156].

As seen from Table 3.5, the conset databases of $\mathcal{D}_1$ and $\mathcal{D}_2$ are reduced by half while that of $\mathcal{D}_3$ has almost doubled. The reduction of conset databases is due to an increase in the size of each conset (see column 4 of Table 3.5) which in turn is caused by more pages being identified as content pages in a session. By setting $\gamma = 0.25$, the average access time of a session would be decreased (if a few pages have many images that have long loading time in the session) and more pages would have an access time greater than the average. However, in the case of $\mathcal{D}_3$, the average size of consets have decreased. This is probably due to the fact that most pages in a session need to load several images and with $\gamma = 0.25$ now, such pages would have a shorter access time compared to the scenario where $\gamma = 1$ and thus, would not qualify as content pages. Another observation is that in all three databases, the average size of traset has increased and this causes an huge increase in the time to process each traset for the construction of the WebTrie (see columns 5 and 6 of Table 3.5).

Due to the lack of a fair metric for comparing various logfile mining algorithms, only three algorithms are used in the experiments: Apriori (for conset mining), WTBD1 and WTDB2. Hence, our objective here is to prove the practicality of our

Chapter 3. Web Usage Mining

Figure 3.3: Execution times for $\mathcal{D}_1$ at varying minimum support thresholds. Figure 3.4: Execution times for $\mathcal{D}_2$ at varying minimum support thresholds.

approach while comparison issues are discussed in the next section. Figures 3.3, 3.4 and 3.5 show the performance of the three algorithms on $\mathcal{D}_1$, $\mathcal{D}_2$ and $\mathcal{D}_3$ at varying minimum support thresholds. Note that execution times are plotted on a log scale. The construction time of the WebTrie is not reflected in the performance figures due to amortization (the WebTrie can be re-used for several different support thresholds). This premise is reasonable because the optimal support threshold cannot be known a priori and must be adjusted by the user through experiments with various support thresholds.

From the three figures, it can be seen that WTBD1 performs poorly at low support thresholds (in many cases, its computation time is not plotted as it exceeds 1,000s).

Chapter 3. Web Usage Mining

Figure 3.5: Execution times for $\mathcal{D}_3$ at varying minimum support thresholds.

In all cases, WTBD2 outperforms WTBD1 by a wide margin especially at the lower support thresholds by eliminating the need for candidate paset generation. Conset mining generally takes a longer time even though its database D_c is much smaller than D_t . Probably, this is due to the use of the classic Apriori algorithm which also involves candidate generation. Fortunately, conset mining has the advantage of having the ability to plug in faster association rule mining algorithms whenever they are available and thus, minimal work is needed to speed up conset mining.

Another striking observation is that paset mining takes less than a second for high support thresholds (between 0.5% and 1%) for both WTBD1 and WTBD2. This is because the WebTries holds information that allows frequent pasets of lengths one and two to be discovered almost instantly without database scans. This characteristic is highly desirable as analysts need to make adjustments to the support threshold quickly especially at higher support thresholds. However, it can be seen that performance degrades exponentially for WTBD1 as the support threshold is lowered. This is due to the need to scan D_t several times to determine whether a candidate paset of size 3 and above is frequent. WTBD2 fares much better and can perform within 200s in all three logfiles even at a low threshold of 0.05%. The performance difference between WTBD1 and WTBD2 is most evident in $\mathcal{D}_2$ because it contains longer trasets; there are many more longer frequent pasets and thus, many more candidate pasets are generated by WTBD1.

Chapter 3. Web Usage Mining

Figure 3.6: Number of frequent k -consents and k -pasets in $\mathcal{D}_1$, $\mathcal{D}_2$ and $\mathcal{D}_3$ at a minimum support threshold of 0.05%.

Finally, Figure 3.6 shows the number of mined frequent k -consents and k -pasets in the three logfiles at a minimum support threshold of 0.05%. The number of frequent consents discovered is generally larger than that of frequent pasets except in $\mathcal{D}_3$. Therefore, it is obvious that the support thresholds for consent and paset mining should be different and adjusted accordingly. Another observation is that there are many more consents and pasets of size 3 and below. If the objective is to uncover long frequent patterns, then such shorter patterns would be redundantly generated.

3.4 Framework for Evaluating Web Usage Mining Algorithms

Without a proper means of measurement, WUM is at best a piece of calculated guesswork. Therefore, in this section, we design a framework termed *WALMAGE* (**W**eb **A**ccess **L**og **M**ining **A**lgorithm **E**valuator) to represent web scenarios and web access log mining algorithms in a way that facilitates the choice of the most appropriate algorithm to apply in a particular scenario. We propose a cognitive approach to model our framework so that a wide range of user behavior can be quantified in order to imbue our framework with realism and practicality. In this preliminary work, we shall focus only on the domain of electronic commerce since it is touted as the killer domain for data mining.

We assume that in a real situation, sessions can be accurately determined using heuristics proposed by Berendt *et al.* [17]. Hence, our objective is to evaluate the correctness of transaction identification algorithms as well as their corresponding mining algorithms. Currently, such algorithms justify themselves by making cognitive assumptions without concrete evidence or formal proofs. Our framework allows us to evaluate them more conclusively by providing a set of cognitive measures to describe electronic commerce scenarios as well as logfile mining algorithms and a suitability measure to determine the best algorithm for the scenario. As we lack real web access logs of electronic commerce sites, we cannot evaluate the correctness and usefulness of our framework in this preliminary work. However, in any case, WALMAGE helps to remove the need for groundless assumptions and speculations and brings us a step ahead in current WUM research.

3.4.1 Measuring User Behavior

Here are formulation preliminaries: Let $x = \langle l, a, h \rangle$ be an ordered set of measures denoting the degree of a characteristic being held true where l (low), a (average) and h (high) are the lowest, average and highest levels respectively. The measure x appears as the subscript of a characteristic.

Firstly, we introduce the following four characteristics that describe browsing behavior:

1. *Backtracking Tendency (B)*: Likelihood of backtracking to visited pages
2. *Exploration Tendency (E)*: Likelihood of exploring the whole web site
3. *Errancy (Er)*: Likelihood of going to a wrong page during browsing
4. *Time (T)*: Likelihood of spending a long time on a web page

They are chosen because web users display strong tendencies in backtracking, revisiting pages and viewing pages for short periods of time in a recent empirical study of the behavior of web users [41]. They are termed *base* characteristics because they are the building blocks for describing users and logfile mining techniques. Without

	B	E	Er	T
D_l	1	0	0	0
D_h	-1	0	0	0
F_l	0	1	1	1
F_h	-1	-1	-1	-1
R_l	0	0	0	1
R_h	0	0	0	-1

Table 3.6: Relationship of browsing and user characteristics.

loss of generality and for simplicity, we assign them the quantitative numerical values of -1, 0 and 1 for the degrees of l , a and h of x respectively. In future, empirical studies like those of Huberman *et al.* can be conducted to yield a fitting probability distribution for their values [85].

Secondly, we introduce the following three characteristics that describe users:

1. *Decisiveness* (D_x): Decisiveness is the degree of certainty when a user chooses a particular web page; a highly decisive (D_h) user is unlikely to explore the web site thoroughly because he can make a decision on what he wants quickly and does not backtrack frequently.
2. *Familiarity* (F_x): Familiarity refers to how knowledgeable a user is with regards to the structure of a web site as well as to the products being offered on the site. A user who is very familiar (F_h) with a web site can get to his desired page very quickly with little exploration.
3. *Responsiveness* (R_x): Responsiveness is a measure of how quickly a user can read a web page and respond to it. A responsive (R_h) user, usually an experienced user with a high-speed internet connection, can navigate quickly within a web site.

User characteristics, in turn, can be described by browsing characteristics and Table 3.6 shows the relationship between user and browsing characteristics. Each user characteristic is a point in a 4-dimensional plane where each base characteristic

	Characteristics	B	E	Er	T
U_c	$F_l \wedge D_l$	1	1	1	1
U_n	$F_l \wedge D_h \wedge R_l$	-1	1	1	2
U_d	F_h	-1	-1	-1	-1
U_l	$F_h \wedge R_h$	-1	-1	-1	-2

Table 3.7: Characteristics of various types of customers.

represents a dimension. Every user characteristic can be combined with other user characteristics to describe users.

3.4.2 Describing Customer Behavior

We propose the following basic ways to describe an online customer:

- *Curious* (U_c): A curious customer is one who surfs the site every now and then but does not buy anything.
- *New* (U_n): A new customer is one who has recently bought something from the site.
- *Dormant* (U_d): A dormant customer is one who has not bought anything from the site for a while.
- *Loyal* (U_l): A loyal customer is one who has been buying products from the web site for a long time.

Like base characteristics, the above ways of describing a customer can be combined so that customers can be described more accurately. Table 3.7 shows the combination of characteristics for various types of customers. When characteristics are combined, the values of the base characteristics are summed. It makes sense because combinations of certain customer characteristics increase the likelihood of base characteristics. Note that the quantitative numbers are not proportions and can exceed 1; the greater the number, the greater the likelihood of a base characteristic. For example, from Table 3.7, a new customer U_n has a value 2 for its time T characteristic. This means that such a customer has a much higher likelihood of spending a long time on a page compared to other types of customers.

	B	E	Er	T
C_l	1	-1	-1	0
C_h	-1	1	1	0
S_l	0	-1	-1	0
S_h	0	1	1	0

Table 3.8: Adjustive Characteristics.

Mining algorithm	Symbol	B	E	Er	T
Maximal Forward	$\mathcal{L}_1$	3	3	-3	0
Reference					
Reference Length	$\mathcal{L}_2$	-3	3	0	-3
Generalized ARM	$\mathcal{L}_3$	-3	-3	-3	0
Subpath-Oriented	$\mathcal{L}_4$	0	3	3	0

Table 3.9: Description of logfile mining algorithms using WALMAGE.

3.4.3 Describing Web Site Structure

We propose to describe a web site by the following characteristics:

- *Connectivity* (C_x): A well-connected web site (C_h) is one that probably has a search facility that allows users to zero in on a particular product directly. In addition, web pages can be reached from several parts of the site.
- *Size* (S_x): A large site (S_h) has several web pages featuring several products.

We term the above characteristics *adjustive* characteristics because they are used to adjust the final vector of a particular scenario. Table 3.8 shows how they affect a web scenario. The values of the adjustive characteristics are added after considering the type of users for a particular web site.

3.4.4 Describing Logfile Mining Algorithms

In this section, we will describe existing logfile mining algorithms using our framework. Table 3.9 shows how they can be described using the base characteristics. Note that the minimum and maximum values for a base characteristic are pegged at -3 and 3 respectively because user characteristics and two web site adjustive characteristics are

Chapter 3. Web Usage Mining

used to describe a web scenario. The following explains how we arrive at the values of their base characteristics:

- *Maximal Forward Reference*: This technique places emphasis on forward navigation patterns and disregards backward navigation patterns. Hence, it is suitable for scenarios where users are interested in exploring the web site; it assumes that backtracking is not due to errors or a user's unfamiliarity.
- *Reference Length*: Viewing time of a page is the main concern of this technique because it is used to determine the nature of the page itself according to the user's perspective. However, the viewing time may be misleading if users are already familiar with the web site and thus, spend almost the same amount of time on both content and navigation pages. Moreover, if most users have slow connections, it is also difficult to tell the difference between content and navigation pages based on viewing time as majority of the time is spent on retrieving documents.
- *Generalized ARM*: This method assumes that the path a user takes depends only on the current page. It is unsuitable in situations where new users are exploring a large site and making lots of wrong choices because the current page is unlikely the desired page in the first place.
- *Subpath-Oriented*: Subpath-oriented algorithms are highly resistant to noise and browsing mistakes. Therefore, it is most suitable in situations where users are likely to make navigation errors. They are not useful in other scenarios because they tend to generate too many patterns.

In the next section, we will present how WALMAGE can be used to describe an electronic commerce scenario and subsequently show how the suitability of a logfile mining algorithm can be assessed for it.

3.4.5 Suitability Analysis

According to WALMAGE, an electronic commerce scenario, $\mathcal{W}$, can be defined by the types of customers assessing its web site as well as the structure of its web site:

$$\mathcal{W} = w_1U_1 + w_2U_2 + \dots + w_nU_n + C_x + S_x$$

where $w_1 \dots w_n$ are weights denoting the proportion of users and thus sum up to 1, and $U_1 \dots U_n$ are various types of users, and C_x and S_x are the adjustive characteristics. We use the Euclidean distance, $\mathcal{E}_{\mathcal{W}\mathcal{L}}$, between a web scenario, $\mathcal{W}$ and a logfile mining algorithm, $\mathcal{L}$, as our suitability measure. The smaller $\mathcal{E}_{\mathcal{W}\mathcal{L}}$ is, the more suitable $\mathcal{L}$ is for $\mathcal{W}$.

Example Let us consider a scenario where a new electronic commerce web site, $\mathcal{W}_n$, has just been set up. As it is a new site, it probably has very few loyal customers and the majority of its visitors are either curious or new customers. In addition, it offers only a few products and thus, its site is small and simple. Based on WALMAGE and our proposed categories of customers, $\mathcal{W}_n$ can be described in the following way:

$$\begin{aligned}\mathcal{W}_n &= 0.5U_c + 0.4U_n + 0.1U_l + C_l + S_l \\ &= (1, -1.2, -1.2, 1.1)\end{aligned}$$

We shall consider the algorithms in Table 3.9 as possible candidates for mining the logfiles of $\mathcal{W}_n$. The following are the values of their suitability measures:

$$\begin{aligned}\mathcal{E}_{\mathcal{W}_n\mathcal{L}_1} &= \sqrt{(3-1)^2 + (3+1.2)^2 + (-3+1.2)^2 + (0-1.1)^2} \\ &= \sqrt{26.09} \\ &= 5.1 \\ \mathcal{E}_{\mathcal{W}_n\mathcal{L}_2} &= \sqrt{(-3-1)^2 + (3+1.2)^2 + (0+1.2)^2 + (-3-1.1)^2} \\ &= \sqrt{51.89} \\ &= 7.2 \\ \mathcal{E}_{\mathcal{W}_n\mathcal{L}_3} &= \sqrt{(-3-1)^2 + (-3+1.2)^2 + (-3+1.2)^2 + (0-1.1)^2}\end{aligned}$$

$$\begin{aligned}
&= \sqrt{23.69} \\
&= 4.9 \\
\mathcal{E}_{\mathcal{W}_n \mathcal{L}_4} &= \sqrt{(0-1)^2 + (3+1.2)^2 + (3+1.2)^2 + (0-1.1)^2} \\
&= \sqrt{37.49} \\
&= 6.1
\end{aligned}$$

From the above, $\mathcal{E}_{\mathcal{W}_n \mathcal{L}_3}$ has the lowest value and thus, $\mathcal{L}_3$, Generalized ARM, is the most suitable logfile mining algorithm for $\mathcal{W}_n$. Alternatively, the cosine of the angle between the two vectors, $\mathcal{C}_{\mathcal{W}\mathcal{L}}$, can also be used to verify the results by the Euclidean distance measure in the event that the distances are very close to one another. The closer the value of $\mathcal{C}_{\mathcal{W}\mathcal{L}}$ is to one, the more similar the two vectors are. The choice of Generalized ARM makes sense here because for such small new sites, it is unnecessary or even detrimental to make assumptions about backward references, page access times and noise.

3.5 Applications

WUM is fast gaining importance because of the wide availability of logfiles as well as its applicability in CRM [156]. Unlike existing approaches, TRALOM is able to address the needs of current applications because it is online, incremental, fast and discovers knowledge that can be easily used to improve the user-friendliness of websites. As mentioned, TRALOM does not make unnecessary assumptions that might otherwise distort mining results. TRALOM can be used on diversified applications such as the following:

- *Web Personalization* [112, 110]: Web personalization is described as any action that personalizes the web experience of a user based on his/her preferences. It is important in electronic commerce because it helps to attract customers by anticipating their needs and boost sales by recommending products that customers would probably like based on the analysis of their clickstream data using TRALOM techniques.

Figure 3.7: Product Development Collaborators

- *Web Site Structuring* [106, 139]: A good web site is one that allows a user to browse to his/her desired page with minimum clicks. TRALOM can discover groups of frequently accessed pages and this knowledge can be used to re-structure a web site effectively by storing those pages in a single page or ensure that they are within a single click away.
- *Caching and Prefetching* [162]: Caching and prefetching are two effective ways of reducing/eliminating page loading time. TRALOM can help predict future pages to be accessed based on the user's current clickstream data and prefetch those pages. In addition, frequently accessed pages can be discovered by TRALOM and cached.

As we have studied in depth the application of WUM in the domain of manufacturing, we will discuss it here in detail [153]. According to *Parametric Technology Corporation*, global manufacturing investment in product development solutions is expected to reach more than \$25 billion dollars [49]. This is largely due to a revolutionary change of mindset of manufacturing companies. Instead of just focusing on developing great products, companies are looking into the prospect of converting product development processes into a competitive weapon.

Figure 3.7 shows the common collaborators in product development. Current product development solutions allow them to interact with one another and partic-

Chapter 3. Web Usage Mining

ipate actively throughout the entire product development process as represented by the bidirectional arrows in Figure 3.7. Some commercial product development systems include *Windchill* [48] and *eMatrix* [107]. In global manufacturing, designers and engineering no longer contribute most in product design; all collaborators play substantial roles and in some cases, suppliers contribute up to 70% of the final product design [49].

As interactions and activities of collaborators are carried out via web-based clients, large amounts of data are captured in the form of logfile entries. Hence, there is valuable knowledge waiting to be unearthed in such logfiles. The following are some previously unknown knowledge that can be extracted and used as competitive weapons:

1. *Part-Assembly relationships*: Although such relationships can be obtain by querying the part-assembly database, logfile mining can help sieve out significant relationships with greater speed and lesser supervision. Such knowledge is critical for fast product design.
2. *Frequently-accessed web pages*: Patterns of sequential accesses of web pages allow webmasters to improve structure of web client for faster access.
3. *Collaboration relationships*: Collaborators access the product development system at various timepoints of the product development process. By understanding how collaboration is usually carried out, workflow processes can be fine-tuned and optimized. For example, if it is discovered that customers usually step in to alter design even after suppliers have confirmed shipment of parts, then the workflow should include time buffers to cushion sudden changes of customer requirements. Due to the complexity and security of login mechanisms of product development software, such relationships are still difficult to mine and will be handled only in our future work.

Figure 3.8 shows the traditional phases of the product development process and here is a brief description of each phase in proper sequence [130]:

1. *Requirements Definition*: Involves identifying the needs of the customer and defining the business and design objectives for the product

Figure 3.8: Product Development Processes

2. *Conceptual Design*: Involves identifying possible design approaches based on requirements
3. *Detailed Design*: Involves evaluating design approaches and finalizing the design of the product
4. *Test and Evaluation*: Involves correcting problems, identifying areas for improvement and reducing risks
5. *Manufacturing*: Involves producing the product
6. *Logistics*: Involves planning and controlling the flow and storage of products and their components

The bidirectional arrows in Figure 3.8 indicate that the above phases constantly interact with one another at various stages of the product development process. For example, even when a product is already being shipped to customers (last phase), transportation problems such as the unforeseen fragility of the product may force engineers to go back to the drawing board (second phase). Therefore, it is highly desirable to use WUM to discover trends in the complex mechanics of the product development process for streamlining purposes.

3.6 Summary

In this chapter, the problem of WUM has been discussed and its existing approaches reviewed. We have designed a new SOTrieIT variant called the WebTrie to work together with our WUM approach called TRALOM. TRALOM is an attempt to provide a more holistic and yet efficient solution to WUM. Its strength and usefulness stem from the following features:

- The use of a content proportion value to specify the amount of content in images
- The application of a transactionization technique on a logfile to split it into two different databases for two different mining tasks so as to discover more meaningful knowledge
- The use of a reliable and logical way to differentiate between a content and navigation page
- The use of the WebTrie to efficiently store preprocessed information for fast mining
- The ability to use any association rule mining algorithm for discovering web usage rules
- The use of an efficient algorithm to traverse the WebTrie to discover the web traversal blueprint

We have also presented our preliminary work on a framework for comparing the suitability of WUM algorithms for specific scenarios. Finally, the usefulness and practicality of WUM are proven with discussions on some of its many and varied applications.

Chapter 3. Web Usage Mining

Algorithm 7 Web Traversal Blueprint Discovery AlgorithmInput: A set of traset D_t , a support threshold $\mathcal{S}$ and a WebTrie $\mathcal{W}$

Output: A Web Traversal Blueprint

```

1: Let  $\ell$  be size of the largest traset in  $D_t$ 
2: Initialize set  $C_j = \emptyset$  for  $1 \leq j \leq \ell$ 
3: Initialize set  $L_j = \emptyset$  for  $1 \leq j \leq \ell$ 
4: for each WebTrie tree node  $w_{p_i} \in C(R)$  do
5:   if  $\beta_{w_{p_i}} \geq |D_t| \times \mathcal{S}$  then
6:      $L_1 = L_1 \cup p_i$ 
7:     Invoke FindTrail( $\{w_{p_i}\}, w_{p_i}, 2$ )
8:   else
9:     Exit foreach loop
10:  end if
11: end for
12:  $\mathcal{C} = \bigcup_{j=1}^{\ell} C_j$ 
13: for each  $n$ -paset  $x_n \in \mathcal{C}$  do
14:   if  $\alpha_{x_n} \geq |D_t| \times \mathcal{S}$  then
15:      $L_n = L_n \cup x_n$ 
16:   end if
17: end for
18:  $\text{WTB} = \bigcup_{j=1}^{\ell} L_j$ 

```

Procedure FindTrail(paset X , WebTrie node $parent$, int k)

```

1: for each WebTrie tree node  $w_{p_i} \in C(parent)$  do
2:   if  $w_{p_i} \in X$  then
3:     Continue loop with other nodes
4:   end if
5:   if  $\beta_{w_{p_i}} \geq |D_t| \times \mathcal{S}$  then
6:     paset  $Y = X \uplus \{p_i\}$ 
7:     if  $k > 2$  then
8:        $C_k = C_k \cup Y$ 
9:     else
10:       $L_k = L_k \cup Y$ 
11:    end if
12:    Invoke FindTrail( $Y, w_{p_i}, k + 1$ )
13:   else
14:     Exit foreach loop
15:   end if
16: end for

```

Chapter 3. Web Usage Mining

Algorithm 8 Web Traversal Blueprint Discovery 2 Algorithm

Input: A set of traset D_t , a support threshold $\mathcal{S}$ and a WebTrie $\mathcal{W}$

Output: A Web Traversal Blueprint

```

1: Let  $\ell$  be size of the largest traset in  $D_t$ 
2: Initialize set  $L_j = \emptyset$  for  $1 \leq j \leq \ell$ 
3: for each WebTrie tree node  $w_{p_i} \in C(R)$  do
4:   if  $\beta_{w_{p_i}} \geq |D_t| \times \mathcal{S}$  then
5:      $L_1 = L_1 \cup p_i$ 
6:     Invoke FindTrail( $\{w_{p_i}\}, w_{p_i}, 2$ )
7:   else
8:     Exit foreach loop
9:   end if
10: end for
11:  $WTB = \bigcup_{j=1}^{\ell} L_j$ 

```

Procedure FindTrail(paset X , WebTrie node $parent$, int k)

```

1: for each WebTrie tree node  $w_{p_i} \in C(parent)$  do
2:   if  $w_{p_i} \in X$  then
3:     Continue loop with other nodes
4:   end if
5:   if  $\beta_{w_{p_i}} \geq |D_t| \times \mathcal{S}$  then
6:     paset  $Y = X \uplus \{p_i\}$ 
7:     if  $(k == 2) \vee (\alpha_Y \geq |D_t| \times \mathcal{S})$  then
8:        $L_k = L_k \cup Y$ 
9:       Invoke FindTrail( $Y, w_{p_i}, k + 1$ )
10:    end if
11:   else
12:     Exit foreach loop
13:   end if
14: end for

```

“If you want truly to understand something, try to change it.”

KURT LEWIN (1890-1947) Psychologist

“To some people, I am kind of a Merlin who takes lots of crazy chances, but rarely makes mistakes. I’ve made some bad ones, but fortunately, the successes have come along fast enough to cover up the mistakes. When you go to bat as many times as I do, you’re bound to get a good average.”

WALT DISNEY (1901-1966) Film and theme park entrepreneur

4

Clustering

Clustering is the process of grouping physical or abstract objects into classes of similar objects [74]. It is a useful and practical data mining technique with broad applications because of its ability to function with little or no human supervision. Though the first clustering technique dated back to as far as three decades ago, it is only recently that clustering enjoys a revival of interest due to the coming of age of the Web and bioinformatics.

To meet the demands of the information age, researchers analyze traditional clustering techniques in an attempt to adapt them to today’s massive fast-changing data termed *data streams* [11]. Data streams require algorithms that are fast and do not need the data to be ordered and scanned repetitively. Many clustering algorithms have been proposed to meet the demands for speed and scalability but to date, none is able to achieve both timely and quality results; some can only discover spherical clusters [88, 118]; some can discover arbitrary clusters but are slow [160, 4]; while some require the fine-tuning of complex parameters [68, 62].

In this chapter, we propose a novel synergistic approach which combines the capabilities of both clustering and ARM techniques to achieve unprecedented speed-ups without sacrificing quality and ease of configuration. The main idea is to capitalize

on the efficiency of ARM to perform clustering. We term this approach *CLUID* (**CL**ustering **US**ing **IT**emset **D**iscovery). The clustering problem is first mapped into an ARM problem using a semi-automatic discretization technique. Next, any ARM algorithm can be applied to obtain frequent itemsets that represent dense regions. Finally, we introduce a merging algorithm to form clusters from the mined results. CLUID requires only one user-defined parameter to find arbitrary-shaped non-overlapping clusters of varying sizes. It is a combination of density-based, grid-based and resolution-based clustering approaches. For increased efficiency and scalability, we designed a new SOTrieIT variant called the *nSOTrieIT* (*n*-level SOTrieIT) and a tailor-made algorithm for it. Through experiments involving CLUID and DENCLUE [81] using both real and synthetic datasets, the viability and efficiency of this approach are proven.

The rest of the chapter is organized as follows. The next section defines the problem of clustering formally. Section 4.2 surveys the plethora of existing clustering algorithms. Our novel data structure and algorithms are presented in Section 4.3. Section 4.4 shows empirical results involving a variety of synthetic and real datasets. Clustering applications are discussed in Section 4.5 and finally, the chapter is summarized in Section 4.6.

4.1 Problem Definition

The following is a formal statement of the problem of clustering:

- Let $D_o = \{o_1, o_2, \dots, o_n\}$ be a database of objects and $|D_o|$ be its size.
- Let $A = \{a_1, a_2, \dots, a_n\}$ be a set of attributes/features of all objects in D_o and $|A|$ be its size. The set A is analogous to the dimensions of an object.
- Let v_x^y be the value of the feature $a_y \in A$ of object $o_x \in D_o$
- Let $\Phi(o_i, o_j)$ be the similarity function between objects o_i and o_j where $o_i, o_j \in D_o$.
- A cluster c contains objects $(o_i, o_{i+1}, \dots, o_j) \in D_o$ where $1 \leq i \leq j \leq n$.

Chapter 4. Clustering

- Let C_{D_o} be a set of clusters $(c_i, c_{i+1}, \dots, c_j)$ for database D_o .

The problem of clustering is to group objects such that $\Phi(o_i, o_j)$ is maximized for $o_i, o_j \in c$ and $\Phi(o_i, o_k)$ is minimized for $o_i \in c$ and $o_k \in (C_{D_o} - c)$. In short, to achieve high intraclass similarity and low interclass similarity. In practice, the similarity function Φ is usually a distance measure and objects that are closer to each other are considered more similar [74]. Here, without loss of generality, unless otherwise specified, we shall use the Euclidean distance measure as the similarity measure: $\Phi(o_i, o_j) = \sqrt{\sum_{a=1}^{|A|} (v_i^a - v_j^a)^2}$

4.2 Related Work

In this section, existing work in clustering, outlier detection as well as clustering and ARM mergers are briefly discussed and analyzed.

4.2.1 Clustering

We will review prominent clustering algorithms and discuss their strengths and weaknesses in this section.

Partitioning Algorithms

The k -means algorithm is the pioneering algorithm in clustering [103]. It is a partitioning algorithm which iteratively assigns objects to appropriate clusters based on a distance-based similarity measure. The algorithm begins by randomly generating k cluster centers known as *centroids*. An object is assigned to the cluster where the distance between itself and the cluster's centroid is the shortest. After all objects have been grouped into appropriate clusters, the centroids of the clusters are re-computed by taking the average of all member objects. The entire process is reiterated until a certain stopping criterion (stable cluster membership, squared-error criterion etc.) is satisfied. It is fast and easy to implement but suffers from five main weaknesses:

1. Users must correctly specify k .
2. It is very sensitive to outliers.

Chapter 4. Clustering

3. It can only discover spherical clusters.
4. It is sensitive to the initial selection of cluster centers.
5. It is not scalable with respect to both the dimensionality and size of the dataset.

PAM (Partitioning Around Medoids) overcomes the sensitivity of k -means to outliers at the expense of speed [88]. It is similar to k -means except that *medoids* (the most centrally-located object in a cluster) instead of centroids are used to represent the center of gravity in clusters. The use of medoids effectively diminishes the effect of outliers but much computation is needed to determine medoids. CLARA (Clustering LARge Applications) [88] and CLARANS (Clustering Large Applications based upon RANdomized Search) [118] are introduced to improve the performance of PAM by using sampling techniques. However, both of them are highly dependent on the choice and size of samples and do not guarantee an optimal solution.

The X -means algorithm tries to simplify the user's task of selecting an exact k by requesting for a range of values for k instead [128]. It then iteratively searches for the best k value within the range using a model selection criteria such as the *Bayesian Information Criterion* or the *Akaike Information Criterion*. However, it still requires an accurate range of k values and it can be very slow if the user specifies a wide range. Recently, a novel approach is suggested to solve the partitional clustering problem without the need to specify k [119]. It relies on the concept of *facility location* to iteratively adjust k until a facility clustering cost function is minimized [108]. This technique may not be impressive in terms of speed but it significantly improves cluster quality.

There are also other works which attempt to improve the k -means algorithm by optimizing various auxiliary components of it. Constraints for the number of objects in a cluster in order to avoid poorly populated clusters have been proposed [24]. A new initialization method has been introduced to allow k -means to achieve better results [25]. To reduce the sensitivity of k -means to the initialization of centroids, a harmonic-based similarity measure is suggested to replace distance-based measures [168]. To evaluate the choice of k , a measure known as *silhouette coefficient* is introduced

[88]. The silhouette coefficient of a point p of a cluster c specifies how much p truly belongs to c . There are also algorithms which speed up partitioning algorithms in general through the use of sufficient statistics [169] or geometric reasoning [127].

Hierarchical Algorithms

The key idea of the hierarchical approach is to group objects into a tree of clusters by systematically joining small clusters or splitting big clusters. Hierarchical clustering techniques discover a hierarchy of clusters quickly but like k -means, they cannot find arbitrary-shaped clusters. CURE (Clustering Using REpresentatives) solves this problem by representing a cluster by c scattered points instead of using one representative [68]. To handle outliers, CURE shrinks cluster according to an input parameter α . However, CURE still requires users to enter an appropriate k as well as the additional parameters c and α .

The Chameleon algorithm is proposed to reduce the number of input parameters to just one: the k in a k -nearest neighbor model [87]. It uses both dynamic modeling and the k -nearest neighbor method. Chameleon firstly builds tight clusters and then combines them to form bigger arbitrary-shaped clusters using measures that compute the inter-connectivity and relative closeness of two clusters. It improves CURE but is computationally-intensive.

Density-based Algorithms

Density-based methods are faster alternatives for finding arbitrary-shaped clusters. They consider clusters as dense regions of objects separated by sparse regions of objects. DBSCAN (Density Based Spatial Clustering of Applications with Noise) is the pioneering technique in this class of techniques [59]. It needs two input parameters to define what constitutes the neighborhood of an object and whether its neighborhood is dense enough to be considered. It is fast but it is difficult for users to determine the input parameters correctly. This shortcoming is corrected by DBCLASD (Distribution Based Clustering of Large Spatial Databases), which requires no user input [160]. It uses the chi-square test to check whether a cluster has the expected distribution

Chapter 4. Clustering

of distances and decides if it should be connected to the next nearest cluster. It is also considered a grid-based method as it firstly splits the search space into fixed-size partitions.

Another density-based method called DENCLUE (DENSity-based CLUstEring) uses density functions [81]. As DENCLUE is very similar to CLUID in terms of speed and approach and would be used in our experiments, we shall describe it here at length. Its basic idea is that the influence of each object within its neighborhood can be modelled as a mathematical function (Gaussian, square wave etc.) known as the *influence function*. The overall density of the data space is then defined as the sum of the influence function of all objects. Clusters are found by locating the local maxima, known as *density-attractors*, of the overall density function. To speed up computation, the dataset is first divided into d -dimensional hypercubes. This partitioning is dependent on a user-defined influence factor σ . The cubes are then checked to see if they are highly populated (depends on a user-defined density threshold ξ).

Next, the density-attractors for every object in the highly populated hypercubes are determined using a hill-climbing approach. Finally, clusters are formed by objects that are density-attracted to one another. DENCLUE has a good time complexity and is general but requires three input parameters (influence factor σ , density threshold ξ , hill-climbing step size δ) that cannot be easily estimated and it uses a grid-based method initially too. Hence, it suffers from the weaknesses of the grid-based approach which is described next.

Subspace Algorithms

Grid-based/Subspace clustering techniques scale very well with respect to dimensionality. They partition the search space into a number of cells/units and perform clustering on such units. CLIQUE (CLustering In QUEst) combines concepts from grid-based and density-based clustering as well as from ARM [4]. It works well for high-dimensional datasets but requires the user to specify a density threshold and the size of grids. MAFIA (Merging of Adaptive Finite IntervAls) improves upon CLIQUE by computing the grid sizes dynamically [114]. It combines similar grids to

Chapter 4. Clustering

form bigger grids and effectively improves cluster quality and speed. However, it also requires critical input parameters from the user: the cluster dominance factor and merging factor. Both MAFLA and CLIQUE are limited in scalability due to their dependence on Apriori [6]. Finally, grid-based methods may not yield good results in situations where the data points of clusters spread over several neighboring grid cells; the cluster centers are cut by several planes. The OptiGrid algorithm is proposed to address this problem by computing the best partitioning hyperplanes for every dimension [82]. However, it requires the user to correctly specify the number of cutting planes as well as a density threshold.

The notion of *projected clustering* is first proposed by Aggarwal *et al.* to generalize subspace clustering and to discover interesting partitions of data points [1]. A *projected cluster* is defined as a subset C of points with a subset D of dimensions such that the points in C are closely clustered in the subspace of dimensions D . It is argued that projected clustering is suitable for several applications where a partition of the data is required and is more interpretable. PROCLUS (PROjected CLUStering algorithm) is the hill-climbing algorithm introduced to find projected clusters. It is fast but it may fail to discover clusters with large differences in size.

Another highly-related approach is pattern-based clustering, which is introduced lately by Wang *et al.* to relax constraints posed by the similarity measure [148]. In this approach, two objects are considered similar if they share a coherent pattern on a *subset* of dimensions instead of all the dimensions in the traditional case. The main difference between pattern-based and projected clustering lies in the similarity measure: pattern-based clustering uses a more general measure termed *pScore* that measures the similarity between two objects on two attributes. A cluster is one in which all its objects have a pScore has is less than a user-specified cluster threshold. As the search space is exponential with respect to dimensionality, scalability and efficiency are critical issues. Recently, Pei *et al.* presented an algorithm called MaPle (Maximal Pattern-based Clustering) that tackles effectively the above issues [126]. For good results, both pattern-based and projected clustering require the proper setting of the cluster, dimensionality and object thresholds. This poses great difficulty because

Chapter 4. Clustering

of the large number of possible combinations of values and the inability to objectively assess results.

Resolution-based Algorithms

A recent innovation is the resolution-based approach that has been applied successfully on noisy datasets. The basic idea is that when viewed at different resolutions, the dataset reveals different clusters and by visualization or change detection of certain statistics, the correct resolution at which noise is minimum can be chosen. WaveCluster is one such method that uses wavelet transformation to distinguish clusters from noise [138]. Users must first determine the best quantization scheme for the dataset and then decide on the number of times to apply wavelet transform. The TURN* algorithm is another recent resolution-based algorithm [62]. It iteratively scales the data to various resolutions. To determine the ideal resolution, it uses the third differential of the series of cluster feature statistics to detect an abrupt change in the trend. However, it is unclear how certain parameters such as the closeness threshold and the step size of resolution scaling are chosen.

Model-based Algorithms

Another form of clustering uses models that can accurately describe the dataset. Model-based clustering techniques try to find an optimal fit between the input data and a mathematical model. The *Self-Organizing Map* (SOM) [94] and *Auto Classification* (AutoClass) [36] are examples of this approach. The concept of SOMs is based on the observation that various areas of the brain, especially the cerebral cortex, are organized spatially for specific tasks. Each element of a SOM represents a model which is actually a set of numerical values. Every element has an area of influence over nearby elements. When an input arrives, all elements compete to decide who best matches the input and then the most similar element becomes the winner and influences other elements within its neighborhood to match the input better. The SOM is excellent for clustering noisy data but it is time-consuming and requires the fine-tuning of complex parameters. AutoClass uses a Bayesian approach to approxi-

Chapter 4. Clustering

Parameter	Description
k	Number of clusters
n	Number of objects
d	Number of dimensions
γ	Sample size
β	Density threshold
g	Grid size
η	Number of nearest neighbors
$ \mathcal{L} $	Number of cMaximal itemsets
A_x	Time complexity of ARM algorithm A with respect to parameter x

Table 4.1: Common parameters used in clustering

mate the number of clusters [36]. It is an unsupervised classification technique that models classes (clusters) using the finite mixture distribution.

Summary

A good summary can be seen in Table 4.2 where the characteristics of some of the discussed and more relevant clustering algorithms as well as those of CLUID are shown. Common parameters used in the table can be found in Table 4.1. CLUID has two algorithms CLUID1 and CLUID2 which will be elaborated in Section 4.3. From Table 4.2, it can be observed that DENCLUE has the best performance with minimal input parameters and can discover clusters with arbitrary shapes. Although CLUID seems have a higher time complexity than DENCLUE, DENCLUE is much slower because of its high-computational requirements. Comprehensive comparisons and analyses of DENCLUE and CLUID can be found in Section 4.4 where experiments are conducted to ascertain their actual performance in reality. All in all, CLUID2 has good time complexity and can discover arbitrary clusters with just one input parameter. Therefore, it outperforms many existing approaches theoretically.

4.2.2 Outlier Detection

Outlier detection is another means of tackling noise and is closely related to clustering. One classic notion is that of DB (Distance-Based)-outliers [89]. An object is

Chapter 4. Clustering

Algorithm	Parameters needed	Time Complexity	Cluster Shape
k -means	k	$O(dn)$	Spherical
PAM	k	$O((dn)^2)$	Spherical
CLARA	k, γ	$O(dn)$	Spherical
CLARANS	k, η, γ	$O((dn)^2)$	Spherical
CURE	$k, \text{shrinking_factor}, \text{rep_pts}$	$O(dn)$	Arbitrary
CHAMELEON	$\eta, \text{min_size},$	$O((dn)^2)$	Arbitrary
DBSCAN	$\text{radius}, \text{min_pts}$	$O((dn)^2)$	Arbitrary
DBCLASD	-	$O((dn)^2)$	Arbitrary
DENCLUE	$\text{noise_threshold}, \beta$	$O(dn)$	Arbitrary
CLIQUE	g, β	$O(a^d + dn)$	Arbitrary
MAFIA	$\text{cluster/merge_factors}$	$O(a^d + dn)$	Arbitrary
WaveCluster	g, scale	$O(a^d + dn)$	Arbitrary
OptiGrid	$\text{num_planes}, \beta$	$O(dn \log n)$	Arbitrary
TURN*	$\text{closeness_threshold}, \text{step_size}$	$O(dn + A_n A_{ I })$	Arbitrary
CLUID1	$\text{distribution_spread}$	$O(dn + A_n + A_d +$ $ \mathcal{L} ^2 + \mathcal{L} \cdot d)$	Arbitrary
CLUID2	$\text{distribution_spread}$	$O(dn + \mathcal{L} ^2 +$ $ \mathcal{L} \cdot d)$	Arbitrary

Table 4.2: Characteristics of clustering algorithms

considered to be a DB-outlier if a certain fraction f of the dataset lies greater than a distance $\mathcal{D}$ from it. A recent enhancement of it involves the use of the concept of k -nearest neighbors [132]: the top m points with the largest $\mathcal{D}^k$ (distance of the k^{th} nearest neighbor of a point) are treated as outliers. Lately, we have also introduced a noise filtering technique called *FLUID* (**F**iLtering **U**sing **I**temset **D**iscovery) which does not require any user parameters [150]. It first maps the data space to a form suitable for itemset discovery and automatically finds the best support threshold for finding frequent itemsets. Noise, in this case, is defined as objects encompassed by infrequent itemsets.

4.2.3 Clustering and Association Rule Mining Mergers

In this section, we examine works that merge the ideas of both clustering and ARM techniques.

Han *et al.* were the first researchers to embed ARM techniques into clustering [71]. They proposed the use of discovered frequent itemsets from transaction databases to create a *hypergraph* and then use a partitioning algorithm to find clusters. In a hypergraph, the vertices correspond to items in the universal itemset, the edges correspond to the frequent itemsets. Association rules that appear within the clusters are used to discover strongly-connected items or memberships of future transactions. Kusters *et al.* used association rules to find clusters in a population of customers based on the products that were bought by them [95]. Based on a certain selection criterion, the best association rule is selected to construct a cluster such that customers who bought the products that appear in the antecedent of the rule are grouped together. These customers will be removed from the database and the process is repeated until an appropriate stopping criterion is reached.

The text mining domain is another major area where clustering and ARM mergers are explored. Beil *et al.* introduced a technique to use frequent itemset discovery to perform clustering of text documents [16]. Here, an item corresponds to a term/word found in documents. The proposed algorithms allow the discovery of non-overlapping and hierarchical clusters.

Perhaps the work that is most similar to ours is that of Yiu *et al.* where concepts of ARM and projected clustering are merged for the first time [164]. The authors transform the problem of finding the best projected cluster for a medoid into one of finding the best itemset in a transformed search space. The *quality* of an itemset X for a cluster C is measured not by its support but by $\mu(a, b) = a(1/\beta)^b$ where a is the support of X in C , b is the dimensionality of C and β measures the importance of the size of the subspace over the size of C . The best itemset is found using a modified version of FP-growth [75]. The algorithm, termed MineClus, iteratively generates one cluster at a time by finding the best itemset for random medoids until k clusters are

found. Experiments showed the efficiency and robustness of MineClus as compared to its predecessor, the DOC algorithm [131]. However, little has been mentioned on how to assign good values to its parameters, the extent and density threshold of clusters and β . Moreover, MineClus can only discover spherical clusters.

In short, existing clustering and ARM mergers are limited in their applicability because they are designed for specific situations and they require the adjustment of several complex parameters. None can be used for general clustering and could not be easily modified for such a purpose. In the next section, we introduce a novel algorithm that integrates clustering and ARM in an attempt to improve the efficiency and effectiveness of clustering in general with easy and minimal parameter-tuning.

4.3 Algorithms

In this section, we introduce our novel synergistic technique, *CLUID* (**CL**ustering **U**sing **I**temset **D**iscovery). CLUID utilizes the power of ARM to improve clustering. We shall describe its algorithm, propose optimization strategies and examine its complexity. The proper setting of its parameter, the distribution spread ψ is also discussed. Finally, we critically and impartially analyze the strengths and weaknesses of the approach.

4.3.1 Key Ideas

CLUID integrates ideas from density-based, grid-based and resolution-based clustering. Its basic assumption is that interesting clusters have approximately uniform density and outliers are found in regions with lower densities. Here are its pertinent characteristics:

- *Itemset-based*: Objects are first mapped into items using *discretization techniques* and thus, regions are represented by itemsets. Discretization techniques make continuous attributes/features discrete so that ARM can be applied. Here, uniform binning is adopted for discretization. CLUID discovers clusters in the form of frequent itemsets. Therefore, it utilizes the power of existing ARM

Chapter 4. Clustering

algorithms.

- *Density-based*: CLUID perceives clusters as regions of much higher density than regions of noise. By transforming the object space into to transactional space, CLUID removes noisy regions with a minimum support threshold that is used in ARM.
- *Grid-based*: Assuming a Gaussian distribution of values, CLUID partitions values in each dimension appropriately such that each bin corresponds to an item in the transformed transactional space. Hence, the search space becomes a non-uniform multi-dimensional grid.
- *Resolution-based*: Resolution here is controlled in two ways:
 1. *Distribution spread ψ* : This parameter determines the size of the grids in each dimension and thus, controls the distance resolution of the transformed space.
 2. *Minimum support threshold s* : This parameter adjusts the density resolution of the transformed space by removing low-density regions in the form of infrequent itemsets.
- *Minimally-supervised*: The chief strength of CLUID lies in its ease of parameter setting. There is just one parameter, ψ , to be set by the user and usually, after familiarization with the dataset through a few iterations with different ψ values, the user would be able to pinpoint a suitable ψ value for use in future similar datasets. For example, by running CLUID using a range of ψ values, it is possible to discover a set of ψ values for which the results are stable (see Section 4.4). This gives the user a good feel of the nature of such datasets so that for future similar datasets, the range of ψ values to try out is narrower. Many other clustering algorithms would need much more supervision as the wrong settings or combinations of parameter values would lead to drastically inappropriate results.

4.3.2 Algorithm

There is a need to define two special types of itemsets before introducing the algorithm.

Definition 3 (Maximal Itemset) *A frequent itemset X is maximal if there does not exist any frequent itemset Y such that $Y \supset X$.* ■

Definition 4 (cMaximal Itemset) *A maximal itemset X is cMaximal if there does not exist any maximal itemset Y such that $|Y| > |X|$.* ■

Given a d -dimensional dataset D_o consisting of n objects $o_1, o_2, \dots, o_n$, CLUID discovers a set of arbitrary-shaped clusters $C_{D_o} = c_1, c_2, \dots, c_m$ where $m \ll n$ in four main steps with a single user-defined parameter ψ (distribution spread):

1. *Converting*: Convert dataset D_o into a transactional database D_t using algorithm **MapDB**
2. *Mining*: Mine D_t for cMaximal itemsets using algorithm **MineDB**
3. *Merging*: Merge neighboring cMaximal itemsets to form cluiders (to be defined later)
4. *Mapping*: Convert the cMaximal itemsets of cluiders back to their original object form

Steps 1, 2 and 4 are similar to those of FLUID, our previous work on noise filtering, except that FLUID fixed the distribution spread ψ at 0.5% of the dataset size [150]. CLUID also takes a step further to discover clusters in step 3. It is important to allow users to control ψ because it affects clustering quality as seen in Section 4.4. Here, it is assumed that all features/dimensions are equally important and dimensionality reduction is assumed to have been carried out prior to clustering. Therefore, unlike many other grid-based clustering techniques, CLUID only considers cMaximal itemsets in the search for clusters. However, in future, we would certainly consider using CLUID as a tool for dimensionality reduction by using smaller-sized

Chapter 4. Clustering

frequent itemsets. In addition, CLUID can be adapted to perform both projected and pattern-based clustering simply by mining frequent itemsets of various sizes with similar items. A frequent j -itemset ($1 \leq j \leq d$) here represents a dense region in the subspace of j dimensions. For pattern-based clustering, it is necessary only to bring in the notion of similarity using the pScore measure in determining the connectivity of itemsets during the discovery of cluiders (see Definition 7). Such modifications are currently beyond the scope of this work and will be addressed in our future work.

Note that the resultant clusters do not contain the original objects themselves but centroids of the mapped itemsets. An additional step is required if there is a need to determine the actual objects that belong to each cluster. This trivial step simply involves repeating step 1 and comparing the centroids of the mapped objects with the resultant centroids. Objects whose centroids cannot be matched are simply discarded because they represent noise.

Step 1: Converting

Step 1 converts the d -dimensional dataset D_o into a database D_t of transactions of items using discretization techniques. For ease of discussion yet without loss of generality, we apply uniform binning within each dimension in this work, which means that each dimension has a fixed bin size. However, other sophisticated discretization methods may be used depending on the situation. If class labels are available for attributes, entropy-based methods are more effective [55]. Multivariate discretization can also be used to create semantically meaningful intervals [13].

MapDB (Algorithm 9) tries to discretize the features of dataset D_o in a way that minimizes the number of required bins without losing the pertinent structural information of D_o . Every dimension has its own distribution of values and thus, it is necessary to compute the bin sizes of each dimension/feature separately. To approximately determine the data distribution in each dimension, the mean and standard deviation of the closest neighbor distance of every object in every dimension are computed. Assuming that all dimensions follow a Gaussian distribution, an object has at least one neighboring object within three standard deviations of the mean nearest

Chapter 4. Clustering

Algorithm 9 MapDB (CLUID Algorithm)Input: A dataset D_o and a distribution spread parameter ψ Output: A discretized transactional dataset D_t

- 1: Sort each dimension of a d -dimensional dataset D_o in ascending order
- 2: For each dimension x , compute its mean μ_x and standard deviation σ_x of its nearest object distance by checking the left and right neighbors of each object
- 3: Find range of values r_x for each dimension x
- 4: For each dimension x , compute its number of required bins β_x :

$$\beta_x = r_x / ((\mu_x + 3 \times \sigma_x) \times \psi)$$
- 5: Map each bin to a unique integer item $a \in I$
- 6: Convert each object o_i in D_o into a transaction T_i with exactly d items by binning its feature values, yielding a transactional database D_t

neighbor distance.

To avoid having too many bins, there is a need to ensure that each bin contains a certain number of objects. This is accomplished in step 4 using the distribution spread parameter ψ ; the greater the ψ value, the lesser the number of bins. A good estimate for ψ is about 1% of the size of the dataset. Generally, it is necessary to attempt a few ψ values to obtain the best results. This is not a major issue because CLUID executes very fast and thus, even if it is run a few times, it is still faster than many algorithms (see Section 4.4 where CLUID algorithms outperform DENCLUE, a prominent fast clustering algorithm, by more than an order of magnitude). In fact, as mentioned in Section 4.2, most clustering algorithms require a few executions to fine-tune their parameters.

In the event that the values are spread out too widely; i.e., the standard deviation is much larger than the mean, the number of standard deviations used in step 4 should be reduced to 1 instead of 3. Note that if a particular dimension has less than 100 unique values, steps 2-4 are unnecessary and the number of bins is the number of unique values. As mentioned in step 6, each object becomes a transaction with exactly d items as each item represents one feature of the object. The transactions do not have duplicated items as every feature has its own unique set of bins that is represented by a set of unique integer items. This is true even when dimensions have

Chapter 4. Clustering

Object		Attributes			Transaction		Items	
ID	x	y	z		ID			
100	1.5	26	80.8	$\Rightarrow$	100	1	7	13
200	2.4	31	100.9		200	2	12	18
300	2.1	28	97.8		300	2	8	18
400	1.9	26	83.2		400	1	7	13
500	5.0	32	88		500	6	12	15
600	2.3	29	95.1		600	2	10	17
700	2.5	30	97		700	2	11	17
800	3.2	31	87.3		800	3	12	14

Table 4.3: Example of mapping step.

Range	Item	Range	Item	Range	Item
$1.5 \leq x < 2.08$	1	$26 \leq y < 27$	7	$80.8 \leq z < 84.15$	13
$2.08 \leq x < 2.66$	2	$27 \leq y < 28$	8	$84.15 \leq z < 87.5$	14
$2.66 \leq x < 3.24$	3	$28 \leq y < 29$	9	$87.5 \leq z < 90.85$	15
$3.24 \leq x < 3.82$	4	$29 \leq y < 30$	10	$90.85 \leq z < 94.2$	16
$3.82 \leq x < 4.4$	5	$30 \leq y < 31$	11	$94.2 \leq z < 97.55$	17
$4.4 \leq x \leq 5$	6	$31 \leq y \leq 32$	12	$97.55 \leq z \leq 100.9$	18

Table 4.4: Sample mapping table.

similar range of values because different unique integers will be assigned for different dimensions to avoid duplicated items in transactions.

Once D_o is mapped into transactions, it is now in a form that can be mined by any ARM algorithm.

Table 4.3 illustrates how a small 3-dimensional dataset with numerical attributes is converted to a database of transactional items using the mapping shown in Table 4.4. For illustration purposes, we have divided every dimension into uniform bins and map every attribute value to a bin. In practice, according to **MapDB**, different dimensions can have different number of bins. The final mapped database D_t is a transaction database that can be mined by any ARM algorithm. Note that every transaction in D_t has 3 unique items. **MapDB** works similarly for non-numerical attributes except that the number of items allocated for each attribute varies according to its number

Chapter 4. Clustering

of unique values; an item is allocated for each category/value in a categorical/ordinal attribute.

Step 2: Mining

Before we discuss the mining algorithm, here are some formal definitions.

Definition 5 (Connectivity) *A frequent k -itemset*

$X = \{a_1, a_2, \dots, a_k\}$ *is said to be* **connectable** *to another frequent k -itemset* $Y = \{b_1, b_2, \dots, b_k\}$, $X \Omega Y$, *iff* $\left(\forall i |a_i - b_i| \leq 1 \text{ where } (i = 1 \dots k) \right) \wedge \left(\sum_{i=1}^k |a_i - b_i| \leq \mathfrak{S} \text{ where } 1 \leq \mathfrak{S} \leq k \right) \wedge (X \neq Y)$. ■

Definition 6 (Loner) *A frequent itemset X is said to be a* **loner** *if there exists no frequent itemset Y such that $X \Omega Y$ at a threshold $\mathfrak{S}$ where $1 \leq \mathfrak{S} \leq k$.* ■

Definition 7 (Cluider) *A* **Cluider** C_{set} *is a set of connectable frequent itemsets, $C_{set} = \{X_1, X_2, \dots, X_n\}$ s.t. $\forall X_i, X_j \in C_{set}, (X_i \Omega X_j) \wedge (i \neq j)$ at a threshold $\mathfrak{S}$ where $1 \leq \mathfrak{S} \leq k$.* ■

Here, connectable itemsets actually represent adjacent regions in the original data space. The parameter $\mathfrak{S}$ controls the strictness of connectivity; the higher the $\mathfrak{S}$ value, the less strict the criteria for connectivity. In other words, when $\mathfrak{S}$ is set to k , there will be more connectable frequent k -itemsets than if $\mathfrak{S}$ were set to 1. In this work, for ease of discussion, $\mathfrak{S}$ is fixed at $k/2$ and k when determining loners and cluiders respectively. We want to be stricter when finding loners so that more unimportant itemsets can be removed early before the merging phase. However, in Section 4.4, we will see that for high-dimensional datasets, it may be necessary to modify $\mathfrak{S}$ so that clusters can be discovered. In any case, the adjustment of $\mathfrak{S}$ is simple and does not affect computation time.

Definition 8 (Outlying Cluider) *A cluider is said to be* **outlying** *if its size is less than 1% of the size of the largest cluider.* ■

Chapter 4. Clustering

Definition 9 (Outlier) An *outlier* (noise) is identified in three ways here: 1) It resides in the regions represented by infrequent itemsets; 2) It is a representative object that is mapped from a loner; 3) It is a representative object of an outlying cluider. ■

Rationale: An outlier is the result of random noise and thus resides in regions with a low density of objects or small isolated regions. An infrequent itemset here represents a region with few objects and thus, contains outliers. A loner denotes the smallest possible isolated region in the transformed transactional space and hence, contains outliers too. Finally, outlying cluiders represent pockets of high density regions but as they are significantly smaller than normal cluster regions, they are also considered as rare pockets of noise. ■

Lemma 1 (Loner Monotonicity) When the support threshold s is increased, the number of loners decreases or remains the same unless clusters are disintegrated into smaller clusters.

Proof: When s is increased, the number of frequent itemsets is less than or equal to the previous value and the average support count of frequent itemsets is greater than or equal to the previous value because of the need to satisfy the minimum support threshold s . This implies the density of regions represented by the cluiders is greater than or equal to the previous value and the number of outliers that are filtered away is greater than or equal to the previous value. If the clusters are intact, the number of loners will be less than or equal to the previous value as loners represent regions of outliers (Definition 9). If there are more loners, it means that clusters are being broken down into smaller isolated clusters that become loners. ■

Theorem 1 (Optimal Clustering Result) CLUID obtains its optimal clustering result at the point when there are no loners or just before loner monotonicity is violated. ■

Proof: The higher the minimum support threshold s , the smaller the number of outliers (Definition 9). Hence, it is desirable to set s as high as possible to ensure

Chapter 4. Clustering

Algorithm 10 MineDB (CLUID Algorithm)Input: A discretized transactional dataset D_t Output: A set of cMaximal itemsets $\mathcal{L}$

- 1: Set support threshold $s = 0.1$ (10%)
- 2: Let the number of frequent itemsets L be $|L|$
- 3: Given n objects in the original search space, set $k = n$
- 4: **repeat**
- 5: **repeat**
- 6: Use an association rule mining algorithm to discover a set of frequent itemsets L from D_t
- 7: Remove itemsets with less than d (number of dimensions) items from L to obtain a set of cMaximal itemsets $\mathcal{L}$
- 8: Adjust s using a variable step size to bring $|\mathcal{L}|$ closer to k
- 9: **until** $|\mathcal{L}| = k$ or $|\mathcal{L}|$ stabilizes
- 10: Set $k = \frac{1}{2}|\mathcal{L}|$
- 11: Set $s = 0.1$
- 12: Remove loners from $\mathcal{L}$
- 13: **until** the number of loners increases or becomes zero

that the final clustering result contains minimal noise. It is then obvious that when there are no loners, s has the value where noise is eliminated. Another important cue is given by loner monotonicity (Lemma 1). When loner monotonicity is violated, it signifies that the support threshold s is increased to the point when inherent clusters start to break up. Therefore, the value of s is optimal just before the point when it becomes too high and breaches loner monotonicity. ■

It is now appropriate to introduce the mining algorithm **MineDB** (Algorithm 10). The key idea is to discover the optimal set of frequent itemsets that represents the important characteristics of the original dataset. We consider important characteristics as dense regions in the original dataset. In this case, the support threshold s is akin to the density threshold used by density-based clustering algorithms and thus, it can be used to remove regions with low density (itemsets with low support counts). The crucial point here is how to automate the fine-tuning of s . This is done by checking the number of *loner* itemsets after each iteration (steps 6-12) and applying Theorem

Chapter 4. Clustering

1. Hence, if the number of loners increases, it signifies that s is too high and the previous s value represents an important turning point where clusters still remain intact. This point is made more evident in Section 4.4 where the resultant clusters can be visually observed.

In step 7, smaller frequent itemsets are removed from L to yield cMaximal itemsets automatically. There is no need to compare the remaining itemsets with one another to confirm if they are maximal as each item of an itemset represents a dimension and each dimension has its unique set of integers denoting its bins. Obviously, this means that much effort is wasted in finding unnecessary frequent itemsets. However, ARM algorithms cannot find cMaximal itemsets directly but must iteratively build them from smaller frequent itemsets or scan the database repeatedly (even algorithms that find maximal itemsets directly need to do so [31]). In the next section, we shall introduce a fast technique to discover cMaximal itemsets directly.

The number of desired frequent cMaximal itemsets (frequent itemsets with exactly x items), k , is initially set to the size of the original dataset as seen in step 3. The goal is to obtain the finest resolution of the dataset that is attainable after its transformation. The algorithm then proceeds to derive coarser resolutions in an exponential fashion in order to quickly discover a good representation of the original dataset that retains the inherent clusters with minimal noise. This is done at step 10 where k is being reduced to half of $|L_x|$. The amount of reduction can certainly be lowered to get more resolutions but this will incur longer processing time and may not be necessary.

In step 8, the support threshold s is incremented/decremented by a variable step size. The step size is variable as it must be made smaller in order to zoom in on the best possible s to obtain the required number of cMaximal itemsets, k . In most situations, it is quite unlikely that $|L_x|$ can be adjusted to equal k exactly and thus, if $|L_x|$ stabilizes or fluctuates, its closest approximation to k is considered as the best solution as seen in step 9. As ARM needs to be performed with several values of s , MineDB will benefit greatly from ARM algorithms that are support-independent to a certain extent such as our FOLD-growth algorithm [157].

Chapter 4. Clustering

Large itemsets	
L_1	$\{1\}, \{2\}, \{7\}, \{12\}, \{13\}, \{17\}, \{18\}$
L_2	$\{1, 7\}, \{1, 13\}, \{2, 17\}, \{2, 18\}, \{7, 13\}$
L_3	$\{1, 7, 13\}$

Table 4.5: Discovered frequent itemsets.

For the dataset in Table 4.3, the frequent itemsets discovered at $s = 25\%$ (support count of 2) are shown in Table 4.5. The only cMaximal itemset (L_3) is $\{1, 7, 13\}$.

Step 3: Merging

With the set of cMaximal itemsets found, it is time to merge them together to form cluders (Definition 7) based on their connectivity (Definition 5). For example, the itemset $\{8, 15, 24, 35\}$ is connectable to the itemsets $\{7, 15, 24, 36\}$ and $\{7, 16, 23, 35\}$ and they all belong to the same cluder (Definition 7). The algorithm for discovering cluders from cMaximal itemsets is trivial and will not be discussed here. It simply involves scanning all cMaximal itemsets and merging all connectable maximal itemsets. For our sample dataset in Table 4.3, there is only one cluder with the cMaximal itemset $\{1, 7, 13\}$.

This merging technique is similar to techniques used in grid-based clustering except that it is performed as a postprocessing step. Grid-based methods usually employ an apriori-based approach to generate and test candidate cells and thus, their performance is similar to that of the Apriori algorithm which is limiting due its need for candidate generation (see Chapter 2). In addition, such methods must modify certain internal functions of ARM algorithms to suit the objectives of clustering. In our approach, there is no need to modify ARM algorithms as they can be plugged in seamlessly. This allows us to tap into the findings of the latest in ARM research instead of being confined to one particular ARM algorithm.

Step 4: Mapping

Definition 10 (Representative Object) A *representative object* is obtained when an itemset X is mapped back to its original object form. Each item here rep-

Chapter 4. Clustering

resents a value in a dimension and thus, mapping is done by assigning every item $I \in X$ to the center value of the interval it represents. ■

Definition 11 (Cluster) A *cluster* is a set of representative objects of a cluider. ■

The last step of CLUID is the simplest: it involves mapping the cMaximal itemsets back to their original object form.

The clusters formed by representative objects have arbitrary shapes and sizes because cMaximal itemsets may be connected in an arbitrary manner as long as they are close to each other. Unfortunately, the final clustering results may not be perfect due to the loss of some information during the mapping step. The following are definitions of possible errors.

Definition 12 (Rogue cMaximal Itemset) A *rogue cMaximal itemset* is an cMaximal itemset that represents a region with both data and noise due to discretization during the mapping phase. ■

Definition 13 (Positive Merging Error) The *positive merging error (PME)* refers to the probability of merging 2 clusters when they are supposed to be separate clusters. This occurs when one or more rogue cMaximal itemsets connects cMaximal itemsets of the 2 clusters. $PME = |\mathcal{L}| / \sum_{i=1}^d \beta_i$ where β_x is the number of bins in dimension x . ■

Definition 14 (Negative Merging Error) The *negative merging error (NME)* refers to the probability of not merging 2 clusters when they are supposed to be in the same cluster. This occurs when a cluster does not have uniform density and an itemset representing a critical adjoining low-density area does not become frequent. $NME = 1 / (\text{expected uniformity of cluster density})$. ■

Definition 15 (Mapping Residual Error) The *mapping residual error $\mathcal{E}$* of the clustering results of CLUID is the probability of discovering wrong clusters and is given by: $\mathcal{E} = PME + NME$ ■

Figure 4.1: Example of Rogue cMaximal itemset regions.

The mapping residual error $\mathcal{E}$ is an inherent problem of CLUID that can be controlled to a certain extent by minimizing PME. One obvious naive solution is to increase the number of bins by reducing ψ but this increases computation time and limits scalability. In addition, NME is beyond our control at this moment and depends on the nature of the dataset. It is part of our ongoing work to try to eliminate or minimize the mapping residual error.

Figure 4.1 shows four 2-dimensional cells, A, B, C and D and cells B and C are regions where rogue cMaximal itemsets are found. Suppose that cells B and C contain data points of 2 distinct clusters, these 2 clusters will be merged because cells B and C correspond to connectable cMaximal itemsets. But if after discretization, the cell with dotted line is created instead and it contains only noise objects, then the 2 clusters will be separated properly.

4.3.3 Complexity Analysis

The following are the time complexities of the four main steps of CLUID:

1. *Converting*: The time complexity of the sorting step of MapDB is $|D| \cdot d$. The rest of the algorithm has similar complexity and thus its overall complexity is still $|D| \cdot d$.
2. *Mining*: Let the time complexities of the ARM algorithm A used in the ARM step with respect to the dataset size and dimensionality be $A_{|D|}$ and A_d respectively. The time for the identification of loners is $|\mathcal{L}|^2$. Since MineDB iteratively

Chapter 4. Clustering

executes the above 2 steps, its overall complexity is $A_{|D|} + A_d + |\mathcal{L}|^2$.

3. *Merging*: The time for this straightforward step is $|\mathcal{L}|^2$ as every cMaximal itemset must be compared with each other to see if they are connectable.
4. *Mapping*: As every item of the resultant cMaximal itemset needs to be mapped, its time complexity is $|\mathcal{L}| \cdot d$.

Hence, the overall time complexity of CLUID is $O(|D| \cdot d + A_{|D|} + A_d + |\mathcal{L}|^2 + |\mathcal{L}| \cdot d)$.

4.3.4 Optimization Strategies

The main bottleneck of CLUID lies in MineDB, which suffers from two main weaknesses:

1. *Error in setting $k = \frac{1}{2}|L|$* : It is possible to miss out critical k values that lie between $|L|$ and $\frac{1}{2}|L|$ at every iteration.
2. *Generation of unnecessary frequent itemsets*: Frequent itemsets with less than d items are unnecessarily generated in the process of finding d -itemsets.

We introduce **MineDB2** in this section to overcome the weaknesses of MineDB. For clarity, we term the algorithms that use MineDB and MineDB2, CLUID1 and CLUID2 respectively. A new data structure, a variant of the SOTrieIT [51], called the **nSOTrieIT** (n-level Support Ordered Trie Itemset) is proposed here. It is used by MineDB2 to store the transaction database D_t in a way that allows cMaximal itemsets to be found in a fast and straightforward manner without the use of any ARM algorithm.

Definition 16 (nSOTrieIT) *A nSOTrieIT consists of multiple levels of tree nodes such that every tree node w has a label (represents an item) and an integer (represents support count). (As every tree node corresponds to some item $a_i \in I$, for brevity, we use w_i to refer to a tree node that corresponds to $a_i \in I$.) Let $P^l(w_i)$ be the l^{th} parent node of w_i . If w_i is a n-level node, let $\text{path}(w_i)$ be a set of items that corresponds to the nodes of $w_i, P(w_i), P^2(w_i), \dots, P^n(w_i)$ where $P^n(w_i) = \emptyset$. Let $S(w_i)$ is the*

Chapter 4. Clustering

Algorithm 11 Construction Algorithm of the nSOtrieITInput: A discretized transactional dataset D_t Output: A nSOtrieIT $\mathcal{S}$

```

1: Let  $\mathcal{S}$  be a nSOtrieIT
2: for each transaction  $T \in D_t$  do
3:   Set node pointer  $p$  to ROOT of  $\mathcal{S}$ 
4:   for each item  $i \in T$  do
5:      $p = \text{AddChild}(p, i)$ 
6:   end for
7: end for

```

AddChild(nSOtrieIT $node$, Item i)

```

1: if  $node$  has child node  $w_i$  then
2:   Increment  $S(w_i)$  by one
3:   Order nodes by support count in descending order
4:   Return  $w_i$ 
5: else
6:   Create new child node  $w_i$ 
7:   Set  $S(w_i) = 1$ 
8:   Return  $w_i$ 
9: end if

```

support count node w_i . Let $C(w_i)$ be the set of child nodes of node w_i . If $C(w_i) \neq \emptyset$, then $C(w_i) \subseteq \{w_j, \dots, w_N\}$ where $j > i \wedge S(w_j) \geq S(w_{j+1})$, i.e., the child nodes are sorted by support counts in descending order from left to right. ■

A set of nSOtrieITs, possibly rooted at $w_1, w_2, \dots, w_{|I|}$, is built from a transaction database D_t with a universal itemset I . We use a special node called *ROOT* to link all these nSOtrieITs together and keep them ordered by support count as well. For brevity, we shall refer to a set of nSOtrieITs as simply nSOtrieIT henceforth.

The nSOtrieIT is constructed from the transaction database D_t with just a single pass using Algorithm 11. Every item in every transaction is used once to populate the nSOtrieIT with algorithm **AddChild** (steps 4-5). Note that in step 3 of **AddChild**, the nodes are sorted. Fortunately, this sorting step is not computationally expensive as the support count of any node is incremented only by one (step 2), which means

Figure 4.2: A nSOTrieIT Example.

that at any one time, only two nodes need to be swapped to maintain the correct order. The construction process is fast, simple and straightforward and the resultant nSOTrieIT is at most the size of D_t .

Figure 4.2 shows an nSOTrieIT constructed from our sample dataset in Table 4.3. The number beside a node is its item value and the bracketed number is its support count. Ignore the arrows and numbers above them for now. There are a total of 17 nodes while the dataset has a total of $(3 \times 8 = 24)$ items. This is because nodes may contain information of multiple transactions if their items are similar. Observe that the nodes are all sorted in support descending order at every level.

Theorem 2 (nSOTrieIT Leaf cMaximality) *The support counts of the leaf nodes of the nSOTrieIT are the support counts of the cMaximal itemsets they represent. ■*

Proof: A transaction is converted from a d -dimensional object and thus has exactly d items. Since cMaximal itemsets are the largest and most encompassing frequent itemsets, they have exactly d items as well. The nSOTrieIT is constructed using every item of every transaction and every item represent a level. Hence, a leaf node of the nSOTrieIT represents a itemset with exactly d items. Every dimension has its unique set of representative integers, which means that there are no duplicated items among different levels of the nSOTrieIT. This implies that every path to the leaf node of a nSOTrieIT is unique and cannot be found in other parts of the nSOTrieIT. Therefore,

Chapter 4. Clustering

it follows that the support count of a leaf node is the actual support count of the cMaximal itemset it represents. ■

Algorithm 12 MineDB2 (CLUID Algorithm)

Input: A discretized transactional dataset D_t

Output: A set of cMaximal itemsets $\mathcal{L}$

- 1: Build a nSOTrieIT $\mathcal{S}$ with Algorithm 11
- 2: Set stepsize δ to a very small value
- 3: Set support threshold $s = 0$
- 4: **repeat**
- 5: Find cMaximal itemsets $\mathcal{L}_k$ with
 MineNSOTrieIT(ROOT, $s \times |D|$) where k is the iteration count
- 6: **if** ($\mathcal{L}_k == \mathcal{L}_{k-1}$) **then**
- 7: Set $s = s + \delta$
- 8: Increase δ if necessary to avoid entering here too frequently
- 9: Continue with the loop at step 4
- 10: **end if**
- 11: Count and remove loners from $\mathcal{L}_k$
- 12: **until** the number of loners increases or becomes zero

MineNSOTrieIT(Node parent, int minCount)

- 1: **for each** child node w_i of parent **do**
 - 2: **if** $S(w_i) \geq \text{minCount}$ **then**
 - 3: **if** w_i is a leaf node **then**
 - 4: Create an itemset X
 - 5: Set its support count $\sigma_X = S(w_i)$
 - 6: Populate it with items from $\text{path}(w_i)$
 - 7: $\mathcal{L}_k = \mathcal{L}_k \cup X$
 - 8: **else**
 - 9: **MineNSOTrieIT**($w_i, \text{minCount}$)
 - 10: **end if**
 - 11: **else**
 - 12: Exit
 - 13: **end if**
 - 14: **end for**
-

Based on Theorem 2, MineDB2 (Algorithm 12) is able to use the nSOTrieIT to discover cMaximal itemsets without any need for database scanning or association rule

mining algorithms. At the fifth step, MineDB2 recursively traverses the nSOTrieIT with algorithm **MineNSOTrieIT** to find cMaximal itemsets. **MineNSOTrieIT** is fast due to its ability to avoid traversing unnecessary branches of the nSOTrieIT. Level-wise traversal stops the moment a node is found whose support count is below the minimum support count (step 2). In Figure 4.2, we shall witness such efficient pruning of the search space. The arrows in the figure represent the traversals of **MineNSOTrieIT** and the numbers on them represent the sequence number. Although there are 17 nodes, only 8 moves are necessary to discover all the cMaximal itemsets with a minimum support count of 2. One example of pruning can be seen in the third move. When node 17(1) is found, **MineNSOTrieIT** skips the rest of its siblings because node 17(1) has already failed to meet the minimum support count requirement and it is supposed to be the node with the highest support count among its siblings.

Unlike MineDB, MineDB2 chooses the resolution of the search space by adjusting the support threshold by small steps (steps 7-8). This inevitably leads to more iterations (cushioned by the speed gained through the use of the nSOTrieIT) but ensures greater completeness. Hence, MineDB2 successfully overcomes the weaknesses of MineDB and in Section 4.4, this fact is empirically proven. The time complexity of MineDB2 is determined by the time to build and traverse the nSOTrieIT as well as the time to determine loners. As the nSOTrieIT has at most $|D| \times d$ nodes, both its construction and traversal times have a complexity of $O(|D| \cdot d)$ each. All in all, the time complexity of MineDB2 is $O(|D| \cdot d + |\mathcal{L}|^2)$ and thus CLUID2 has an overall time complexity of $O(|D| \cdot d + |\mathcal{L}|^2 + |\mathcal{L}| \cdot d)$.

4.3.5 Evaluation Issues

To objectively and correctly evaluate the clustering results for the fine-tuning of ψ , especially in situations where the dataset has more than three dimensions, validity indices could be used though they are quite inadequate. The *silhouette coefficient* specifies how much an object truly belongs to a particular cluster [88]. It gives a good indication of how suitable the chosen number of clusters is. A recent clustering validity index called *S_Dbw* is proposed to determine the ideal number of clusters for

the optimal partitioning of a dataset [69]. A low S_Dbw value indicates compactness and good cluster separation. It is defined as the sum of the following:

- *Inter-cluster Density (Dens_bw)*: Average density in the region among clusters in relation with the density of the clusters — indicates how well-separated clusters are
- *Intra-cluster variance (Scat)*: Average scattering for the clusters — indicates how compact clusters are

The number of clusters k that minimizes S_Dbw is the optimal number of clusters for the dataset. Unfortunately, the above indices do not work in situations where non-convex or arbitrary-shaped clusters are expected to be present. Hence, we do not use them in our experiments because it is highly likely that real datasets contain clusters with sophisticated shapes. Until the development of a general validity index for all types of clusters, it is still difficult to objectively analyze the quality of clustering results. One possible way is to use *multidimensional scaling* (MDS) to visualize the clusters and see if they are well-separated [96]. MDS is a family of methods for analyzing and visualizing high-dimensional proximity data. Its goal is to map objects to a lower dimensional space in a way that approximates well the dissimilarities (distances in our case) between objects. We will use MDS to evaluate our clustering results for a high-dimensional dataset in Section 4.4.

4.3.6 Strengths and Weaknesses

This section serves as a summary to our discussion on CLUID. CLUID's numerous strengths are fundamentally inherited from grid-based clustering, density-based clustering and association rule mining:

1. It discovers clusters of arbitrary shapes and sizes.
2. It discovers clusters of different sizes.
3. It does not require the user to input the number of clusters.
4. It is insensitive to outliers.

Chapter 4. Clustering

5. It does not depend on the ordering of objects.
6. It does not scan the dataset repeatedly.
7. It is fast because it capitalizes on the efficiency of ARM algorithms. In addition, consistent speed improvement is expected due to intense research interest in ARM.
8. It is scalable with respect to both the dimensionality and size of the dataset.
9. It does not require the adjustment of many complex parameters as only the distribution spread ψ is needed.

The main strength of CLUID is its independence on user-supplied parameters. Unlike its predecessors, CLUID requires minimal human supervision. Not only does it remove noise/outliers, it *compresses* the dataset into a set of representative points without any loss of pertinent structural information of the original dataset. In addition, it is reasonably scalable with respect to both the size and dimensionality of the dataset as it inherits the efficient characteristics of existing ARM algorithms. Hence, it is an attractive preprocessing tool for clustering, neural network training or other data mining tasks.

Ironically, its weakness also stems from its use of ARM techniques. This is because ARM algorithms do not scale as well as resolution-based algorithms in terms of dataset dimensionality. Fortunately, since ARM is still receiving much attention from the research community, it is possible that more efficient ARM algorithms will be available to CLUID to give speed-ups at no cost. This is the unique strength of CLUID: it allows ARM algorithms to be plugged in seamlessly without alteration.

Fortunately, efficiency and scalability are greatly improved with the use of the nSOTrieIT in CLUID2 to discover cMaximal itemsets directly. The mapping residual error is another notable weakness that is still undergoing research.

Symbol	Description	Size	Dimensionality
D_1	Synthetic dataset	10,000	2
D_2	Postal addresses of three metropolitan areas	123,593	2
D_3	Color histogram of a Corel image collection	68,040	32
D_4	Isolet Spoken Letter Recognition dataset	6,238	617

Table 4.6: Synthetic and real datasets used in the experiments.

Figure 4.3: Real Dataset D_2

4.4 Experiments

This section evaluates the performance and practicality of the CLUID approach to clustering by conducting experiments on a Pentium-4 machine with a CPU clock rate of 2.4 GHz, 1 GB of main memory and running on a Windows 2000 platform. For CLUID1, we shall use FOLD-growth (see Chapter 2) as its ARM algorithm as it is fast, incremental and scalable [157]. Note that the preprocessing time needed by FOLD-growth to build SOTrieITs is not taken into consideration here due to amortization [157]. As we are unable to obtain the original DENCLUE program from the authors, we have implemented it to the best of our understanding. All algorithms are coded in Java.

A total of four datasets are used in the experiments, of which one is synthetic and the rest are real. Table 4.6 gives a brief summary of the datasets. The synthetic dataset D_1 tests the ability of CLUID to discover clusters of various sizes and shapes

Chapter 4. Clustering

amidst much noise. It has been used as a benchmarking test for several clustering algorithms [62]. Experiments reveal that prominent algorithms like k -means [103], DBSCAN [59], CHAMELEON [87] and WaveCluster [138] are unable to properly find the nine visually-obvious clusters and remove noise even with exhaustive parameter adjustments [62]. Only TURN* [62] manages to find the correct clusters but it requires user-supplied parameters as mentioned in Section 4.2.1. Figure 4.4(a) shows the D_1 with 10,000 points in nine arbitrary-shaped clusters interspersed with random noise. The second database D_2 contains 123,593 postal addresses of three metropolitan areas (New York, Philadelphia and Boston) with a lot of noise in the form of rural areas [147]. It is our largest dataset (in terms of number of objects) and is displayed in Figure 4.3. The third database D_3 is image data obtained from a Corel image collection [79] while the last database D_4 is noisy speech data collected from an experiment involving the recognition of letters of the alphabet [20]. They are both used to test the effect of dimensionality on CLUID. Note that due to time constraints, we have limited the running time of each experiment to two hours.

The progressive and final results of executing CLUID on D_1 with a distribution spread $\psi = 50$ can be observed in Figure 4.4. CLUID stops with the solution found in Figure 4.4(d). This is because in subsequent iterations, the number of loners begins to increase, which implies that the inherent clusters are being disintegrated (see Theorem 1). This effect can be clearly seen in Figures 4.4(e) and (f). Both CLUID1 and CLUID2 are able to obtain the same ideal result (9 clean clusters) but CLUID2 (took 1s) is found to be six times faster than CLUID1 (took 6s) because it does not need to discover unnecessary frequent itemsets. The results of running DENCLUE on D_1 with various parameter values can be seen in Figure 4.5. As the creators of DENCLUE did not clarify how to efficiently and correctly merge dense regions, we simply skip this step and just show the individual clusters, which means that DENCLUE will certainly take a longer time than what we observe here. In addition, the choice of values for its parameters is unclear as well and thus, we have tried several combinations and found that the set in Figure 4.5(b) is the best combination in terms of both speed and quality. However, it is still 7 times slower

Chapter 4. Clustering

Figure 4.4: Results of executing CLUID on D_1 with $\psi = 50$.

than CLUID2. Since CLUID2 has only one parameter to fine-tune, it is more practical (lesser combinations to try) than DENCLUE. Moreover, as we note in Figure 4.6, it is quite easy to pin down the optimal ψ value because it yields stable results for a wide range of values ($40 \leq \psi \leq 70$).

Unlike D_1 , D_2 contains much more noise and visually, it has only one cluster. It is thus used to test the scalability of the algorithms. Both CLUID1 and CLUID2 took about 300s to cluster D_2 ; however, CLUID1 needed 900s to build its SOTrieIT while CLUID2 needed only a second to build its nSOTrieIT. DENCLUE took more than two hours to handle D_2 . This is probably due to its need to compute density-attractors

Chapter 4. Clustering

Figure 4.5: Results of executing DENCLUE on D_1 with varying σ, ξ, δ . Figure 4.6: Effect of ψ on D_1

for every object in highly populated hypercubes, which involves a nondeterministic hill-climbing procedure. We found it very difficult to set a suitable step size for the hill-climbing procedure; if it is set too high, the density-attractor may be wrongly determined and if it is set too low, it may take too much time.

Both CLUID1 and DENCLUE could not complete their tasks within two hours for the high-dimensional datasets D_3 and D_4 . CLUID1 spends too much time generating a huge number of frequent itemsets before it can obtain the cMaximal itemsets. DENCLUE fails for the same mentioned reasons and moreover, since it uses a single σ

value to partition all dimensions into hypercubes, it is highly likely that some dimensions may have too many/little bins and this affects computation time and results. Only CLUID2 succeeds in clustering D_3 with a maximum time of 800s regardless of its distribution spread ψ . However, we need to set $\mathfrak{S}$ to 1 when determining the connectivity of itemsets because higher $\mathfrak{S}$ values result in single-cluster solutions. After removing loners, the remaining cMaximal itemsets are found to be very *close* to each other. This is probably due of the nature of the dataset, which contains objects with several similar attribute values. CLUID’s result for D_3 can be visually seen in Figure 4.7 where MDS is used to reduce dimensionality to just three. We use the XGvis system to perform MDS on D_3 [30]. As it is difficult to capture the 3-dimensional image, we include two particular snapshots of it at different angles to show all the clusters. It is clear that the ten discovered clusters are quite well-separated and have varying sizes and shapes — another testimony of the ability of CLUID to discover clusters with arbitrary shapes and sizes.

CLUID2 took about 300s to execute for D_4 but is unable to find any cluster in D_4 because both the number of loners and cMaximal itemsets suddenly drop from 6,238 to zero with a slight increase in support threshold regardless of whatever values we set for ψ and $\mathfrak{S}$. This is mainly due to what is commonly known as the *dimensionality curse*, where the huge number of dimensions causes the data space to be sparsely populated. In such cases, dimensionality reduction techniques should be applied first before CLUID is executed. Alternatively, CLUID could be modified for projected or pattern-based clustering to overcome this problem.

All in all, it is empirically proven that CLUID2 clearly outperforms CLUID1 and DENCLUE in terms of speed and cluster quality but much work is still needed before CLUID2 can be effectively and directly used for very high-dimensional datasets.

4.5 Applications

Clustering has broad applications in diversified areas where scalability is critical (abundance of data), optimal user parameters are impossible to predetermined (un-

Figure 4.7: Results of CLUID2 with $\psi = 600$ on D_3 visualized using multidimensional scaling.

known or previously unencountered situations) and clusters have arbitrary shapes - areas where CLUID has an edge over existing algorithms. The following are just some of the more prominent applications:

- *Customer Segmentation* [58]: Understanding customer segments is critical for customer-focused businesses. Clustering can be used to discover naturally occurring groups within the customer database. With a good segmentation of customers, it is then possible to vary marketing strategies accordingly to maximize profitability.
- *Spatial Data Analysis* [118]: Spatial data is in abundance in the form of satellite and medical images. Very often, it is not known what form of patterns could be found in such images and hence, clustering methods would come in handy here by automatically extracting interesting spatial patterns and features. For example, in a clustering can be used to separate magnetic resonance imaging data of the brain into tissues of white matter, gray matter, and cerebral spinal fluid automatically [170].
- *Personalization of Web Pages* [111]: As seen in the previous chapter, web page personalization is important for electronic commerce. However, WUM may be limited in scalability because of the huge number of users. Here, clustering can be used to cluster user transactions to obtain aggregating user profiles first

before recommender systems take over. This allows personalization to be done in real time.

- *Bioinformatics* [57, 146]: Bioinformatics is one important area that can benefit tremendously from clustering as biological data is huge and largely unknown. In gene expression analysis, clustering helps to automatically find patterns among genes as well as samples. This allows the identification of critical genes that affect diseases and the understanding of functions of unknown genes.

4.6 Summary

The importance of clustering in our information age is undisputed. As data becomes more abundant, it is necessary to design more efficient and effective clustering techniques. We have reviewed and realized that the current arsenal of clustering methods is unable to cope with data streams. Since association rule mining (ARM) has received intense research interest, we propose the integration of ARM techniques, particularly the fast discovery of frequent itemsets, into clustering. The end result is a novel algorithm called CLUID. Experiments reveal that CLUID can discover clusters of varying sizes and shapes. In addition, an optimized version of CLUID, which uses a new SOTrieIT variant called the nSOTrieIT, delivers impressive performance in terms of scalability and speed as compared to prominent existing techniques. Finally, some interesting applications of clustering are discussed.

“Success is the ability to go from one failure to another with no loss of enthusiasm.”

WINSTON CHURCHILL (1874-1965) British prime minister

“Where there is an open mind, there will always be a frontier.”

CHARLES KETTERING (1876-1958) Inventor

5

Conclusions

Data mining is a time-consuming and complex process. With advances in information collection and storage, there is now abundant data for mining and this makes data mining even more computationally-intensive with terabytes of information to sieve through. Hence, in this work, we seek to address this problem and successfully enhanced the efficiency of *association rule mining* (ARM), *web usage mining* (WUM) and clustering. In this concluding section, we shall summarize our work and discuss new research directions.

5.1 Thesis Summary

We have developed three new data structures:

1. *SOTrieIT*: The SOTrieIT is a dual-level trie data structure that stores itemset information and is ordered by support counts.
2. *WebTrie*: The WebTrie is a SOTrieIT variant that stores support counts of web pages.
3. *nSOTrieIT*: The nSOTrieIT is a multi-level SOTrieIT that stores support counts of cMaximal itemsets.

We have also developed several algorithms that exploit the above data structures to address speed, scalability and reusability issues effectively. The following are three main algorithms:

1. *FOLD-growth*: FOLD-growth uses the SOTrieIT to quickly discover frequent itemsets of lengths one and two so that infrequent items can be pruned away from the transactions. Next, FOLD-growth builds FP-trees and uses them to discover frequent itemsets. FOLD-growth achieves impressive speed-ups against FP-growth because it uses less FP-tree nodes due to early pruning.
2. *TRALOM*: TRALOM consists of sessionization, transactionization and mining algorithms. Its sessionization and transactionization algorithms ensure that the final discovered knowledge is meaningful and useful while its sequential mining algorithm uses the WebTrie to quickly sequences of frequently-accessed web pages with high efficiency.
3. *CLUID*: CLUID is a synergistic integration of ARM and clustering ideas. It firstly discretizes the object space and transforms it into a transactional database. Next, any ARM algorithm can be used to discover frequent itemsets, which represent dense regions in the object space. An optimized mining algorithm is developed to use the nSOTrieIT to directly and rapidly discover cMaximal itemsets. Finally, the results are mapped back to the object space, yielding clusters of varied shapes and sizes.

5.2 Future Research Frontiers

In this section, a few highly promising research frontiers are proposed in the areas of ARM, WUM and clustering.

5.2.1 Association Rule Mining

Electronic commerce is touted to be the killer domain for data mining [93]. However, more research in ARM is needed before it can be successfully applied to this domain. In the above paper, an example is given about the use of ARM in building customer

Chapter 5. Conclusions

profiles and it is found that 96% of the discovered rules are discarded through a filtering process by experts. Therefore, much time is wasted in trying to extract thousands of useless rules. The usefulness of ARM in electronic commerce is beyond doubt but much work is necessary to make ARM more focused and dedicated to this specialized domain. Variants of association rules should be explored to attune ARM to the idiosyncrasies of electronic commerce. Some possible variants include:

- *Evolving association rules*: Due to the constant changing nature of electronic commerce data, it may not be very useful to obtain association rules of a particular snapshot of the database. Instead, more useful knowledge may be derived from association rules that evolve with time. Evolving association rules reveal trends over time. For instance, the items bought by customers during the four seasons of the year follow certain trends. In such a scenario, an example of a useful evolving association rule may be that if a customer buys suntan lotions and sunglasses during summer, he will probably buy moisturizing creams and lip glosses during winter.
- *Association rules with multiple support thresholds*: Using just one single support threshold usually results in a huge number of short association rules (rules with few items) and a few long association rules (rules with many items). Sometimes, it is desirable to have equal number of short and long association rules. Consider an online store selling computer hardware. Useful short association rules pertain to computer peripherals like printers and scanners; printers and ink cartridges are usually bought together. Long association rules pertain to complete computer systems; AMD processors, Maxtor hard disks, Creative sound cards and Plextor CD-writers are usually bought together. The above rules can only be obtained if multiple support thresholds are used. One obvious solution is to modify the support threshold s according to the length l of the desired association rule but it is difficult to determine a suitable relationship between s and l .
- *Association rules with additional thresholds*: Additional thresholds like prof-

Chapter 5. Conclusions

itability and availability thresholds would be useful to prune away more association rules that are not relevant. For example, if certain items in an association rule have narrow profit margins and low availability, then it is desirable to drop the rule.

Finally, the mining of association rules in data streams poses big challenges not tackled in this thesis such as mining with only a single database scan and using limited memory to store critical data.

5.2.2 Web Usage Mining

We have proposed WALMAGE, an evaluation framework for WUM algorithms, which takes into consideration, the cognitive nature of browsing behaviour as well as the structure of a web site. We contend that it is better than existing evaluation approaches which are based on assumptions that are applicable only in certain situations. With WALMAGE, it is now possible to choose the most suitable WUM algorithm for a particular electronic commerce scenario. However, there are still several open issues to be handled before WALMAGE can be endorsed for practical usage:

- Human behavior is extremely complex and thus, difficult to describe and generalize. Hence, there is a need to integrate theories from psychology to come up with richer cognitive measures. One possible additional measure would be the likelihood of a user trying to compare prices of online products so as to get the best bargain: *cost consciousness*.
- Empirical studies must be conducted to determine the probability distribution of the base characteristics so that they would have more precise values rather than just three values for low, average and high.
- It is necessary to use real life logfile data from electronic commerce sites to run experiments and assess the practicality of WALMAGE.
- More measures should be developed for other web-based applications like digital libraries and search engines.

5.2.3 Clustering

The curse of dimensionality is one issue that CLUID has failed to address effectively. It may not be possible for CLUID to find clusters in high-dimensional datasets because of the sparsity of the data. Data in the important text and biology domains usually have high dimensions; in text datasets, the number of key words or phrases is the dimensionality; in biological datasets, the number of samples or genes is the dimensionality. Therefore, it is imperative that clustering techniques work well with high-dimensional data.

By adapting the ideas of subspace clustering, CLUID may be able to lift the dimensionality curse. Subspace clustering focuses only on specific dimensions where interesting clusters can be discovered. In addition, CLUID also opens up new possible research avenues particularly in the generalization of clustering by its potential in easily accommodating the ideas of grid-based, density-based, resolution-based, pattern-based and projected clustering. By modifying the type of itemsets CLUID discovers or changing the criteria by which itemsets are deemed interesting, it is possible to generalize CLUID.

5.3 Concluding Thoughts

Our three-year quest for a good data structure for data mining rightly and worthily ends here with three novel and practical data structures as well as a research roadmap for future aspirants. In the course of this quest, human imagination and creativity have proven to be boundless—each time we think that a technique is beyond improvement, someone somewhere would come up with something better. Therefore, we are certain that our data structures would one day be surpassed in ingenuity, perhaps even at this very moment. However, one thing is certain: it has been a fulfilling quest because we have contributed in one way or another to *the* unending QUEST of our human race for the mastery of the physical world.

“By three methods we may learn wisdom: First, by reflection, which is noblest; Second, by imitation, which is easiest; and third by experience, which is the bitterest.”

CONFUCIUS (551-479 B.C.) Philosopher

References

- [1] C. C. Aggarwal, C. Procopiuc, J. L. Wolf, P. S. Yu, and J. S. Park. Fast algorithms for projected clustering. In *Proc. ACM SIGMOD Conf.*, pages 61–72, Philadelphia, Pennsylvania, USA, June 1999.
- [2] C. C. Aggarwal and P. S. Yu. Online generation of association rules. In *Proc. 14th Int. Conf. on Data Engineering*, pages 402–411, Orlando, Florida, USA, February 1998.
- [3] C. C. Aggarwal and P. S. Yu. A new approach to online generation of association rules. *IEEE Trans. on Knowledge and Data Engineering*, 13(4):527–540, July 2001.
- [4] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proc. ACM SIGMOD Conf.*, pages 94–105, Seattle, WA, June 1998.
- [5] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. ACM SIGMOD Conf.*, pages 207–216, Washington D.C., May 1993.
- [6] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. on Very Large Databases*, pages 487–499, Santiago, Chile, September 1994.
- [7] H. Ahonen, O. Heinonen, M. Klemettinen, and A. I. Verkamo. Applying data mining techniques for descriptive phrase extraction in digital document collec-

REFERENCES

- tions. In *Proc. Advances in Digital Libraries Conference*, pages 2–11, Santa Barbara, California, USA, April 1998.
- [8] A. Amir, R. Feldman, and R. Kashi. A new and versatile method for association generation. *Information Systems*, 22(6/7):333–347, September 1999.
- [9] F. Angiulli, G. Ianni, and L. Palopoli. On the complexity of mining association rules. In *Proc. 9th Italian Symp. on Advanced Database Systems*, pages 177–184, Venice, Italy, June 2001.
- [10] M. F. Arlitt and C. L. Williamson. Web server workload characterization: the search for invariants. In *Proc. ACM SIGMETRICS Conf. on Measurement and modeling of computer systems*, pages 126–137, Philadelphia, Pennsylvania, USA, May 1996.
- [11] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proc. ACM Symp. PODS*, pages 1–16, Madison, Wisconsin, USA, June 2002.
- [12] A. D. Baxevanis. *Bioinformatics: a practical guide to the analysis of genes and proteins*. New York: Wiley-Interscience, 2001.
- [13] S. D. Bay. Multivariate discretization for set mining. *Knowledge and Information Systems*, 3(4):491–512, November 2001.
- [14] R. J. Bayardo(Jr.). Efficiently mining long patterns from databases. In *Proc. ACM SIGMOD Conf.*, pages 85–93, Seattle, Washington, USA, June 1998.
- [15] C. Becquet, S. Blachon, B. Jeudy, J. F. Boulicaut, and O. Gandrillon. Strong-association-rule mining for large-scale gene-expression data analysis: A case study on human sage data. *Genome Biology*, 3(12), November 2002.
- [16] F. Beil, M. Ester, and X. Xu. Frequent term-based text clustering. In *Proc. 8th ACM SIGKDD*, pages 436–442, Edmonton, Alberta, Canada, July 2002.

REFERENCES

- [17] B. Berendt, B. Mobasher, M. Spiliopoulou, and J. Wiltshire. Measuring the accuracy of sessionizers for web usage analysis. In *Proc. Int. Workshop on Web Mining*, pages 7–14, Chicago, USA, April 2001.
- [18] A. Berson, S. Smith, and K. Thearling. *Building data mining applications for CRM*. McGraw-Hill, New York, 2000.
- [19] P. Bertone and M. Gerstein. Integrative data mining: The new direction in bioinformatics. *IEEE Engineering in Medicine and Biology*, 20(4):33–40, July 2001.
- [20] C.L. Blake and C.J. Merz. *UCI Repository of machine learning databases*. <http://www.ics.uci.edu/~mlearn/MLRepository.html>, 1998.
- [21] J. Borges and M. Levene. Mining association rules in hypertext databases. In *Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining*, pages 149–153, New York, USA, August 1998.
- [22] J. Borges and M. Levene. Data mining of user navigation patterns. In *Proc. Int. Workshop on Web Usage Analysis and User Profiling*, pages 31–36, San Diego, California, USA, August 1999.
- [23] C. Bounsaythip and E. Rinta-Runsala. Overview of data mining for customer behavior modeling. *VTT Information Technology Technical Report TTE1-2001-18*, 2001.
- [24] P. S. Bradley, K. P. Bennett, and A. Demiriz. Constrained k-means clustering. *Microsoft Research Technical Report MSR-TR-2000-65*, 2000.
- [25] P. S. Bradley and U. M. Fayyad. Refining initial points for k-means clustering. In *Proc. 15th Int. Conf. on Machine Learning*, pages 91–99, Madison, Wisconsin, USA, July 1998.
- [26] T. Brijs, B. Goethals, G. Swinnen, K. Vanhoof, and G. Wets. A data mining framework for optimal product selection in retail supermarket data: the gener-

REFERENCES

- alized profset model. In *Proc. 6th ACM SIGKDD*, pages 300–304, Boston, MA USA, August 2000.
- [27] T. Brijs, G. Swinnen, K. Vanhoof, and G. Wets. Using association rules for product assortment decisions: A case study. In *Proc. 5th ACM SIGKDD*, pages 254–260, San Diego, California, USA, August 1999.
- [28] S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: generalizing association rules to correlations. In *Proc. ACM SIGMOD Conf.*, pages 265–276, Tucson, Arizona, USA, May 1997.
- [29] A. G. Buchner and M. D. Mulvenna. Discovering internet marketing intelligence through online analytical web usage mining. *ACM SIGMOD Record*, 27(4):54–61, December 1998.
- [30] A. Buja, D. F. Swayne, M. Littman, N. Dean, and H. Hofmann. Xgvis: Interactive data visualization with multidimensional scaling. *Journal of Computational and Graphical Statistics (to appear)*.
- [31] D. Burdick, M. Calimlim, and J. Gehrke. Mafia: A maximal frequent itemset algorithm for transactional databases. In *Proc. 17th Int. Conf. on Data Engineering*, pages 443–452, Heidelberg, Germany, April 2001.
- [32] M. D. Byrne, B. E. John, N. S. Wehrle, and D. C. Crow. The tangled web we wove: a taskonomy of www use. In *Proc. Int. Conf. on Human Factors in Computing Systems*, pages 544–551, Pittsburgh, Pennsylvania, USA, May 1999.
- [33] S. Chakrabarti. Data mining for hypertext: A tutorial survey. *SIGKDD Explorations*, 1(2):1–11, January 2000.
- [34] S. Chakrabarti, B. E. Dom, and P. Indyk. Enhanced hypertext categorization using hyperlinks. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 307–318, Seattle, US, June 1998.

REFERENCES

- [35] J. H. Chang and W. S. Lee. Finding recent frequent itemsets adaptively over online data streams. In *Proc. 9th ACM SIGKDD*, pages 487–492, Washington, DC, USA, August 2003.
- [36] P. Cheeseman and J. Stutz. Bayesian classification (AutoClass): Theory and results. In *Advances in Knowledge Discovery and Data Mining*, pages 153–180. MIT Press, March 1996.
- [37] M. S. Chen, J. S. Park, and P. S. Yu. Data mining for path traversal patterns in a web environment. In *Proc. 16th Int. Conf. on Distributed Computing Systems*, pages 385–392, Hong Kong, May 1996.
- [38] M. S. Chen, J. S. Park, and P. S. Yu. Efficient data mining for path traversal patterns. *IEEE Trans. on Knowledge and Data Engineering*, 10(2):209–221, March 1998.
- [39] D. W. Cheung, J. Han, V. T. Ng, and C. Y. Wong. Maintenance of discovered association rules in large databases: An incremental updating technique. In *Proc. 12th Int. Conf. on Data Engineering*, pages 106–114, New Orleans, Louisiana, February 1996.
- [40] D. W. Cheung, S. D. Lee, and B. Kao. A general incremental technique for maintaining discovered association rules. In *Proc. 5th Int. Conf. on Database Systems for Advanced Applications*, pages 185–194, Melbourne, Australia, April 1997.
- [41] A. Cockburn and B. McKenzie. What do web users do? An empirical analysis of web use. *Int. Journal of Human-Computer Studies*, 54(6):903–922, June 2001.
- [42] E. G. Coffman and J. Eve. File structures using hashing functions. *Communications of the ACM*, 13(7):427–432, July 1970.
- [43] World Wide Web Consortium. *The Common Logfile Format*. <http://www.w3.org/Daemon/User/Config/Logging.html#common-logfile-format>, 1995.

REFERENCES

- [44] R. Cooley, B. Mobasher, and J. Srivastava. Grouping web page references into transactions for mining world wide web browsing patterns. In *Proc. IEEE Knowledge and Data Engineering Workshop*, pages 2–9, Newport Beach, CA, November 1997.
- [45] R. Cooley, B. Mobasher, and J. Srivastava. Web mining: Information and pattern discovery on the world wide web. In *Proc. 9th Int. Conf. on Tools with Artificial Intelligence*, pages 558–567, Newport Beach, CA, November 1997.
- [46] R. Cooley, B. Mobasher, and J. Srivastava. Data preparation for mining world wide web browsing patterns. *Knowledge and Information Systems*, 1(1):5–32, February 1999.
- [47] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.
- [48] Parametric Technology Corporation. *Windchill*.
<http://www.ptc.com/products/windchill/>.
- [49] Parametric Technology Corporation. *PTC's Product Development System: Product Lifecycle Management for Manufacturing Companies*.
<http://www.ptc.com/company/whitepapers/>, 2002.
- [50] P. Danzig, J. Mogul, V. Paxson, and M. Schwartz. *The Internet Traffic Archive*.
<http://ita.ee.lbl.gov/html/./index.html>, 2000.
- [51] A. Das, W. K. Ng, and Y. K. Woon. Rapid association rule mining. In *Proc. 10th Int. Conf. on Information and Knowledge Management*, pages 474–481, Atlanta, Georgia, November 2001.
- [52] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.
- [53] N. Dean, editor. *OCLC Researchers Measure the World Wide Web*. Number 248. Online Computer Library Center (OCLC) Newsletter, November 2000.

REFERENCES

- [54] G. Dong and J. Li. Efficient mining of emerging patterns: Discovering trends and differences. In *Proc. 5th ACM SIGKDD*, pages 43–52, San Diego, USA, August 1999.
- [55] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In *Proc. 12th Int. Conf. on Machine Learning*, pages 194–202, San Francisco, California, USA, July 1995.
- [56] R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. John Wiley and Sons, New York, 1973.
- [57] M. B. Eisen, P. T. Spellman, P. O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proc. Natl. Acad. Sci. USA*, 95(25):14863–14868, December 1998.
- [58] K. Elliott, R. Scionti, and M. Page. *The Confluence of Data Mining and Market Research for Smarter CRM*. http://www.spss.com/home_page/wp133.htm, 2003.
- [59] M. Ester, H. P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining*, pages 226–231, Portland, Oregon, August 1996.
- [60] F. J. Ferrer-Troyano, J. S. Aguilar-Ruiz, and J. C. R. Santos. Prototype-based mining of numeric data streams. In *Proc. ACM Symp. on Applied Computing*, pages 480–484, Melbourne, Florida, March 2003.
- [61] D. Florescu, A. Y. Levy, and A. O. Mendelzon. Database techniques for the world-wide web: A survey. *SIGMOD Record*, 27(3):59–74, 1998.
- [62] A. Foss and O. R. Zaiane. A parameterless method for efficiently discovering clusters of arbitrary shape in large datasets. In *Proc. Int. Conf. on Data Mining*, pages 179–186, Maebashi City, Japan, December 2002.

REFERENCES

- [63] E. Fredkin. Trie memory. *Communications of the ACM*, 3(9), September 1960.
- [64] J. Gama, R. Rocha, and P. Medas. Accurate decision trees for mining high-speed data streams. In *Proc. 9th ACM SIGKDD*, pages 523–528, Washington, DC, USA, August 2003.
- [65] M. Gardner and J. Bieker. Data mining solves tough semiconductor manufacturing problems. In *Proc. 6th ACM SIGKDD*, pages 376–383, Boston, MA, USA, August 2000.
- [66] W. Gaul and L. Schmidt-Thieme. Mining web navigation path fragments. In *Proc. Workshop on Web Mining for E-Commerce - Challenges and Opportunities*, Boston, MA, USA, August 2000.
- [67] M. D. Gordon and S. Dumais. Using latent semantic indexing for literature based discovery. *Journal of the American Society for Information Science*, 49(8):674–685, June 1998.
- [68] S. Guha, R. Rastogi, and K. Shim. Cure: An efficient clustering algorithm for large databases. In *Proc. ACM SIGMOD Conf.*, pages 73–84, Seattle, Washington, USA, June 1998.
- [69] M. Halkidi and M. Vazirgiannis. Clustering validity assessment: finding the optimal partitioning of a data set. In *Proc. 1st IEEE Conf. on Data Mining*, pages 187–194, San Jose, California, USA, November 2001.
- [70] P. M. Hallam-Baker and B. Behlendorf. *Extended Log File Format*. <http://www.w3.org/TR/WD-logfile.html>, 1996.
- [71] E. H. Han, G. Karypis, V. Kumar, and B. Mobasher. Clustering based on association rule hypergraphs. In *Proc. Workshop on Research Issues on Data Mining and Knowledge Discovery*, pages 9–13, Tucson, Arizona, USA, May 1997.

REFERENCES

- [72] J. Han, Y. Cai, and N. Cercone. Knowledge discovery in databases: An attribute-oriented approach. In *Proc. 18th Int. Conf. on Very Large Databases*, pages 547–559, Vancouver, Canada, August 1992.
- [73] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *Proc. 21st Int. Conf. on Very Large Databases*, pages 420–431, Zurich, Switzerland, September 1995.
- [74] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, San Francisco, 2001.
- [75] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proc. ACM SIGMOD Conf.*, pages 1–12, Dallas, Texas, USA, May 2000.
- [76] D. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining*. The MIT Press, 2001.
- [77] J. Heer and E. H. Chi. Separating the swarm: Categorization methods for user access sessions on the web. In *Proc. ACM CHI Conf. on Human Factors in Computing Systems*, pages 243–250, Minneapolis, Minnesota, April 2002.
- [78] M. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. In *Digital Equipment Corporation, Technical Report TR 1998-011*, 1998.
- [79] S. Hettich and S. D. Bay. *The UCI KDD Archive*. <http://kdd.ics.uci.edu>, 1999.
- [80] C. Hidber. Online association rule mining. In *Proc. ACM SIGMOD Conf.*, pages 145–154, Philadelphia, Pennsylvania, USA, June 1999.
- [81] A. Hinneburg and D. A. Keim. An efficient approach to clustering in large multimedia databases with noise. In *Proc. 4th ACM SIGKDD*, pages 58–65, New York, August 1998.

REFERENCES

- [82] A. Hinneburg and D. A. Keim. Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. In *Proc. 25th Int. Conf. on Very Large Databases*, pages 506–517, Edinburgh, Scotland, September 1999.
- [83] J. D. Holt and S. M. Chung. Efficient mining of association rules in text databases. In *Proc. 8th Int. Conf. on Information and Knowledge Management*, pages 234–242, Kansas City, Missouri, USA, November 1999.
- [84] M. Houtsma and A. Swami. Set-oriented mining of association rules. *Research Report RJ 9567, IBM Almaden Research Center*, October 1993.
- [85] B. A. Huberman, P. L. T. Pirolli, J. E. Pitkow, and R. M. Lukose. Strong regularities in World Wide Web surfing. *Science*, 280(5360):95–97, April 1998.
- [86] A. Joshi and R. Krishnapuram. On mining web access logs. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 63–69, Dallas, Texas, USA, May 2000.
- [87] G. Karypis, E. H. Han, and V. Kumar. Chameleon: Hierarchical clustering using dynamic modeling. *Computer*, 32(8):68–75, August 1999.
- [88] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley and Sons, New York, 1990.
- [89] E. M. Knorr and R. T. Ng. Algorithms for mining distance-based outliers in large datasets. In *Proc. 24th Int. Conf. on Very Large Data Bases*, pages 392–403, New York, USA, August 1998.
- [90] D. E. Knuth. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*. Addison-Wesley Publishing Company, 1968.
- [91] D. E. Knuth. *The Art of Computer Programming, Vol. 3: Sorting and Searching*. Addison-Wesley Publishing Company, 1973.
- [92] R. Kohavi. Mining e-commerce data: the good, the bad, and the ugly. In *Proc. 7th ACM SIGKDD*, pages 8–13, San Francisco, California, USA, August 2001.

REFERENCES

- [93] R. Kohavi and F. J. Provost. Applications of data mining to electronic commerce. *Data Mining and Knowledge Discovery*, 5(1/2):5–10, January 2001.
- [94] T. Kohonen. *Self-Organization and Associative Memory*. Springer-Verlag, 1984.
- [95] W. Kusters, E. Marchiori, and A. Oerlemans. Mining clusters with association rules. In *Proc. 3rd Int. Symp. on Intelligent Data Analysis*, pages 39–50, Amsterdam, The Netherlands, August 1999.
- [96] J. B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29:1–27, 1964.
- [97] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling the web for emerging cyber-communities. In *Proc. 8th Int. World Wide Web Conf.*, pages 403–415, Toronto, Canada, May 1999.
- [98] C. M. Kuok, A. W. C. Fu, and M. H. Wong. Mining fuzzy association rules in databases. *ACM SIGMOD Record*, 27(1):41–46, March 1998.
- [99] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Proc. 1st IEEE Conf. on Data Mining*, pages 313–320, San Jose, California, USA, November 2001.
- [100] R. D. Lawrence, G. S. Almasi, V. Kotlyar, M. S. Viveros, and S. Duri. Personalization of supermarket product recommendations. *Data Mining and Knowledge Discovery*, 5(1/2):11–32, January 2001.
- [101] W. Lin, S. A. Alvarez, and C. Ruiz. Collaborative recommendation via adaptive association rule mining. In *Proc. Workshop on Web Mining for E-Commerce - Challenges and Opportunities*, Boston, MA, USA, August 2000.
- [102] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *Proc. 4th ACM SIGKDD*, pages 80–86, New York, USA, August 1998.

REFERENCES

- [103] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. 5th Berkeley Symp. on Mathematical Statistics and Probability*, pages 281–297, 1967.
- [104] S. K. Madria, S. S. Bhowmick, W. K. Ng, and E. P. Lim. Research issues in web data mining. In *Proc. 1st Int. Conf. on Data Warehousing and Knowledge Discovery*, pages 303–312, Florence, Italy, August 1999.
- [105] F. Massealia, M. Teisseire, and P. Poncelet. Real time web usage mining: a heuristic based distributed miner. In *Proc. 12th Int. Workshop on Research Issues on Data Engineering*, pages 288–297, San Jose, USA, February 2002.
- [106] F. Masseglia, P. Poncelet, and M. Teisseire. Using data mining techniques on web access logs to dynamically improve hypertext structure. *ACM SigWeb Letters*, 8(3):13–19, October 1999.
- [107] MatrixOne. *eMatrix*. <http://www.matrixone.com>.
- [108] P. B. Mirchandani and R. L. Francis, editors. *Discrete Location Theory*. John Wiley & Sons, New York, 1990.
- [109] D. Mladenic. Turning yahoo to automatic web-page classifier. In *Proc. 13th European Conference on Artificial Intelligence*, pages 473–474, Brighton, UK, August 1998.
- [110] B. Mobasher, R. Cooley, and J. Srivastava. Automatic personalization based on Web usage mining. *Communications of the ACM*, 43(8):142–151, August 2000.
- [111] B. Mobasher, H. Dai, T. Luo, M. Nakagawa, Y. Sun, and J. Wiltshire. Discovery of aggregate usage profiles for web personalization. In *Proc. Workshop on Web Mining for E-Commerce - Challenges and Opportunities*, Boston, MA, USA, August 2000.

REFERENCES

- [112] B. Mobasher, H. Dai, T. Luo, Y. Sun, and J. Zhu. Integrating web usage and content mining for more effective personalization. In *Proc. 1st Int. Conf. on Electronic Commerce and Web Technologies*, pages 165–176, September 2000.
- [113] D. R. Morrison. PATRICIA - practical algorithm to retrieve information coded in alphanumeric. *Journal of the ACM*, 15(4):514–534, October 1968.
- [114] H. Nagesh, S. Goil, and A. Choudhary. Adaptive grids for clustering massive data sets. In *Proc. 1st SIAM Int. Conf. on Data Mining*, Chicago, April 2001.
- [115] A. Nanopoulos, D. Katsaros, and Y. Manolopoulos. Effective prediction of web-user accesses: a data mining approach. In *Proc. WEBKDD 2001*, San Francisco, CA, USA, August 2001.
- [116] A. Nanopoulos, D. Katsaros, and Y. Manolopoulos. A data mining algorithm for generalized web prefetching. *IEEE Trans. on Knowledge and Data Engineering*, 15(5), September 2003.
- [117] A. Nanopoulos and Y. Manolopoulos. Finding generalized path patterns for web log data mining. In *Proc. 4th East-European Conf. on Advances in Databases and Information Systems*, pages 215–228, Prague, Czech Republic, September 2000.
- [118] R. T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *Proc. 20th Int. Conf. on Very Large Data Bases*, pages 144–155, Santiago, Chile, September 1994.
- [119] L. O’Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani. Streaming-data algorithms for high-quality clustering. In *Proc. 18th Int. Conf. on Data Engineering*, pages 685–696, San Jose, California, USA, February 2002.
- [120] The University of Massachusetts Boston. *ARMiner Project*. <http://www.cs.umb.edu/~laur/ARMiner/>, 2000.

REFERENCES

- [121] National Association of Recording Merchandisers. *2000 Annual Survey Results*. <http://www.narm.com/research/annualsurvey.htm>, 2000.
- [122] J. S. Park, M. S. Chen, and P. S. Yu. Using a hash-based method with transaction trimming for mining association rules. *IEEE Trans. on Knowledge and Data Engineering*, 9(5):813–824, September 1997.
- [123] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *Proc. 7th Int. Conf. on Database Theory*, pages 398–416, Jerusalem, Israel, January 1999.
- [124] J. Pei, J. Han, and R. Mao. CLOSET: An efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 21–30, Dallas, Texas, USA, May 2000.
- [125] J. Pei, J. Han, B. Mortazavi-Asl, and H. Zhu. Mining access patterns efficiently from web logs. In *Proc. Pacific-Asia Conf. on Knowledge Discovery and Data Mining*, pages 396–407, Kyoto, Japan, April 2000.
- [126] J. Pei, X. Zhang, M. Cho, H. Wang, and P. S. Yu. Maple: A fast algorithm for maximal pattern-based clustering. In *Proc. 3rd Int. Conf. on Data Mining*, pages 689–692, Melbourne, Florida, USA, December 2003.
- [127] D. Pelleg and A. Moore. Accelerating exact k-means algorithms with geometric reasoning. In *Proc. 5th ACM SIGKDD*, pages 277–281, San Diego, California, USA, August 1999.
- [128] D. Pelleg and A. Moore. X-means: Extending K-means with efficient estimation of the number of clusters. In *Proc. 17th Int. Conf. on Machine Learning*, pages 727–734, San Francisco, California, USA, June 2000.
- [129] J. Pitkow. In search of reliable usage data on the www. In *Proc. 6th Int. World Wide Web Conf.*, pages 451–463, Santa Clara, California, USA, April 1997.

REFERENCES

- [130] J. W. Priest and J. M. Sanchez. *Product Development and Design for Manufacturing*. Marcel Dekker, 2001.
- [131] C. M. Procopiuc, M. Jones, P. K. Agarwal, and T. M. Murali. A monte carlo algorithm for fast projective clustering. In *Proc. ACM SIGMOD Conf.*, pages 418–427, Madison, Wisconsin, USA, June 2002.
- [132] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. In *Proc. ACM SIGMOD Conf.*, pages 427–438, Dallas, Texas, USA, May 2000.
- [133] Matthew R. Sanders. *Global eCommerce Approaches Hypergrowth*. <http://www.forrester.nl/ER/Research/Brief/Excerpt/0,1317,9229,FF.html>, 2000.
- [134] N. L. Sarda and N. V. Srinivas. An adaptive algorithm for incremental mining of association rules. In *Proc. 9th Int. Workshop on Database and Expert Systems Applications*, pages 240–245, Vienna, Austria, August 1998.
- [135] K. Satou, G. Shibayama, T. Ono, and Y. Yamamura. Finding association rules on heterogeneous genome data. In *Proc. Pacific Symp. on Biocomputing*, pages 397–408, Hawaii, USA, January 1997.
- [136] M. Seno and G. Karypis. Lpminer: An algorithm for finding frequent itemsets using length decreasing support constraint. In *Proc. 1st IEEE Conf. on Data Mining*, pages 505–512, San Jose, California, USA, November 2001.
- [137] J. Sharrard, S. J. Kafka, and M. J. Tavilla. *Global Online Trade Will Climb To 18% Of Sales*. <http://www.forrester.com/ER/Research/Brief/Excerpt/0,1317,13720,00.html>, December 2001.
- [138] G. Sheikholeslami, S. Chatterjee, and A. Zhang. Wavecluster: A wavelet based clustering approach for spatial data in very large databases. *VLDB Journal*, 8(3-4):289–304, February 2000.

REFERENCES

- [139] M. Spiliopoulou. Web usage mining for web site evaluation. *Communications of the ACM*, 43(8):127–134, 2000.
- [140] M. Spiliopoulou and L. C. Faulstich. WUM: A Web Utilization Miner. In *Proc. Int. Workshop on the Web and Data Bases*, pages 109–115, Valencia, Spain, March 1998.
- [141] M. Spiliopoulou and C. Pohle. Data mining for measuring and improving the success of web sites. *Data Mining and Knowledge Discovery*, 5(1/2):85–114, January 2001.
- [142] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *Proc. ACM SIGMOD Conf.*, pages 1–12, Montreal, Quebec, Canada, June 1996.
- [143] J. Srivastava, R. Cooley, M. Deshpande, and P. N. Tan. Web usage mining: Discovery and applications of usage patterns from web data. *SIGKDD Explorations*, 1(2):12–23, January 2000.
- [144] A. Sun, E. P. Lim, and W. K. Ng. Personalized classification for keyword-based category profiles. In *Proc. 6th European Conf. on Research and Advanced Technology for Digital Libraries*, pages 61–74, Rome, Italy, September 2002.
- [145] A. Sun, M. M. Naing, E. P. Lim, and W. Lam. Using support vector machine for terrorism information extraction. In *Proc. 1st NSF/NIJ Symp. on Intelligence and Security Informatics*, pages 1–12, Tucson, Arizona, USA, June 2003.
- [146] C. Tang, L. Zhang, M. Ramanathan, and A. Zhang. Interrelated two-way clustering: An unsupervised approach for gene expression data analysis. In *2nd IEEE Int. Symp. on Bioinformatics and Bioengineering*, pages 41–48, Bethesda, Maryland, March 2001.
- [147] Y. Theodoridis. *The R-tree-portal*. <http://www.rtreeportal.org>, 2003.

REFERENCES

- [148] H. Wang, W. Wang, J. Yang, and P. S. Yu. Clustering by pattern similarity in large data sets. In *Proc. ACM SIGMOD Conf.*, pages 394–405, Madison, Wisconsin, USA, June 2002.
- [149] K. Wang, S. Zhou, and J. Han. Profit mining: From patterns to actions. In *Proc. 8th Int. Conf. on Extending Database Technology*, pages 70–87, Prague, Czech Republic, March 2002.
- [150] Y. K. Woon, X. Li, W. K. Ng, and W. F. Lu. Parameterless data compression and noise filtering using association rule mining. In *Proc. 5th Int. Conf. on Data Warehousing and Knowledge Discovery*, pages 278–287, Prague, Czech Republic, September 2003.
- [151] Y. K. Woon, X. Li, W. K. Ng, and W. F. Lu. Cluid: Clustering using itemset discovery. *IEEE Trans. on Knowledge and Data Engineering (submitted)*, 2004.
- [152] Y. K. Woon, W. K. Ng, and A. Das. Fast online dynamic association rule mining. In *Proc. 2nd Int. Conf. on Web Information Systems Engineering*, pages 278–287, Kyoto, Japan, December 2001.
- [153] Y. K. Woon, W. K. Ng, X. Li, and W. F. Lu. Efficient web log mining for product development. In *Proc. 1st Int. Conf. on Cyberworlds*, pages 294–301, Singapore, December 2003.
- [154] Y. K. Woon, W. K. Ng, and E. P. Lim. Association rule mining for electronic commerce: New challenges. In *Proc. 3rd Int. Conf. on Information, Communications and Signal Processing*, Singapore, October 2001.
- [155] Y. K. Woon, W. K. Ng, and E. P. Lim. Evaluating web access log mining algorithms: A cognitive approach. In *Proc. 1st Int. Workshop on Mining for Enhanced Web Search*, pages 217–222, Singapore, December 2002.
- [156] Y. K. Woon, W. K. Ng, and E. P. Lim. Online and incremental mining of separately grouped web access logs. In *Proc. 3rd Int. Conf. on Web Information Systems Engineering*, pages 53–62, Singapore, December 2002.

REFERENCES

- [157] Y. K. Woon, W. K. Ng, and E. P. Lim. A support-ordered trie for fast frequent itemset discovery. *IEEE Trans. on Knowledge and Data Engineering*, 16(7):875–879, July 2004.
- [158] Y. K. Woon, W. K. Ng, and E. P. Lim. Accelerating association rule mining with trie data structures. *Encyclopedia of Data Warehousing and Mining (to appear)*, 2005.
- [159] Y. K. Woon, W. K. Ng, and E. P. Lim. Web usage mining: Algorithms and results. *Web Mining: Applications and Techniques*, pages 373–391, 2005.
- [160] X. Xiaowei, M. Ester, H. P. Kriegel, and J. Sander. A distribution-based clustering algorithm for mining in large spatial databases. In *Proc. 14th Int. Conf. on Data Engineering*, pages 324–331, Orlando, Florida, February 1998.
- [161] D. Y. Yang, A. Johar, A. Grama, and W. Szpankowski. Summary structures for frequency queries on large transaction sets. In *Proc. Data Compression Conf.*, pages 420–429, Snowbird, Utah, USA, March 2000.
- [162] Q. Yang, H. H. Zhang, and T. Li. Mining web logs for prediction models in www caching and prefetching. In *Proc. 7th ACM SIGKDD*, pages 473–478, San Francisco, California, USA, August 2001.
- [163] C. H. Yeang, S. Ramaswamy, P. Tamayo, S. Mukherjee, R. M. Rifkin, M. Angelo, M. Reich, E. S. Lander, J. P. Mesirov, and T. R. Golub. Molecular classification of multiple tumor types. In *Proc. 9th Int. Conf. on Intelligent Systems for Molecular Biology*, pages 316–322, Copenhagen, Denmark, July 2001.
- [164] M. L. Yiu and N. Mamoulis. Frequent-pattern based iterative projected clustering. In *Proc. 3rd Int. Conf. on Data Mining*, Melbourne, Florida, USA, December 2003.
- [165] Ernst & Young. *Global Online Retailing Survey 2001*. http://www.ey.com/global/gcr.nsf/UK/Consumer_Products_-_Report_-_GlobalOnlineRetailingSurvey2001, 2001.

REFERENCES

- [166] M. J. Zaki. Scalable algorithms for association mining. *IEEE Trans. on Knowledge and Data Engineering*, 12(3):372–390, May 2000.
- [167] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In *Proc. 3rd Int. Conf. on Knowledge Discovery and Data Mining*, pages 283–286, Newport Beach, California, USA, August 1997.
- [168] B. Zhang. Generalized k-harmonic means - dynamic weighting of data in unsupervised learning. In *Proc. 1st SIAM Int. Conf. on Data Mining*, Chicago, USA, April 2001.
- [169] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. In *Proc. ACM SIGMOD Conf.*, pages 103–114, Montreal, Canada, June 1996.
- [170] C. Zhu and T. Jiang. Multicontext fuzzy clustering for separation of brain tissues in magnetic resonance images. *NeuroImage*, 18(3):685–696, March 2003.