Tree Algorithms for Mining Association Rules

Thesis submitted in accordance with
the requirements of the University of Liverpool
for the degree of Doctor of Philosophy by

Graham Goulbourne

August 2001

Dedicated to my wife and children

Abstract

With the increasing reliability of digital communication, the falling cost of hardware and increased computational power, the gathering and storage of data has become easier than at any other time in history. Commercial and public agencies are able to hold extensive records about all aspects of their operations. Witness the proliferation of point of sale (POS) transaction recording within retailing, digital storage of census data and computerized hospital records. Whilst the gathering of such data has uses in terms of answering specific queries and allowing visulisation of certain trends the volumes of data can hide significant patterns that would be impossible to locate manually. These patterns, once found, could provide an insight into customer behviour, demographic shifts and patient diagnosis hitherto unseen and unexpected.

Remaining competitive in a modern business environment, or delivering services in a timely and cost effective manner for public services is a crucial part of modern economics. Analysis of the data held by an organisaton, by a system that "learns" can allow predictions to be made based on historical evidence. Users may guide the process but essentially the software is exploring the data unaided.

The research described within this thesis develops current ideas regarding the exploration of large data volumes. Particular areas of research are the reduction of the search space within the dataset and the generation of rules which are deduced from the patterns within the data. These issues are discussed within an experimental framework which extracts information from binary data.

Contents

List of Figures

List of Tables

Acknowledgements

I wish to thank my supervisors Paul Leng and Frans Coenen for their guidance, patience and faith throughout this project, without which it would never have been completed.

Thanks are also due to the Facilities Management Division at the Royal & SunAlliance for their financial support and data, and the staff in the technical department at the University of Liverpool, especially Ken Chan.

Finally, I wish to thank all those friends and family who offered invaluable encouragement and support especially, Duncan Neary, Valentina Tamma, and Sarah my wife.

Chapter 1

Introduction

The rapid development of computer hardware and software has led to a revolution in working practices over the past thirty years. One aspect of this revolution is related to the ability of present day computers to store and manipulate databases containing very large quantities of information. This thesis is concerned with knowledge extraction from within such databases. Knowledge discovery in databases (KDD) has been defined by Fawley *et al.* [1991] as, *"the nontrivial extraction of implicit, previously unknown and potentially useful information from data"*. The terms "knowledge discovery in databases" and "data mining"[1] have in recent years, become somewhat confused. At the first KDD conference in Montreal in 1995 it was proposed that the term "knowledge discovery in databases" should be employed to describe the whole process of extracting knowledge from data. The term "data mining" should be used exclusively for the discovery stage of the knowledge discovery process. This thesis will use these definitions.

In the business world, databases store information relating to, customer transactions, product lines, share dealing, market fluctuations etc. Within public services databases contain medical records, library records, planning applications etc. The volume of data currently stored is huge and many businesses adopt the strategy of not disposing of data, believing that it may provide useful information at a later date. Falling prices of storage media make this a feasible and inexpensive option.

[1]There are also many other terms appearing in documents carrying similar or slightly different meanings such as, *knowledge mining from databases, knowledge extraction, data archaeology, data dredging, data analysis etc.*

Over the past ten years terms such as "data warehousing" and "data mining" have been increasingly seen in the mainstream media business sections [Taylor 1998], [Smith 1999], illustrating that businesses no longer see data storage and backup as a chore but as a potentially useful information source that could lead to a commercial gain. The main problem is the volume stored and the fact that a human operator may be unable to discover "hidden" trends within that data. This thesis considers the application of knowledge discovery techniques to the Facilities Management (FM) function of an organization. The FM operation of a company can be defined as an integrated approach to operating, maintaining, improving and adapting the buildings and infrastructure of an organization in order to create an environment that strongly supports the primary objectives of that organization (Further Education Funding Council [HMSO 1997]). Facilities Management is characterized by a great diversity of function, most of which is only peripherally related to the core objectives of the organization, and elements of which are essentially unrelated to each other. For example FM operations may cover such divisions as fleet car management, catering, printing requirements and building maintenance and management. This diversity of function is reflected in a corresponding diversity of information and data organization. The application of knowledge discovery techniques to this data may yield interesting patterns which were previously unexpected.

Figure 1.1 succinctly illustrates the motivation for KDD. This thesis is concerned broadly with advancing the efficiency of the data mining process. The research focuses in particular on issues arising in data mining from very large and dense datasets which have presented computational difficulties for analysts. An active area of research has involved the identification of *association rules* [Agrawal *et al.* 1993] for this purpose. This thesis develops new techniques for deriving association rules from large datasets. The aim of this research is to explore the generation of one type of data mining rule, the discovery driven association rule. Whilst the result of association rule generation i.e. the rule, is a simple and readily understand-

able concept, actually finding the rules within a given search space is a non-trivial problem.

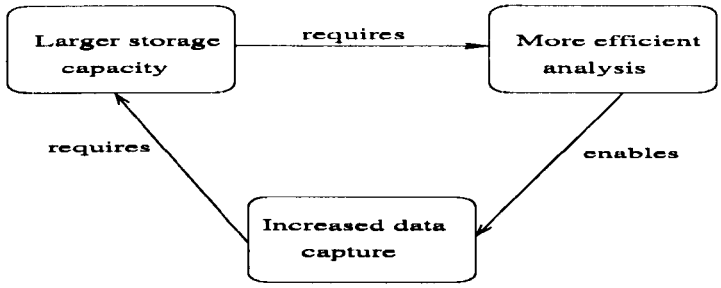


Figure 1.1: The general motivation for this research work

1.1 Database Evolution

Prior to 1970 the main types of database in use were the network and the hierarchical models. These databases lacked flexibility: the network used embedded pointers and pre-defined links which are in themselves not a problem but create one if a user wishes to define a relationship that the designer did not foresee. A more serious problem is that the network representation exposes implementional details that are not relevant from the application viewpoint. The network model evolved specifically to handle non-hierarchical relationships. The hierarchical model predates the network database, and suffers from a rigidity which makes the method awkward unless the application presents a natural hierarchy. Whilst both these systems have largely been supplanted many data processing systems with network or hierarchical cores are still functioning. Much of this legacy software uses IBM's Information Management System (IMS) product which was developed in the 1960's. It left a

large investment in expensively developed software to be amortized over the ensuing decades [Johnson 1997].

The relational model formulated by Codd [1970] allowed non-specialists to use databases, improved flexibility with respect to a wide variety of operations, especially queries, and also allowed formal definitions to make the approach sound and precise [Atzeni and DeAntonellis 1993]. The relational model has now become the standard database management system used throughout the business community because of this flexible and sound approach. Structured Query Language (SQL) provides a uniform interface for users, providing a collection of standard expressions for storing and retrieving data.

More recently two database models have been developed, known as post-relational models, which claim to provide more flexible data representation. Object orientated databases represent an application entity as a class which captures both the attributes and behaviour of the entity. This allows more accurate definitions of real world objects. Deductive databases, also known as the inferential model, store as little data as possible but compensate by maintaining rules (axioms) that allow new data combinations to be created as needed.

Despite the creation of the post-relational models the relational model with its widespread use remains the current favourite.

1.2 The Need for Knowledge Discovery

The requirement for knowledge discovery is directly linked to both the growing number and increase in capacity of databases. As observed above the volume of stored data increases daily, but the ability to effectively examine and use patterns that may be hidden within the data is still the subject of active research. There are a number of factors that have brought knowledge discovery and data mining

techniques to the attention of the business community;

1. a recognition that there is untapped value in large databases
2. a consolidation of database records leading to a single customer view
3. a consolidation of databases including the concept of data warehousing
4. a reduction in the cost of storing and processing allowing the accumulation of detailed data
5. intense competition for the customer's attention in an increasingly saturated market
6. a movement toward *de-massification*[2] [Ester *et al.* 1996] of business practices

Data mining tools must go beyond simply reporting data records or historical performance: they must ultimately provide the analyst with a deep insight into why certain actions have occurred. This insight is best achieved with a combination of hindsight and foresight. KDD techniques have not been limited to commercial activities: medical records [Mitchel 1999] , meteorological data [Buchner *et al.* 1999] and text data [Soderland 1997] have all provided sources for knowledge discovery. The wide variety of applications and the large area of active research suggest that as the world gathers more computerized data it is knowledge discovery techniques that will enable a more complete understanding of the information that has been captured.

[2]During the industrial revolution economies of scale led businesses to mass manufacturing, mass marketing and mass advertising. The information revolution is providing the capability to custom manufacture, market and advertise to small segments and ultimately to the individual. This is de-massification and it is a strong force in business today. Knowledge discovery techniques can be used to identify segment groupings that are not obvious.

1.3 The Knowledge Discovery Process

Manilla [1997] outlines the following steps for extracting knowledge from data;

1. understand the domain,
2. prepare the dataset,
3. discover patterns (data mining),
4. post-process discovered patterns,
5. put results into use.

The outcome of any knowledge discovery process is an enhanced view of the domain of interest The diversity of data types and the different goals of data mining make it unrealistic to expect one data mining system to handle all kinds of data. Specific data mining systems should be constructed for data mining on specific kinds of data, such as systems dedicated to mining in relational databases, transaction databases, multimedia databases etc. However, regardless of the data types or system, the user requires a solid understanding of the domain in order to select the correct subsets of data, suitable classes of patterns and good criteria for interestingness [Bayardo and Agrawal 1999], [Klemettinen *et al.* 1994], [Silberschatz and Tuzhilin 1995] of the pattern. The task is inherently interactive and iterative as to expect useful knowledge to be derived by merely pushing data into a black box is not realistic. Figure 1.2 illustrates the knowledge discovery process.

Since what will be discovered from a database is unpredictable, a high level data mining query should be treated as a probe which may disclose some interesting traces for further exploration. Interactive discovery allows the user to refine the request and investigate the data at a deeper level. Two types of data mining processes are available to the user wishing to exploit their data;

1. Verification: This process takes a hypothesis from a user and tests its validity against the data. The emphasis is with the user to formulate the hypothesis and issue the query; the data returned either affirms or negates the hypothesis. This process creates no new information, but as a result of information received the user can re-assess the hypothesis and pose a new, refined query.

2. Discovery: In this process the system automatically discovers important information hidden within the data. The data is searched with no hypothesis in mind and no interaction from the user and the system groups elements of the data according to some common characteristic.

These two mining processes can be linked to a data warehouse to form an online analytical mining architecture as described by Han *et al.* [1999]. A knowledge discovery system linked to a data warehouse is not a prerequisite since most data mining techniques can also work from data stored in flat files or operational databases. Mining a data warehouse however, usually results in higher quality information because of the diverse but complementary types of data stored.

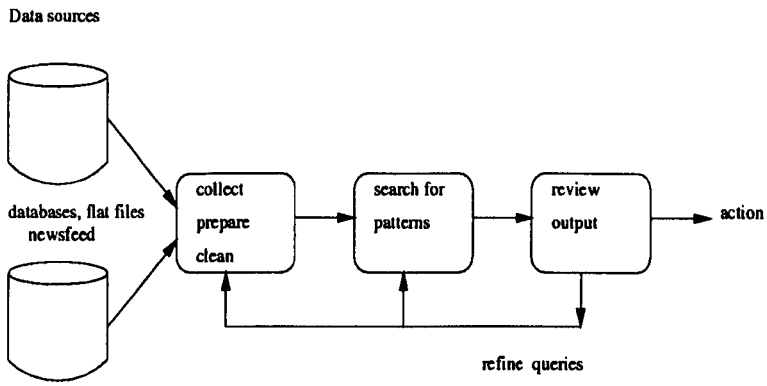


Figure 1.2: The knowledge discovery process from Inmon *et al.* [1997.]

1.4 Data Mining Classifications

Different classification schemes can be used to categorize data mining methods and systems, based on the kind of databases to be studied, the output of the process and the techniques to be used. The following classification was proposed by Chen *et al.* [1996a]:

1. What kind of database is to be mined;
   Classification can be made according to the kinds of databases to be mined. For example a system is a relational miner if it discovers knowledge from a relational database, an object-orientated miner if it discovers knowledge from an object-orientated database etc.

2. The output of the process;
   Different kinds of knowledge can be discovered by data miners, including association rules, classification rules, clustering etc. Data miners can also be categorized according to the abstraction level of the discovered knowledge. This may be classified into generalized, primitive level and multiple level knowledge.

3. What kind of technique is utilized;
   The underlying data mining technique may classify the system. It can be categorized by how it is driven: it may be autonomous, query driven or interactive. It can also be categorized according to its underlying approach into generalization based mining, pattern based mining, mining based on statistics or mathematical theories and integrated approaches.

Whilst the above list is not exhaustive it gives an idea of the many different fields involved within data mining. A more detailed discussion will be found in section 2.3.

1.5 Aims of the Research

The whole concept of data mining is based on the fact that there is physically too much data to be examined manually. This problem, like the volumes of stored data, is likely to grow. Tools and applications are required in the hands of data analysts and knowledge workers which will allow them the freedom to exploit as much of the potential of the data as possible. The increasing size of the potential input means that any solution must be scalable, i.e. increasing in linear space and time. The FM operation of a company provides a previously unexplored domain in which to apply data mining techniques. The research work described here seeks to establish the thesis that pre-processing of data using specialized data structures can offer significant advantages in terms of storage and execution time when mining for association rules. This thesis will be argued for, and established using the following "modus operandi":

- Current methods of association rule generation from binary data will be examined and their advantages and disadvantages discussed.
- Several "benchmark" programs will be created to provide a baseline for assessing the main research.
- A data structure called the P-tree will be developed for organizing and compacting the test datasets.
- A data structure called the T-tree will be developed to sum the data held within the P-tree.
- A case study will be undertaken using Facilities Management data to derive association rules and compare the P-tree/T-tree method with an implementation of the Agrawal and Srikant [1994] Apriori algorithm.

1.6 Structure of the Thesis

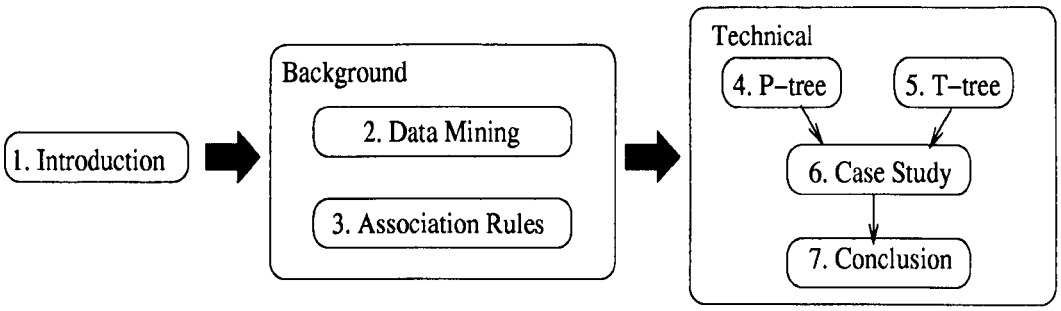


Figure 1.3: The structure of this thesis

This thesis is organized as illustrated by the diagram in Fig. 1.3. The first part (this chapter) provides background material: Chapter 2 describes the field of knowledge discovery and data mining, whilst Chapter 3 details past developments in association rule mining.

The second part of the thesis presents technical work. Chapter 4 presents the development of the P-tree and Chapter 5 the summation of the P-tree using the T-tree. Chapter 6 illustrates a case study using the P-tree and T-tree structures on FM datasets. Chapter 7 concludes the thesis.

Chapter 2

Previous Work on Knowledge Discovery in Databases

2.1 Introduction

Whilst this thesis is mainly concerned with a particular subset of Knowledge Discovery in Databases, the main result of any knowledge discovery process must be to enhance and aid decision making. It is therefore instructive to review the area of KDD to place this research in perspective. This chapter provides an overview of the broad field of KDD.

Knowledge discovery in databases is a relatively new field in computer science which has become popular over the past ten years. It has been brought about by the falling costs and increasing speeds of computer hardware and the proliferation of computer systems throughout modern society. These reasons have allowed organizations to reliably gather and store more data than could have been imagined fifteen years ago. KDD combines ideas drawn from such fields as databases, machine learning, statistics, visualization and parallel and distributed computing, *"its goal is to generate an integrated approach to knowledge discovery that is more powerful and richer than the sum of its parts"* [Stolorz and Musick 1997]. The gathering of data and attempting to extract beneficial information from that data has introduced new problems, primarily because of the significant volumes which are now able to be collected. For example;

- the 1990 US census collected over a million million bytes of data;
- the human genome project stores thousands of bytes for each of the several billion genetic bases;
- NASA earth observation satellites generate a terabyte (i.e. 10^9 bytes) of data each day [Bramer 1999].

2.2 Overview of KDD

The record of the Proceedings of the 1995 Conference in KDD opens with the following quotations:

- *"It is estimated that the amount of information doubles in the world every 20 months. What are we supposed to do with this flood of raw data? Clearly little of it will ever be seen by human eyes."*
- *"Computers promised fountains of wisdom but delivered floods of data"*

These large data repositories will, undoubtedly, hold within them useful information; sorting the interesting[1] from the non-interesting is the goal of KDD.

Decisions made within any organization cannot be made in a random manner, but need to have foundations. The more solid the foundation the more likely the decision made is the correct one. For a large corporation an incorrect decision can mean millions of pounds in lost revenue, for small to medium sized businesses an incorrect decision can be catastrophic. The deeper the understanding of past trends, the more likely that future predictions will produce the desired results. The overriding aim of KDD is to assist in the prediction of future profitable directions.

[1] "Interest" is of course a subjective concept, and a definition of what is interesting is required: it is usually taken as an overall measure of pattern value, combining validity, novelty, usefulness and simplicity [Fayyad *et al.* 1996].

2.2.1 Decision Support

Decision support tools comprehensively analyze/explore current and historical data, identify useful trends and create summaries to support high-level decision making for knowledge workers (executives, managers, analysts) [Ramakrishnan and Gehrke 1998]. There are three classes of data analysis tool as described by Chaudhuri and Dayal [1997]:

1. *complex queries:* tools which support traditional SQL-style queries, but are designed to support complex queries efficiently; relational DBMS optimized for decision support applications;

2. *On-line Analytical Processing (OLAP) [Codd et al. 1993]:* tools which support a class of stylized query which typically involves group-by and aggregation operators; and multiple dimension databases, data cubes etc. [Chen *et al.* 1996a]; these systems support a query style in which the data is best thought of as a multidimensional array and are influenced by end user tools such as spreadsheets, in addition to database query languages. OLAP systems work in a mostly read only environment;

3. *data mining (intelligent exploratory data analysis):* discovery of interesting patterns in the data.

2.2.2 Data Warehousing

Although a data warehouse is not essential for the process of data mining, in practice it is probably the most sensible option for a company wishing to instigate a KDD policy. The data warehouse need not necessarily be a huge database, the size depends on the volume of data stored and the techniques used to extract the information. The primary type of database found in most organizations is the operational database. In most cases these have been designed to support the applications used

in everyday transactions,and they are therefore optimized for this type of work, i.e. high response speeds and a large number of users [Inmon *et al.* 1997.].

The data warehouse is primarily constructed to store historical data and enhance decision support techniques. Most data warehouses contain large amounts of historical data, which is never altered or updated. Typically the warehouse is topped up with the most recent transactions from the on-line database (say) hourly, overnight or even weekly [Montgomery 1999]. The data warehouse can be used for both data mining activities and for OLAP, however, when mining data, the whole warehouse may be used, whereas when using OLAP tools it is common for the user to be accessing only a small subset of the warehouse contained within a *datamart.*

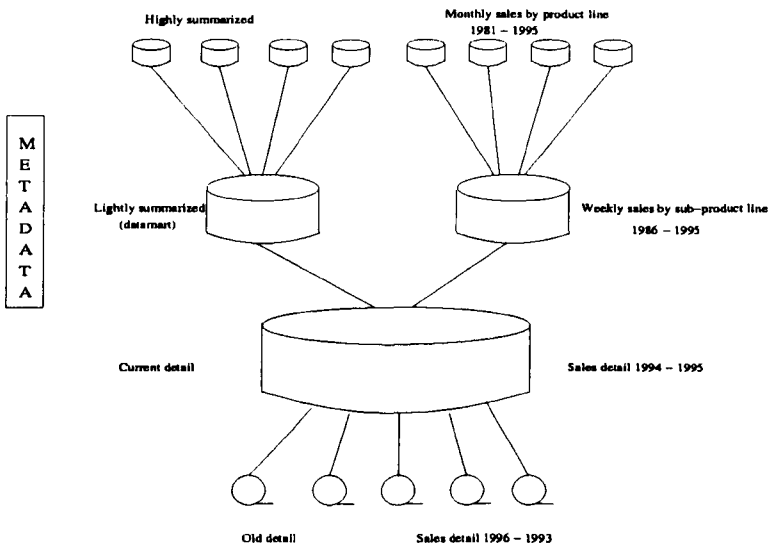


Figure 2.1: The Structure of the data warehouse from Inmon *et al.* [1997.]

Data Marts

There may be a great many users of the data warehouse system; different users will have different needs, from simple canned reports, to ad-hoc queries to advanced analysis. Meeting the needs of all the users with a single centralized system is not always feasible or wise. The solution is the data mart, sometimes called the depart-

mental data warehouse. There are several different types of data mart. The data mart can be implemented within the centeral repository by creating special application specific views on the data in the base tables. These views are flexible and occupy no space in the database, but are expensive at run-time. Another approach is the instantiated view. This is an optimization of the view where the data for a particular view is placed into another table and kept up-to-date with the original data. Instantiated views duplicate storage to improve performance. Keeping instantiated views synchronized with the base tables requires using database replication measures or triggers [Berry and Linoff 1997].

Metadata

Metadata simply means "data about data", the basic metadata is the database schema i.e. the physical layout of the data in the tables. Metadata also answers questions posed by users about the availability of data and gives tools for browsing through the contents of the data warehouse. Good metadata gives the users information about the data warehouse in a format they can understand to enable them to use the system in the most efficient manner.

2.2.3 Data Enrichment

In some cases data enrichment can be applied to the raw data prior to its inclusion within the data warehouse. An example of this would be an operational database containing information on subscriptions to a publisher's magazine titles. The unenriched data contains details of customer name, address, date of purchase and magazine title, and could be enriched by the inclusion of information regarding clients date of birth, income category, car/home ownership details [Adrianns and Zantinge 1996]. This is more realistic than it may initially seem, since it is possible to purchase demographic data on average incomes and details of car/home ownership can

also be traced fairly easily. Reliable data of this type can considerably enhance the knowledge discovery process.

2.2.4 Data Cleaning/Pre-processing

As described above most databases found in the business community are operational databases used for the day-to-day running of the business. To enable data mining to take place frequently the data will need to be "cleaned" [Simoudis *et al.* 1995]. This can be an expensive operation; the following are some of the considerations;

- Data sources may have been implemented in a variety of database systems, DB2, Oracle, Infomix etc. This is a result of maintaining and deploying legacy systems while also bringing in new technology in newer systems.
- In addition to the database organization method, relational versus non-relational, further complications that arise are multiple data encoding, transmission standards and connectivity standards. There is a need for a technology shift as the data is being moved.
- The selection of data from operational environments may be very complex. In order to qualify a record for extraction processing, several co-ordinated lookups to other records in a variety of files may have to be accomplished, requiring keyed reads, connecting logic etc.
- When there are multiple input files, key resolution must be done before the files can be merged together. This means that if different key structures are used in different input files then the merging program must have logic embedded in it that allows resolution.
- Data relationships that have been built into the old legacy program logic must be understood before these files can be used as input.

- Format Differences (data type and length); These occur because the databases that defined data differently did so from the narrow perspectives of the applications they supported, rather than from the need to apply organizational standards to data definitions.

2.2.5 KDD in a Multi-Database Environment

Methods used for KDD when multiple databases are involved, may be broadly viewed under three headings:

1. Application of KDD techniques to each database separately; an example of this would be inducing a set of rules and applying some interestingness measure to them. Such work is found in Silberschatz and Tuzhilin [1995], Kamber and Shinghal [1996] and Dzeroski and Grbovic [1995].

2. Selection of a number of related, although not necessarily homogeneous databases and integrating them for the purpose of data mining. Research in this area has been conducted by McClean and Scotney [1996], Levy [1996] and Ganesh *et al.* [1996].

3. Creation of a data warehouse (as discussed in section 2.2.2) and integrating all the databases, either for the purposes of mining the warehouse or to establish data marts for independent mining activities [Gill and Rao 1996].

Two techniques are reviewed for the integration of information from multiple, disparate, sources;

1. Entity integration techniques.

2. Query language extensions (e.g. SchemaSQL).

Entity integration techniques

Identification of records that represent the same real world entity is an important task in the database integration process. When a common identification mechanism for similar records across heterogeneous databases is not readily available, entity integration can be performed by examining the various attribute values among the records. The distances between attribute values can be used as a measure of similarity between the records they represent. Record matching conditions can, for entity integration, then be expressed as constraints on the attribute distances. KD techniques can be used to automatically derive these conditions, (expressed as decision trees) directly from the data, using a distance based framework [Ganesh *et al.* 1996].

The process of integration requires two distinct tasks: schema integration and entity identification. The former resolves mismatches such as homonyms (the same name for different attributes), and synonyms (different names for the same attributes). The removal of these inconsistencies allows the creation of a global schema, making it possible to treat all the records uniformly for further processing. The mapped global database may contain multiple instances of the same real world entity, therefore the identification of distinct entities is the purpose of the entity identification phase.

Query language extensions

SQL is the most common language for the extraction of information from databases. An extension of SQL, (SchemaSQL) has been developed, that offers the capability of uniform manipulation of data and meta-data in relational multi-database systems [Lakshmanan *et al.* 1996], [Lakshmanan *et al.* 1999]. Interoperability in multiple databases is the ability to share, interpret and manipulate information in component databases. Factors influencing interoperability can be classified into semantic issues (interpreting and cross-relating information in different databases), syntactic issues (heterogeneity in schemas, data models and query processing) and systems

issues (operating systems, communication protocols, security management etc.). SchemaSQL deals with the syntactic issues; its development is aimed at extracting information from relational databases storing semantically similar data in structurally different ways. Some of the key features required of a language for the interoperability of relational multi-database systems are:

1. It must have an expressive power that is independent of the schema with which the database is structured.
2. To promote interoperability, it must permit the restructuring of one database to conform to the schema of another.
3. It must be easy to use yet still be expressive.
4. It must provide full data manipulation and view definition capabilities.
5. It must be downward compatible with SQL in view of SQL's importance in the database world.
6. The language must realise a non-intrusive implementation that would require minimal additions to the RDBMS.

The architecture for implementing SchemaSQL builds on the existing database architecture in a non-intrusive manner. The SchemaSQL system is built on top of the existing SQL systems. A SchemaSQL server communicates with local databases in the federation. Local meta-information, database names, names of attributes and relations, are stored on the SchemaSQL server in the form of a table called the Federation System Table. Global queries are submitted to the SchemaSQL server which determines the queries to submit to the local databases. The server then collects the answers and executes a final series of SQL queries to produce an answer to the global query.

The rapid growth of the world wide web suggests the potential for mining large and distributed datasets, as such the integration of multiple distributed databases is an important research area within KDD. Furthermore, not all companies may wish to commit to a policy of data warehousing and the integration of their data on a temporary basis will allow analysis of the benefits of data warehousing.

2.3 Data Mining

There are three primary paradigms within data mining: *classification* methods, *association* methods and *clustering* methods. There are other data mining techniques such as *pattern-based similarity searches*, *mining path traversal patterns* and *visualization* which will be briefly discussed at the end of this section.

2.3.1 Classification

Data classification finds common properties among sets of objects in a database then places them into classes according to a classification model. To construct the classification model a sample dataset is treated as the training set the remainder is used as the test set. One designated attribute in the training set is called the dependent attribute and the others the predictor attributes. The goal is to build a model that takes the predictor attribute as inputs and produces a value for the dependent attribute. If the dependent attribute is numerical the problem is a regression problem otherwise it is called a classification problem.

Decision Trees

Decision trees classify examples to a finite number of classes, the nodes are labelled with the attribute names, the edges are labelled with possible values for this attribute and the leaves are labelled with the different classes. Objects are classified by fol-

lowing a path down the tree and taking the edges corresponding to the values of the attributes in that object [Dilly 1995].

Decision trees are attractive in a data mining environment for several reasons [Ganti *et al.* 1999];

- Their intuitive representation makes the model easy to understand.
- Constructing decision trees does not require input from the analyst.
- Their predictive accuracy is equal to, or higher than, other classification models.
- Fast, scalable algorithms can be used to construct decision trees from very large training databases.

There are two primary decision tree generation systems Classification And Regression Trees (CART), developed by Briemen *et al.* [1984] and the ID3 system [Quinlan 1986], C4.5 is the most recent available version of this Quinlan [1993]. The differences between CART and C4.5 are small, and so only C4.5 is discussed below.

C4.5 examines numerous recorded classifications and a model is constructed inductively by generalizing from specific examples. The tree is constructed using *leaf* nodes or *decision* nodes. A case is classified by starting at the root of the tree and moving through it until a leaf is encountered. At each decision node the case's outcome for the test at the node is determined and attention shifts to the root of the sub-tree corresponding to this outcome. When this process finally reaches a leaf node the class of the case is predicted to be that which is recorded in the leaf.

One of the problems of decision trees is that they can easily grow too large, making them incomprehensible to humans. The solution is to use the tree to generate *production rules.* These are of the form $L \rightarrow R$, in which the left-hand side, L is

a conjunction of the attribute-based tests and the right-hand side R is a class. To classify a case using the production rule model, the ordered list of rules is examined to find the first whose left-hand side is satisfied by the case. The predicted class is then the one nominated by the right-hand side of this rule.

Constructing classification trees is an extremely active area of data mining research, techniques such as *Rainforest* [Gehrke *et al.* 1998] strive to compact the data structure and *Public* [Rastogi and Shim 1998] attempts to reduce expansion costs of nodes.

Neural Networks

Neural networks are mathematical structures with the ability to *learn*. The method is a result of academic research into modeling nervous system learning. A "trained" network can be thought of as an "expert" in the category it has been given to analyse and can then be used to provide projections given new situations of interest and answer "what if" questions.

Within data mining one of the advantages of neural networks is their ability to generalize and to learn from experience, i.e. a training set; this ability mimics the human process of learning from experience.

The multi-layered perceptron combined with the back propagation technique is a common type of neural network for data mining [Brielley and Batty 1999]. The neural network has three types of layers, input, hidden and output. Each connection is assigned a *weight*, initially a set of random values. At the core of back propagation are the following steps:

1. The network gets a training example and using the existing weights in the network calculates the output.

2. Backpropagation then calculates the error by taking the difference between

the expected result and the output.

3. The error is fed back through the system and the weights are adjusted to minimize the error.

Much research has been conducted regarding using neural networks for data mining, but there are some problems which still need to be addressed.

1. Their predictions/decisions may be difficult to explain.
2. They may converge at a premature solution that is not optimal.

Genetic Algorithms

Based on the Darwinian theory of "survival of the fittest" a genetic classifier consists of a population of classification elements that compete to make a prediction. Genetic algorithms have been used to train neural networks and enhance pre-classified data in classification models for data mining tasks [Hekanaho 1997].

Genetic algorithms (GAs) work by evolving successive generations of genomes that get progressively more and more *fit*. The fitness function used within a GA is critical to its success or failure. Like the neural networks outlined above, the GA can suffer from converging at a solution that is not optimal; the effectiveness of the fitness function dictates this outcome. The goal of a GA is to maximize the fitness of the genomes within the population. This is achieved in the following manner:

1. Identify the genome and fitness function and create an initial generation of genomes.
2. Modify initial population by applying;
    - selection; keeping the size of the population constant but increasing the fitness of the next generation. Genomes with higher fitness survive, lower fitness genomes die off.

- crossover; genomes are broken at a selected position then are recombined to create two new genomes.
- mutation; random changes are made at random positions within the genome to allow features to appear that may not have been present in the initial population.

3. Repeat step 2 until the population no longer improves.

GAs offer a flexible and powerful way to search for optimal values, if the input data can be represented as a binary vector then a GA can process it. However, although many problems fall into this category, the details of the encoding may affect the quality of the result. The results produced by GAs are more understandable to users than those of say, neural networks, because the concept of "survival of the fittest" is well understood. It is also possible to watch the fitness of generations improve over time to get a "feel" for the evolutionary path being taken.

2.3.2 Association Rules

A detailed description of association rules [Agrawal *et al.* 1993] will be given in the following chapter, but the following represents a high-level overview. Given a collection of items and a set of records, each of which contains some number of items from the given collection, an association function is an operation against this set of records which returns affinities or patterns that exist among the collection of items. The patterns can be expressed by association rules such as "72% of all records that contain items A, B and C also contain items D and E". In this rule A, B and C are called the antecedent and D and E the consequent. The associations may involve any number of items on the LHS or RHS.

Constraint Based Association Rule Mining

The research within this thesis is directed towards discovery driven association rule mining. Research has been undertaken to provide verification driven association rule mining by Srikant and Agrawal [1995], Srikant *et al.* [1997], Kahng *et al.* [1997] and Han *et al.* [1999]. Many association rule mining methods only consider the leaf nodes of a taxonomy and are not designed to perform more generalized searching. For example the taxonomy in Figure 2.2 may show no associations between jackets and hiking boots but moving up the taxonomy there may be a significant association between outerwear and hiking boots. Furthermore, the taxonomy may be used to constrain the search space in the following manner; if the user poses the query,

$$(jacket \wedge shoes) \vee (descendants(clothes) \wedge \neg ancestors(hikingboots))$$

this expresses the constraint that rules are required that either (*a*) contain both jackets and shoes or, (*b*) contain clothes or any descendants of clothes and do not contain hiking boots or footwear.

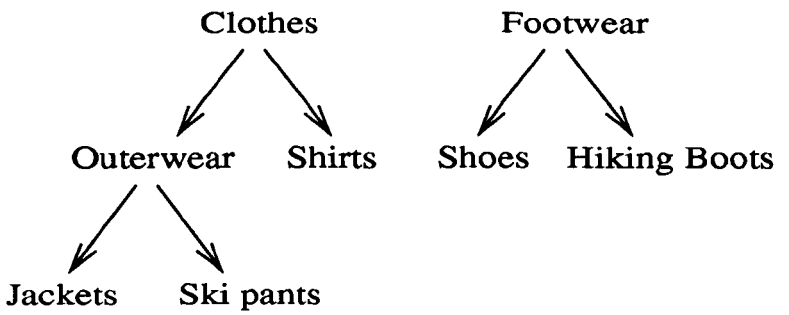


Figure 2.2: Example of a taxonomy

Integrating Association Rules

Several research efforts have been devoted to the integration of association rule mining combined with classification rule mining [Liu *et al.* 1998] and [Bayardo 1997].

These techniques extract association rules from a dataset and use them to produce classes for the classifier. One of the primary issues within this research area is related to the fact that many classification datasets often contain continuous attributes that need to be discretized for the association rule miner to function correctly. This subject itself is a major research area with work by Wang *et al.* [1998] and Yoda *et al.* [1997]

2.3.3 Clustering

The process of grouping physical or abstract objects into classes of similar objects is called clustering or unsupervised classification. Clustering analysis helps to construct meaningful partitioning of large data sets based on a "divide and conquer methodology". The clustering problem has been studied in many fields including, statistics, machine learning and biology. However, scalability was not a design goal in these applications as researchers assumed that the complete data set would fit in main memory, and so the focus was on improving the clustering quality. With the application of clustering to the very large data sets involved in data mining the issue of scalability has assumed increasing importance.

There are two primary methods of clustering used in data mining, the K-means method first published by MacQueen [1967] and the agglomeration hierarchical method of Willet [1988].

The K-Means Method

Partition-based clustering techniques attempt to break the data set into k clusters such that the partition optimizes a given criterion. [Bradley *et al.* 1998] uses cluster features to develop a framework for scaling up k-means methods. Starting with an initial data set partition the algorithm repeatedly moves points between clusters until the distribution optimizes a criterion function.

The framework functions by identifying sets of discardable, compressible and main memory points. A point is discardable if its membership of a cluster can be ascertained: the algorithm discards the actual points and retains only the cluster features of all discardable points. A point is compressible if it is not discardable but belongs to a tight sub-cluster. Such a sub-cluster is summarized by using its cluster feature. A point is a memory point if it is neither discardable nor compressible. The algorithm moves only main memory points and cluster features of compressible points between clusters until the distribution optimizes the criterion function. These algorithms assume that the clusters are hyper-episodal and of similar sizes. They cannot find clusters that vary in size or that have concave shapes Jain and Dubes [1988].

Agglomeration Hierarchical Methods

The above described problems of size and shape were addressed by Karypis *et al.* [1999] using the agglomeration method. In this method each data point initially forms its own cluster and all clusters gradually merge until all points are combined into one cluster. Towards the beginning of the approach the clusters are small and pure, the members of each cluster are very few but closely related. Towards the end of the process the clusters are large and less well defined. The history of the cluster generation is preserved to enable the level of clustering that suits the application to be ascertained.

In some of the models used by Karypis *et al.* [1999] the cluster is represented by a centroid or medoid, a data point that is closest to the centre of the cluster and the similarity between the two clusters is measured by the similarity between the centroids/medoids. Both these schemes fail for data in which points in a given cluster are closer to the centre of another cluster than to the centre of their own cluster.

The single-link hierarchical method measures the similarity between clusters by the

similarity of the closest pair of data points belonging to different clusters. Unlike centroid/medoid based methods this method can find clusters of arbitrary shape and different sizes. However, it is highly susceptible to noise, outliers and artifacts.

Clustering is a good primary data mining technique. The undirected nature of the search can be beneficial when there is no prior knowledge of the internal structure of the database. However, the choice of distance metric or similarity measure can radically alter the output of the procedure. The undirected nature of the search means that careful interpretation of the results is required; important clusters may be mis-interpreted if the analyst does not recognise them as such.

2.3.4 Pattern-Based Similarity

When searching for similar patterns in a temporal or spatial-temporal database two types of data mining queries are usually encountered Chen *et al.* [1996a]:

- Object-relative similarity query (i.e. range or similarity query) in which a search is performed on a collection of objects to find the ones that are within a user defined distance from the queried object.
- All-pair similarity query (i.e. spatial join) where the objective is to find all the pairs of elements within a user specified distance from each other.

Two types of similarity query for temporal data have emerged: whole matching, in which the target sequence and the sequences in the database have the same length; subsequence matching, in which the target sequence could be shorter than the sequences in the database and a match can occur at any arbitrary point.

Pattern matching has been applied to huge data sets such as the mission operations data for NASA's space shuttle [Keogh and Smyth 1997] with promising results.

2.3.5 Mining Path Traversal Patterns

Understanding user access patterns within the environment of the World Wide Web has become an important data mining task. Path traversal patterns are a normal part of graph theory but, as in other examples within this chapter the scale of the problem has led to new algorithms having to be developed. Insight into access patterns will not only assist in improving system design, (e.g. providing efficient access between highly correlated objects, better authoring design for pages, etc.) but, will also lead to improved marketing decision making (e.g. better user classification and user analysis). Because users must traverse a series of links to search for the desired information, some objects are visited because of their locations rather than their content. This feature of the traversal pattern problem unavoidably increases the difficulty of extracting meaningful information from sequences of traversal data and was explored by Chen *et al.* [1996b].

Comparisons have been drawn between the problem of finding reference sequences and finding large itemsets for association rules. However, they differ from each other in that a reference sequence in mining traversal patterns has to be consecutive, whereas a large itemset is just a combination of items in a transaction. This difference between these two problems calls for the use of different algorithms for mining the knowledge required.

2.4 Evaluation

The diversity of function within the facilities management domain offers the potential for extraction of knowledge and information which previously may not have been suspected. Issues arising from the FM domain applicable to data mining techniques include large dense datasets and distributed heterogeneous databases. The notion of a verification driven data mining technique to explore this data is unreal-

istic because of the great diversity of the data: therefore, a discovery driven method was required to address the problem. This research focuses on an association rule based approach.

2.5 Summary

KDD grew out of a need and desire to explore and exploit the potential commercial value contained within ever increasing data stores. This chapter has illustrated the volume and diversity of research being conducted within the field of knowledge discovery in databases and the increase in database sizes.

Section 2.2 gave a high level description of some of the techniques and issues involved in the capture and storage of data.

The quest for increased performance and scalability in knowledge discovery tools, and specifically within data mining applications, is not merely an academic pursuit, but has real world relevance.

Chapter 3

Association Rules

3.1 Introduction

An association rule, [Agrawal *et al.* 1993] is a probabilistic relationship of the form $A \rightarrow B$, where A and B are conjunctions of attributes between sets of database attributes. Only the positive occurance of attributes is of interest i.e. where a binary '1' is found in the set, an attribute such as "sex" is dealt with by converting it into two binary attributes: male and female. Attributes that are not naturally binary may have to be converted into a large number of binary attributes before they can be used. In the simplest case the attributes are boolean and the database takes the form of a set of records each of which reports the presence or absence of each of the attributes within that record. For example, in the database in Figure 3.1 the first complete record consists of the itemset ABC. Within the itemset ABC are also the itemsets A, B, C, AB, AC, and BC. An itemset may consist of one or more attributes.

Attributes

Records	A	B	C	D	E
	1	1	1	0	0
	1	1	1	0	1
	1	1	0	1	0
	0	1	0	0	0

Figure 3.1: Sample database I

The *support* for an itemset is the proportion of database records that contain $A \cup B$. For example, in the database in Figure 3.1 the support for BC is 0.5 because the itemset BC occurs twice. The support for B alone is 1. A support *threshold* is the proportion below which itemsets are not used to generate rules. The threshold is user defined and may be increased or decreased depending on the user's requirements. An itemset that is $\geq$ the support threshold is termed large[1]. The *confidence* in the rule R is the ratio;

$$\frac{support\ for\ R}{support\ for\ A}$$

For example, the confidence in the rule $B \rightarrow C$, from Figure 3.1, would be 50%.

Support and confidence provide an empirical basis for the derivation of the inference expressed in a rule. Support for a rule expresses the number of records within which the association may be observed, while the confidence expresses this as a proportion of the instances of the antecedent of the rule. It is usual to regard these rules as "interesting" only if the support and confidence exceed some threshold values. The initial problem may, therefore be formulated as the search for all association rules within a database for which the required support and confidence levels are attained. This formulation can be refined into two steps:

1. find all "large" itemsets. This step is computationally and I/O intensive and is the focus of most of the research into association rules,

2. generate high confidence rules. This step is relatively straightforward: rules are generated for all large itemsets provided the rules have at least minimum confidence [Parthasarathy *et al.* 1998].

The paradigmatic example is in supermarket *basket analysis*. Point of sale technology allows records of customer transactions to be accurately stored as a binary vector within a database, with a binary '1' indicating an item purchased. The volumes of data are extremely large; for example, Wal-Mart with a chain of over 2000

[1] Some authors use the terms, frequent, supported or covering.

retail stores, every day uploads over 20 million point-of-sale transactions, [Hedberg 1995]. Information about the current capabilities of commercial implementations of Apriori such as Clementine [SPSS 2002] is scarce. Claims have been made that it can handle 25,000 attributes, but these are not substantiated. Knowledge of which items may be purchased together can allow informed decisions to be made regarding pricing policy, product placement or store layout.

3.2 Association Rules and their Properties

Let $D = \{I_1, I_2, \ldots, I_n\}$ be a set of attributes, (or items) over the binary domain $\{0,1\}$. The input for association rule mining is each record in the database. Each record consists of all the attributes in the database, with some number of attributes set to 1 i.e. a set of binary vectors of length n (attribute number) and a number of records m.

If X represents a set of attributes from within the domain, (D), then for each record, if X is a subset of the record, or an exact match of the record the support for X is incremented by 1. An association rule is an expression $X \rightarrow Y$[2], where X and Y are both present within D as non-intersecting subsets. Two filters are applied to the data they are; γ (confidence threshold) and σ (support threshold). $X \rightarrow Y$ is satisfied with respect to the number of records if;

$$|n(X \cup Y)| \geq \sigma m$$

That is the number of records (n) that contain both X and $Y \geq$ the user defined support σ in m number of records.

$$\frac{|n(X \cup Y)|}{|n(X)|} \geq \gamma$$

[2]The use of the $\rightarrow$ symbol follows the conventions set in Agrawal *et al.* [1993] and is usually taken to mean, "when attribute A appears in the dataset then so does B", this is usually followed by a % value indicating the frequency with which this occurs.

That is at least a fraction of σ records have 1's in all attributes of X and Y and at least a fraction γ of the records having a 1 in all attributes of X also have a 1 in all of Y. Given a set of attributes X we say that X is *large* (with respect to the database and the given support threshold σ) if;

$$|n(X)| \geq \sigma m$$

.

The following example refers to Figure 3.1. The support threshold σ is 0.6 and the confidence threshold γ is 0.9. Once the database has been read it can be seen that the support for the single itemsets are $\{A\} = \frac{3}{4} = 0.75$, $\{B\} = \frac{4}{4} = 1$, $\{C\} = \frac{2}{4} = 0.5$, $\{D\}$ and $\{E\} = \frac{1}{4} = 0.25$. The sets $\{A\}$ and $\{B\}$ are both above the required support threshold therefore set $\{AB\}$ is a potentially interesting set. The set $\{AB\}$ has support $\frac{3}{4} = 0.75$ and is therefore considered large. The confidence of $\{A\} \rightarrow \{B\}$ is $\frac{0.75}{0.75}$ and is greater than γ, however, $\{B\} \rightarrow \{A\}$ is not considered interesting in this case because the confidence $\frac{0.75}{1}$ is less than γ. [Mannila *et al.* 1994]
A large itemset is described as monotone with respect to its subsets ([Mannila *et al.* 1994]) because all subsets of the large set must be large themselves. This is sometimes called the *downward closure property*. In the example set because $\{AC\}$ and $\{BC\}$ are not large then $\{ABC\}$ cannot possibly be large. If, however, σ was lowered to 0.4 then $\{AB\}$, $\{AC\}$ and $\{BC\}$ would be large and $\{ABC\}$ would be considered as a candidate set (and in the case of the example set be a large supported set). It is possible to argue that in the example "anything" implies B, however in a "real" situation support for an itemset is seldom 100%.

Association rules do not have monotonicity properties with respect to the contraction or expansion of the left-hand side: if $A \rightarrow B$ holds, for some support threshold, then $AC \rightarrow B$ does not necessarily hold. For example if the support threshold (σ) for the dataset in Figure 3.1 was set to 0.6 then $A \rightarrow B$ holds, clearly, if C was then added ABC would not be supported and no rule could be generated.

If $AC \rightarrow B$ holds then $A \rightarrow B$ does not necessarily hold with sufficient confidence. For example from Figure 3.1, support threshold = 0.4. confidence threshold = 0.6; the rule $AB \rightarrow C$ holds with support 0.5 and confidence $\frac{ABC}{AB} = \frac{0.5}{0.75} = 0.66$. But, $B \rightarrow C = \frac{BC}{B} = \frac{0.5}{1} = 0.5$. Further examples of association rule properties can be found in Ng *et al.* [1998].

The notion of confidence can be misleading. If a rule has a confidence of 1 then the rule cannot be improved upon. However, if it is less than 1 an *improvement measure*[3] may be applied, (p = *probability*) improvement = $\frac{p(AB)}{p(A)p(B)}$. Brin *et al.* [1997] describe implication rules which measure *conviction*, (defined as $\frac{p(A)p(\neg B)}{p(A\neg B)}$) rather than confidence or improvement.

3.3 Association Rule Algorithms

As described in section 3.1 the computationally intensive part of association rule mining is generating supported itemsets. The following sections provide an overview of some of the existing methods for achieving this aim. This thesis is primarily concerned with the generation of supported itemsets and not the production of rules from itemsets.

3.3.1 Exhaustive Methods

The use of exhaustive methods to extract association rules is infeasible in cases where the number of attributes in the dataset exceeds about 30; however, they can provide a benchmark against which the performance of other methods may be measured. The simplest form of such an algorithm would be;

for each record (j) in database do

begin

[3]Sometimes called *interest, or lift.*

```
        for each subset of j do
            add 1 to support count for that subset
    end
```

Four properties of this algorithm are of interest when considering its performance;

1. The number of database passes required,
2. The number of database accesses required,
3. The number of computational steps,
4. The memory requirement.

For a database of m records the algorithm involves a single database pass requiring m database accesses to retrieve the records. If it is assumed that the database consists of binary vectors, then enumeration of the subsets is straightforward and will involve a maximum of $m \times 2^n$ computational steps. The actual number of steps would depend on the number of attributes present in each record. For each subset the binary encoding may be used to reference a simple array of *support counts* so incrementing is trivial, however, the array size will need to be 2^n. For small values of n this method is simple and efficient. However, once n becomes even moderately large i.e. ≥ 30 the number of computational steps makes this infeasible. Even in cases where the number of attributes present in a record is small the size of the support count array is a limiting factor.

For these reasons, practicable algorithms for computing association rules, generally proceed by generating support counts only for those sets which are identified as potentially interesting, rather than attempting to compute support for all the database subsets. To assist in identifying these candidate sets it is helpful to observe that the subsets of an attribute set may be represented as a lattice. A lattice of this form is shown in Figure 3.2.

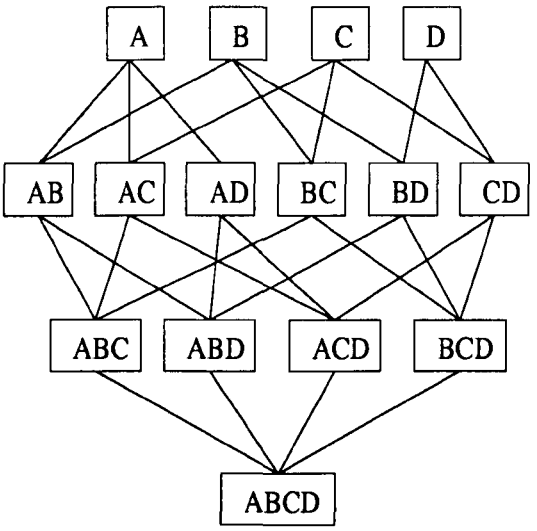


Figure 3.2: Lattice of subsets of $\{A, B, C, D\}$

For any set of attributes to be *large*, i.e. to have support exceeding the required threshold, it is necessary for all its subsets to be large. For example, a necessary (although not sufficient) condition for the set ABC to be considered as interesting is that AB, AC and BC are all large which in turn requires that each of A, B, and C are supported at the necessary level. This observation provides a basis for pruning the lattice of subsets to reduce the search space. If it is known that D is not supported then it is no longer necessary to consider AD, BD, CD, ABD, ACD, BCD or $ABCD$. Algorithms that proceed on this basis reduce their requirement for storage and computation by eliminating candidates for support as soon as they are able to do so. The tradeoff for this is usually greater access to the database in a series of passes.

3.3.2 Overview of AIS and SETM

In 1993 two methods of generating association rules were published: the AIS algorithm [Agrawal *et al.* 1993] and the SETM algorithm [Houtsma and Swami 1993]. The AIS algorithm provided a framework for the extraction of association rules

upon which many subsequent methods have built;

- define a set of candidates
- count the support for the candidates in a database pass
- use the supported sets to define a new candidate set
- continue until all large sets are found

Both AIS and SETM generated and counted candidate itemsets as the database was read and used the notion of *extending* large itemsets. For example, in pass k of the database if a transaction t is read it is determined which of the large itemsets L, generated in the previous pass i.e. L_{k-1}, are present in t. Each large itemset in L_{k-1} is extended with all the large items present in t that are later in the lexicographic ordering than any of the items in L_{k-1}. This led to a significant number of candidate itemsets being generated that would later turn out to be small. In the tests reported only relatively small datasets (63 attributes and 46,873 transactions) were used. SETM, which used SQL, had a further disadvantage in that it used TIDs (transaction IDs) to generate candidates, which required repeated sorting at the end of each database pass.

3.3.3 Breadth First Lattice Traversal

The most influential association rule mining algorithm is the Apriori algorithm developed by Agrawal and Srikant [1994], which is the basis for much of the work in the field. This algorithm, and its many variants, all proceed by examining the search space in a breadth first manner.

The paper containing the AIS algorithm was published in 1993; one year later Agrawal published his Apriori algorithm. It was significantly different from AIS in its candidate generation procedure and as such, allowed much larger datasets to

be mined. As with AIS and SETM the number of database passes required was still k+1, where k is the size of the largest supported set. Apriori was designed with datasets containing 1000 attributes and greater than 100,000 transactions in mind. The paper [Agrawal and Srikant 1994] contained three algorithms, Apriori, AprioriTid and AprioriHybrid the latter being a union of the two former algorithms. Apriori and AprioriTid generate the candidate itemsets to be counted in a pass by using only the itemsets found large in the previous pass, without considering the transactions in the database. The basic intuition is that any subset of a large itemset must be large[4].

Apriori

The algorithm proceeds in a breadth first manner as follows;
L_k = large (supported itemsets) of size k.
C_k = candidate itemsets. Initially, the set C_1 consists of the individual attributes in the dataset. The kth cycle proceeds as ($for\ k = 1, 2, \ldots, until\ C_k = emptyset$):

1. Perform a pass over the database to compute the support for all members of C_k.
2. From this, produce the set L_k of interesting sets of size k.
3. Derive from this the candidate sets C_{k+1}, using the downward closure property, i.e. that all the subsets of any member of C_{k+1} must be members of L_k.

The following version of the algorithm is taken from Agrawal and Srikant [1994];

L_1 = {Large 1-itemsets}
for ($k = 2$; $L_{k-1} \neq \emptyset$; k++) do begin

[4]Some authors use the term *downward closure property* (see section 3.2)

C_k = apriori-gen(L_{k-1});
forall transactions $t \in D$ do begin
C_t = subset(C_k, t);
forall candidates $c \in C_t$ do
c.count++;
end
$L_k = \{c \in C_k \mid c.count \geq minsup\}$
end
Answer = $\bigcup_k L_k$;

The *apriori-gen* function takes as its argument L_{k-1}, the set of all $(k-1)$-itemsets. It returns a superset of the set of all large k-itemsets. The *join* step works as follows, joining L_{k-1} with L_{k-1};

insert into C_k
select $p.item_1, p.item_2, \ldots, p.item_{k-1}, q.item_{k-1}$
from L_{k-1} p, L_{k-1} q
where $p.item_1 = q.item_1, \ldots, p.item_{k-2} = q.item_{k-2}$,
$p.item_{k-1} < q.item_{k-1}$

Finally the *prune* step deletes all itemsets $c \in C_k$ such that some $(k-1)$-subset of c is not in L_{k-1};

forall itemsets $c \in C_k$ do
forall $(k-1)$-subsets s of c do
if($s \notin L_{k-1}$) then
delete c from C_k

The effectiveness of the candidate generation used in the Apriori algorithm can be contrasted with that of AIS in the following example taken from the 1994 paper: Let $L_3 = \{\{ABC\},\{ABD\},\{ACD\},\{ACE\},\{BCD\}\}$. After the join step C_4 would be $\{\{ABCD\},\{ACDE\}\}$. The prune step would delete itemset $\{ACDE\}$ because the itemset $\{ADE\}$ is not in L_3; only $\{ABCD\}$ is left in C_4.

AIS would proceed as follows when the transaction {ABCDE} was read: in the fourth pass two candidates, $\{\{ABCD\},\{ABCE\}\}$ would be generated by extending that large itemset $\{ABC\}$. An additional three candidate itemsets would be generated by extending the other large itemsets in L_3, leading to a total of 5 candidates for consideration in the fourth pass. Apriori generates and counts only one itemset $\{ABCD\}$, because it concludes *a priori* that the other combinations cannot have minimum support.

Hash Tree Function

Candidate itemsets generated by the Apriori algorithm, are stored in a hash tree. A node of the hash tree either contains a list of itemsets (leaf node) or a hash table (interior node). In an interior node each bucket of the hash table points to another node. The root is defined to be at depth 1, and interior nodes of depth d point to nodes at depth $d + 1$. The itemsets are stored in the leaves. When an itemset c is added, starting at the root the tree is traversed until a leaf node is reached. At an interior node at depth d the branch to follow is decided by applying the hash function to the *dth* item of the itemset. All nodes are initially created as leaf nodes. When the number of itemsets in a node exceeds a specified threshold the leaf is converted into an interior node.

Starting from the root all candidates in a transaction t are found as follows: if a leaf node is encountered and it contains itemsets that are in t references are added to the answer set (i.e. the support is incremented). If an interior node is encountered and

it has been reached by hashing on item i each item in t that comes after i is hashed recursively and the procedure applied to the node in the corresponding bucket. For the root every item in t is hashed. The following Figure 3.3 illustrates a small hash tree containing candidate 3 itemsets.

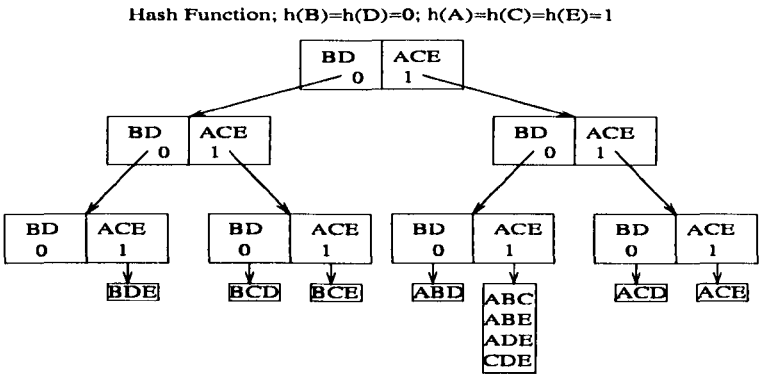


Figure 3.3: An example hash tree structure from Zaki *et al.* [1997]

AprioriTid

AprioriTID differs from the standard Apriori algorithm in the number of database passes made; recall that Apriori makes $k + 1$ passes over the database. AprioriTID makes only one pass over the database to count support; subsequently the candidate set C_k is used as a replacement for the database. Each member of C_k is of the form < TID, $\{X_k\}$ >, where each X_k is a potentially large k-itemset present in the transaction with the identifier TID. For $k = 1$, C_1 corresponds to the database although conceptually each item i is replaced by the itemset $\{i\}$. The member of C_k corresponding to transaction t is $< t.TID, \{c \in C_k \mid c \text{ contained in } t\} >$. If a transaction does not contain any k-itemset then C_k will not have an entry for this transaction. Thus the number of entries in C_k may be smaller than the number of transactions in the database, especially for large values of k. In addition, for large values of k, each entry may be smaller than the corresponding transaction because very few candidates may be contained in the transaction. These two factors have

a considerable effect on the algorithm's performance. If the number of candidates reduces rapidly, to a point where they can fit in main memory, the algorithm is superior to Apriori. However, for small values of k, each entry may be larger than the corresponding transaction because an entry in C_k contains all candidate k-itemsets contained in the transaction. This means that the number of candidates generated in the early passes of the algorithm may exceed the size of the original dataset. This problem is at its worst if a large number of 2-itemsets are supported. The following example uses Figure 3.4 to illustrate how AprioriTID generates more candidates at the 2-itemset stage than Apriori. If support threshold is set to 1, then all itemsets are supported at the 1 level. Aprori would generate 6 candidate 2-itemsets i.e. AB, AC, AD, BC, BD and CD. AprioriTID would generate 12 candidate 2-itemsets i.e. TID 1; AB, AD, and BD, TID 2; AC AD and CD, TID 3; BC, BD and CD, TID 4; AC, AD and CD.

TID	A	B	C	D
1	1	1	0	1
2	1	0	1	1
3	0	1	1	1
4	1	0	1	1

Figure 3.4: Sample database II

AprioriHybrid

AprioriHybrid uses the two algorithms Apriori and AprioriTID to generate rules from a dataset. In the early stages of the algorithm Apriori is used when the number of candidate sets is high. AprioriTID is activated when the number of candidate sets is deemed small enough to fit into main memory. The switching point between the two algorithms is critical to the performance because once the switch is made all subsequent candidates generated in each pass must fit in main memory. The performance of AprioriHybrid compared to Apriori depends upon how the size of

the candidate sets decline. If the candidate set remains large until nearly the end of the process then there is a sudden drop in candidate set size little advantage is gained from the switch.

The Partition Algorithm

Published in 1995 the *Partition* algorithm [Savasere *et al.* 1995] attempts to reduce the number of database passes made to extract all association rules. Partition has the advantage that the number of database passes required to generate all association rules is reduced to two. The algorithm is based on the idea that the reason the database needs to be scanned a number of times is that the number of possible itemsets to be tested for support is exponentially large if it must be done in a single scan. If, however, a small set of potentially large itemsets is known from a previous scan, then the database can be scanned once to find the support for these sets. The database is divided into a number of non-overlapping partitions, for the first database pass. Each partition is scanned and all locally frequent candidate itemsets are generated (the support count for each partition is a proportion of the total support count). Any locally frequent candidate itemsets found are then used to generate global candidate itemsets. The support of the global candidate sets is then derived by the second database pass.

Instead of using a hash tree to store the candidates these are stored in an inverted form of the database structure. This works as follows; associated with every itemset is a structure called a *tidlist*. The tidlist for itemset i contains the TIDs of all transactions that contain itemset i. The TIDs are kept in sorted order. The cardinality of a tidlist of an itemset divided by the total number of transactions gives the support for that itemset. The tidlist for a candidate k-itemset is generated by intersecting the tidlists of the two (k-1) itemsets that were used to generate the candidate k itemset. For example, consider the following database (Figure 3.5); The TID lists associated with $A = \{10,20\}$ and with $B = \{10,20,30\}$. The support for itemset $AB = A \cap B$

TID	A	B	C
10	1	1	0
20	1	1	0
30	0	1	1
40	0	0	1

Figure 3.5: Sample database III

= {10,20} = 2. When support is counted in this way the database is said to have a vertical layout[5] as opposed to horizontal. Using the intersection of TID lists has the advantage that hash structures are not required for support counting, but, when the itemsets are small, many more intersections need to be made than count increments in a hash tree. If the dataset contains dense data then the TID lists would become very large very quickly. For example, if the dataset contained 100,000 transaction records and the average record contained 40 supported attributes the TID list for candidate 2-itemsets would contain $\frac{39\times40}{2} \times 100,000 = 78$ million elements. The number of partitions within the database is important in that, if there are too many then there will be a large number of locally large candidate itemsets generated that will later turn out to be small.

Direct Hashing and Pruning (DHP)

One of the main problem areas within association rule mining is the number of candidates generated during the early stages of the process. If L_1 is large then $\binom{|L_1|}{2}$ becomes an extremely large number. This problem was addressed by Park *et al.* [1995] using the DHP (direct hashing and pruning) algorithm. DHP uses an additional hash tree structure to reduce the size of C_2. In addition to the hash table DHP also prunes the database after each scan to reduce the search space for the next scan.

[5]Sometimes called *inverted or decomposed storage structure.*

Reducing C_2

The hashing function in DHP works as follows: when the support for candidate k-itemset is counted by scanning the database the algorithm counts all the occurrences of the itemsets of size $k+1$ within the transaction and hashes these into a hash table, this is a type of "advanced lookup" function. Each bucket in the hash table contains a number representing how many itemsets have been placed in that bucket so far. Once the database scan is complete a bit vector is constructed to represent the total number of sets in each bucket. If the number exceeds the support level a '1' is placed in the bucket otherwise a '0'. Every item that passes the hash filter i.e has a 1 in its bucket, is placed into another hash tree of the type used in the Apriori algorithm. The following two figures are from Park *et al.* [1995]. The smaller number of 2-

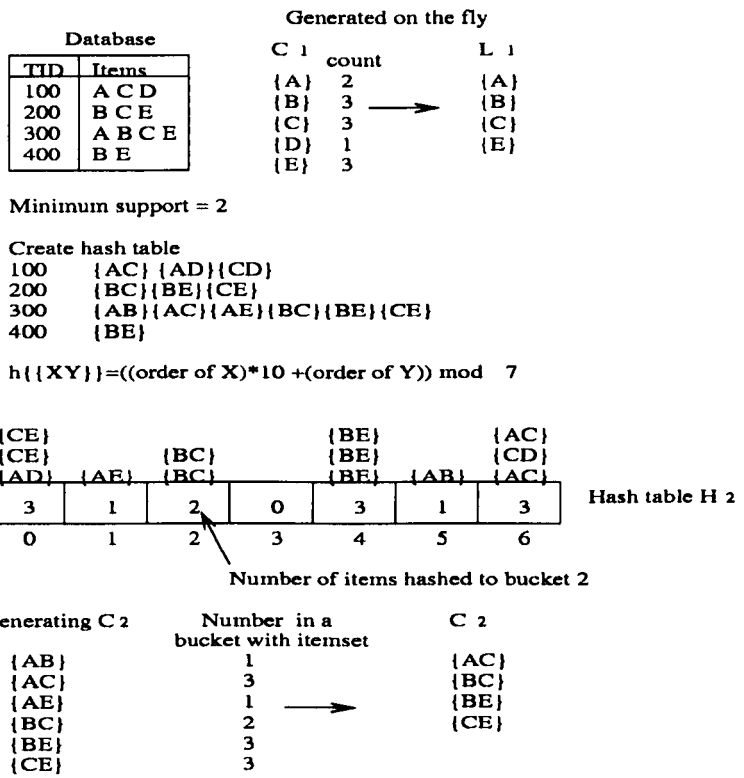


Figure 3.6: An example hash tree structure and generation of C_2

itemset candidates is beneficial when the database pass is made to count support for

these candidates; obviously the smaller number of candidates the less memory and computational effort required. However, as can be seen from Figure 3.6, the itemset D which is below the support level is included in the hash table H_2 as members of the sets AD and CD when clearly they cannot be supported 2-itemsets. This occurs because of the "advanced lookup" function described above. Furthermore, the construction and maintenance of the additional hash tree requires memory and computational effort.

Reducing Database Transactions

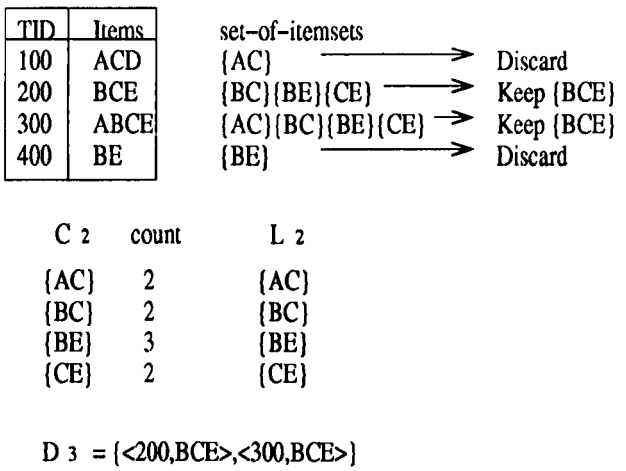


Figure 3.7: An example of L_2 and database D_3

DHP attempts to restructure the database after each pass to reduce the search space in subsequent passes. In Figure 3.7 transaction 100 contains only a single candidate itemset AC in L_2, as shown in the set-of-itemsets. There are no 3-sets in transaction 100 so it is discarded. In transaction 300 the only possible supported 3-set is BCE and therefore A is eliminated from the transaction. Restructuring the database after each pass obviously reduces the search space for subsequent passes, however, for large databases restructuring will undoubtedly add to the execution time of the algorithm.

Sampling Algorithms

After the publication of the AIS algorithm researchers at the Department of Computer Science at Helsinki University attempted to improve on its performance [Mannila *et al.* 1994], and also develop association rule mining using inverted structures and a general purpose DBMS [Holsheimer *et al.* 1995]. With the introduction of Apriori the work on improving the AIS algorithm was quickly abandoned and the target was now to improve on Apriori. The method proposed by Toivonen [1996] was to take a sample of the database from which is derived a candidate set for the full database search. To ensure (with high probability) that this candidate set contains all the actual frequent sets, two devices are used. First, the support threshold is lowered when the database sample is processed, leading to a candidate set S which is a superset of the actual (locally) frequent set. Secondly, the set is further extended by adding its *negative border*, i.e. those sets that are not members of S, but all of whose subsets are included in S. This extended set is used as the candidate set for a full database pass, which, if no members of the negative border are found to be frequent, completes the search.

This method of support counting trades a larger candidate set for reduced database passes. However, if the distribution of the data is not consistent throughout the dataset then sampling risks either missing many large itemsets outside the sample area or producing many candidate sets from the sample that are only large within the sample.

Work by Brin and Page [1998] on mining web pages also uses sampling to mine data which is large and also contains a great many items per transaction (200+). In pursuit of this goal completeness is sacrificed for the ability to penetrate the data in a realistic time. The search is guided by the use of a concept called the *heavy edge property*, which views the search space as a hyper-graph. The edges of the graph are described as *heavy* if they exceed a user defined *weight*. If an edge is found to

be heavy then the edges that are its neighbours are likely to be heavy too.

Dynamic Itemset Counting

The DIC algorithm [Brin *et al.* 1997] aimed at making fewer database passes than Apriori and producing smaller candidate sets than sampling methods. Reducing the number of database passes is achieved by counting candidates of various sizes in one pass of the database. The reduction in the size of candidate sets over sampling methods is achieved because DIC does not need to lower the support threshold and examine candidates in the *negative border*. The reduction in the number of database passes made by DIC provides an advantage over Apriori. Brin attempts to mine census data which although containing fewer records, contains a higher proportion of binary 1's per record. In some cases attributes appear in 95% of records. This more dense data could only be analysed with support thresholds of 30-50% rather than the "usual" support thresholds on supermarket data of 0.5-2%. DIC operates in exactly the same way as Apriori, with the exception of database passes, and therefore suffers from the same problems when confronted with dense datasets.

3.3.4 Depth First Lattice Traversal

Several researchers have tried to improve association rule mining techniques by attempting different strategies for traversing the search space. Both Zaki *et al.* [1997] (and in [Zaki 1997]) and Bayardo [1998] use "look ahead" techniques to enable depth first searching to proceed.

Clustering Algorithms

Zaki *et al.* [1997] present a number of algorithms based on finding *maximal* large itemsets. An itemset is *maximal large*[6] if it has no superset that is large. Any large

[6]Zaki uses the term *maximal frequent.*

itemset is a subset of a maximal large itemset. Once all the maximal large itemsets are found then they can be used to generate all rules because of the monotonicity properties with respect to contraction (as described in section 3.2). The algorithms use equivalence class clustering and maximal uniform hypergraph clique clustering to identify maximal itemsets. An equivalence class is generated in the following manner; if L_k consists of $\{A, B, C, D\}$, then the equivalence class for A = AB, AC, AD, then for B = BC, BD etc. For k = 1 the entire item universe is the potential maximal itemset, however once $k \geq 2$ maximal frequent itemsets can be generated that are more specific. Each maximal equivalence class or clique defines a search space. This is then traversed in breadth first, depth first or a hybrid breadth first/depth first manner. Traversing the search space can be done either using equivalence class itemsets or the clique generated itemsets. The equivalence class itemsets are less specific but require less computation to generate. The clique generated itemsets require more computation because L_2 needs to be produced prior to the cliques being derived, however, more accurate potential maximal itemsets are produced. One of the problems with this method is that the itemsets generated are only "potentially" maximal. If they indeed turn out to be maximal then the search is finished, but if not then all the subsets of the potential maximal itemsets must be checked for support until "true" maximal itemsets are found. Zaki attempts to address this problem by using a hybrid breadth first/depth first search. This method provides an improvement over Apriori because it searches for maximal itemsets first and when found the maximal large itemsets and all their subsets can be removed from the search space. This leaves a smaller space for the breadth first part of the algorithm to search. Using clustering in this manner depends critically on being able to generate small clusters, invert the database and compute the 2-itemsets prior to the algorithm starting. Large clusters derived from dense data, will increase the search space significantly and require a huge number of intersections for support counting.

Rule Generation Using Trees

Set Eumeration Trees

The equivalence classes described above bear a close relationship to the set enumeration trees proposed by Rymon [1992] and developed in Rymon [1993] and Rymon [1996]. Initially developed for generalizing decision trees, the set enumeration framework was proposed for use with association rule mining by Bayardo [1998] and expanded in Bayardo *et al.* [1999]. The set enumeration tree imposes an ordering on the set of items and then enumerates the sets of items according to the ordering as illustrated in Figure 3.8.

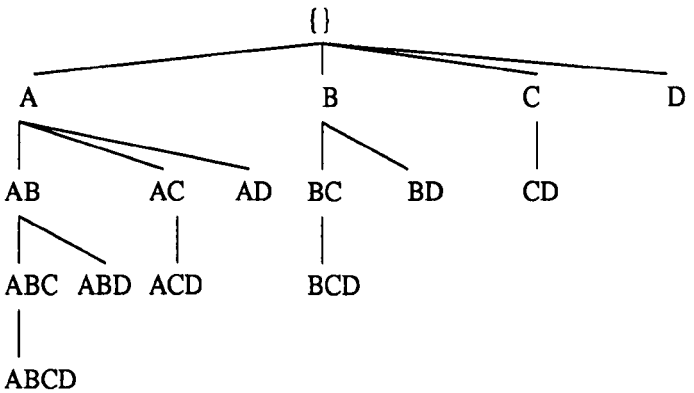


Figure 3.8: Set enumeration tree for {A,B,C,D} ordered lexicographically

Bayardo's Max Miner algorithm searches for maximal large itemsets and proceeds in a similar manner to Apriori in that it must scan the database multiple times. The fact that the algorithm searches for maximal large itemsets enables it to adopt a dual pruning strategy, it prunes both supersets and subsets at the same time.

The candidate itemsets[7] are described as containing the head denoted $h(g)$ representing the item enumerated by the node and the tail $t(g)$ an ordered set containing everything not in the head. In Figure 3.8 the node enumerating itemset {A} has

[7]Bayardo calls them *candidate groups*.

$h(g)$ = {A} and $t(g)$ = {B,C,D}. Superset pruning proceeds by determining if a maximal itemset is a large maximal itemset during a database scan. If it is then all its subsets must be large so the node in the enumeration tree is not expanded into its sub-nodes. Subset pruning is achieved by determining if an itemset $h(g) \cup \{i\}$ for some $i \in t(g)$ is not large. If $h(g) \cup \{i\}$ is not large then any head of sub-node that contains i will also be not large. Subset pruning removes any such item $\{i\}$ from the candidate group before expanding its sub-nodes.

Bayardo uses an itemset re-ording policy to ensure the effectiveness of superset pruning. The idea is to ensure that as many candidate groups as possible contain items that are large. This is based on the notion that items with high support are more likely to be part of large itemsets. Items that appear later in the ordering will appear in the most number of candidate sets. As can be seen from Figure 3.8 item D appears in the head or tail of every node.

Bayardo's approach is primarily directed at mining more dense data than Apriori. This is achieved in a similar manner to the methods of Zaki *et al.* [1997]. The subset and superset pruning idea has the desired effect of reducing the search space and Bayardo's method improves over Zaki's, the former attempts to identify maximal large itemsets during the search, the latter only seeks maximal large itemsets during initialization. Bayardo's method however, still requires multiple passes of the database.

3.3.5 ADtrees

The ADtree [Moore and Lee 1998], [Anderson and Moore 1998] as a data structure was initially developed as a generic framework to facilitate fast counting and query matching. The tree structure aims to represent all possible queries and counts.

The tree has two primary features which facilitate sparse representation;

1. For any query that matches zero records a NULL is stored,

2. When an attribute has a MCV (see below) a NULL is stored.

The database attributes contain integer values in the range $\{1 \ldots n\}$. A query is a set of $(attribute = value)$ pairs in which the left hand side form a subset of the attributes. The count of a query $C(query)$ is the number of records matching all the $(attribute = value)$ pairs in $query$. The tree is constructed starting at the root which is described by Moore as an "ADnode", where each attribute has a "don't care" (*) value within the ADnode. This points to a "Vary" node for each attribute which lists the most common value (mcv) for that attribute. Vary nodes then point to AD nodes which list and contain the counts of the attributes which do not contain the mcv. From the example (Figure 3.9 taken from Moore and Lee [1998]) it can be seen that the mcv, in the database, for $A = 3$, when the tree is constructed the Vary node contains the mcv for A (in this case 3). The remaining, smaller values for A are enumerated in 'child' ADnodes while the child ADnode that "would" have the value $A = 3$ in it points to NULL because the fact that mcv for A = 3 is stored in the Vary node above.

The ADtree is constructed once and is intended as a fast method of answering user defined queries as opposed to discovery driven rule mining. It does however, illustrate an interesting structure for representing a database. For example, the combination of a set enumeration tree using the sparse representation of the ADtree could be a way to reduce the size of a database, that would allow the data to be held in main memory. This in turn would reduce the access times required when counting support of records in the database.

3.3.6 Parallel and Distributed Association Rule Mining

The number of candidate sets generated in the second pass of databases has led some researchers to propose using parallel processing to address the problem. The

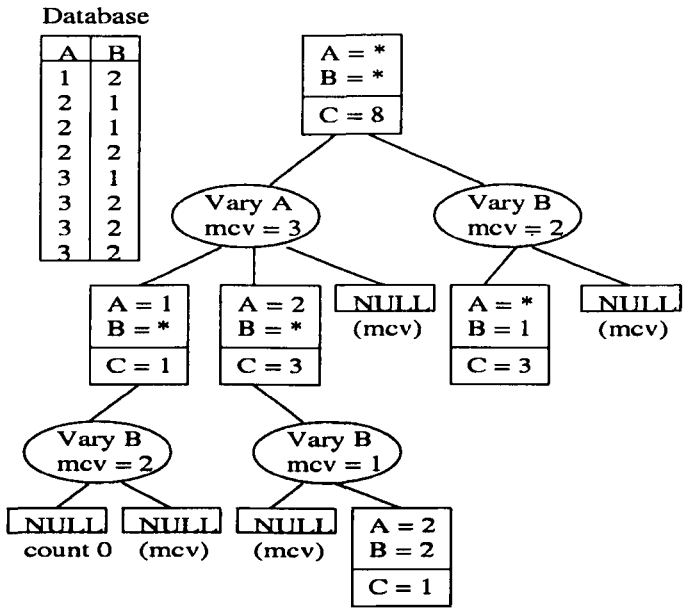


Figure 3.9: An AD tree and sample database

basis for most of the parallel algorithms is Apriori and the target area for improvement is the support counting phase. Research by Shintani and Kitsuregawa [1996] calculated support within several hash tables (one per processor) then integrated the hash tables to produce a result for that database pass. Parthasarathy *et al.* [1998] has attempted to reduce/eliminate *false sharing*, on shared memory machines i.e. allocating memory in such a way that only the correct processor accesses the correct memory area. Further research in this area has been carried out by Tamura and Kitsuregawa [1999], Han *et al.* [1997] and Shintani and Kitsuregawa [1998].

Distributed algorithms for computing association rules have been proposed by Cheung *et al.* [1996a] Cheung *et al.* [1996b]. In these algorithms local pruning and global pruning are used to reduce the number of messages passed. Mining association rules over large distributed databases contains all the problems/issues associated with mining rules within one database but the additional complication of selecting the correct message passing strategy.

3.3.7 Sequential Analysis

Given a set of data-sequences, each of which is a list of transactions ordered by the transaction time, the problem of mining sequential patterns is to discover all sequences with a user specified minimum support. Examples can be found in Thomas and Sarawagi [1998] and Hatonen *et al.* [1996]. Each transaction contains a set of items. A sequential pattern is an ordered list (sequence) of itemsets. The itemsets that are contained in the sequence are called the elements of the sequence. For example, `((computer,modem) (printer))` is a sequence with two elements `(computer,modem)` and `(printer)`. The support for a sequential pattern is the number of data-sequences that contain the sequence. A sequential pattern can be further qualified by specifying minimum and /or maximum time gaps between adjacent elements, and a sliding time window within which items are considered part of the same sequence element. Sequential patterns are essentially associations over temporal data. Whilst some of the datasets under consideration in this thesis contain time stamps, many do not. The current motivation for this research is the extraction of supported sets: once this has been achieved satisfactorly then the integration of time into the attribute set could be explored.

3.4 Evaluation

The description of association rule mining techniques above illustrates that since 1993 and the introduction of the AIS algorithm many research efforts have been devoted to improving rule generation. The Apriori algorithm remains the foundation of *many* subsequent algorithms and the basis against which *all* subsequent algorithms are judged.

The number of database passes required by Apriori is k+1, where k is the number of attributes in the largest supported itemset. The number of database accesses

required in each pass is m (number of records in the database). The maximum number of computational steps required in each database pass is

$$\sum_{i=1}^{m} \binom{k_i}{j} \tag{3.1}$$

This is for each record the number of subsets of size j in the set of size k_i where j is the jth pass of the database. The memory requirement is largest number of candidate itemsets generated during a pass of the database, and a hash tree structure to contain and update the candidates. The number of database passes required to generate rules has, in cases such as Partition [Savasere *et al.* 1995] and Sampling [Toivonen 1996] and and Dynamic Itemset Counting, [Brin *et al.* 1997], been reduced. However, the number of candidates generated with Partition and Sampling increases and with DIC the computation involved in processing each record is increased. Partition requires more computational steps in the early database passes because the TID-lists of small itemsets provide little information about associations among itemsets. For example, a database with 10,000 transactions, 500 large (single) itemsets and an average transaction size of 10 items has TID-lists of average size 200. To find all the candidate 2-itemsets intersections must be made between each pair of items, this requires $\binom{500}{2} \times (2 \times 200) \approx 50$ million operations. Using hash tables only the pairs appearing in a transaction need to be formed and their count incremented, requiring $\binom{10}{2} \times (10{,}000) \approx 0.5$ million operations. Attributes that are very common can also cause extremely large TID lists, if an attribute appears in all transactions its TID list will be as big as the original dataset.

Both Zaki *et al.* [1997] and Bayardo [1998] attempt to look ahead and reduce the search space by early pruning. Zaki *et al.* [1997] looks ahead during the initialization phase of the algorithms and uses a pre-processing step to obtain support for all the 2-itemsets (requiring two database passes before the algorithm starts). Zaki's computational steps increase (over Apriori) because of the intersection problem (described above). Max-Miner [Bayardo 1998] can complete its search in fewer database passes than Apriori if it locates all maximal large itemsets early in the

search. If, however, the maximal large itemsets are not located early in the process (because they are not supported), then Max-Miner can take as many database passes as Apriori. Early location of, at least some of, the maximal sets is important for Max-Miner to facilitate the reduction in search space and number of computations required.

3.5 Summary

The first part of this chapter explained how association rules are arrived at and some of the different evaluation criteria used to judge the supported itemsets. In the second part many different approaches for deriving the support count of an itemset were examined. These were sub-divided into the methods mainly based on the Apriori algorithm and those that adopted a slightly different strategy.

When considering the databases contained within the facilities management division of an organization there may be attributes that occur with great frequency e.g. job category, sex, age, a frequency not usually found within supermarket datasets. The implications of this are that frequent sets may contain large numbers of attributes, so that an Apriori type algorithm would require many database passes and generate many candidate sets. Methods such as Max-Miner encounter problems when the potential maximal set is very large but the actual supported sets are small because the search space does not shrink quickly. Methods such as Moore's ADtrees and Bayardo's Max-Miner provide an interesting start point for the exploration of Facilities Management databases.

Chapter 4

The Partial Support Tree

4.1 Introduction

The previous three chapters have provided background material.

- **Chapter 1** described the motivation for the research, and the emergence of the field of knowledge discovery in databases. The domain of facilities management was introduced and highlighted as a previously unexplored area.
- **Chapter 2** introduced the broad area of knowledge discovery in databases and specifically discussed data mining as a key area of research.
- **Chapter 3** examined in detail data mining methods for the extraction of association rules developed over the last eight years and provided some benchmarks (database passes, computational effort, etc.) by which algorithms could be judged.

The remaining chapters of this work aim to establish the thesis that the pre-processing of data using specialised data structures can offer significant advantages when mining association rules. The following chapters will seek to address some of the issues arising from using data mining (Chapter 2) within the domain of facilities management (Chapter 1) building upon previous association rule mining techniques (Chapter 3). This chapter deals with the construction of a data structure described as the P-tree or partial support tree. The motivation behind the development of the

A	B	C	D	E	F	G
1	1	1	0	1	1	0
1	1	1	0	1	1	0

Table 4.1: Sample database IV

P-tree was to reduce the number of database passes required to generate support counts, this being one of the primary issues in association rule mining research, and to reduce the combinations involved in performing these counts. All the algorithms described in the previous chapter that repeatedly read the database take no account of any of the previous transaction records they have processed. As each transaction record is read all the subsets of size k, (k being the number of the database pass) within the record must be examined. Each transaction record, therefore, is treated as a unique entity. For example, if two records follow each other in the dataset, as shown in table 4.1, then during the generation of candidate itemsets containing pairs each record will require $\frac{5\times4}{2}$ increments within the hash tree structure. Obviously both records are exact copies of each other but this fact is not taken into account. Clearly if a database contains many duplicate records a great deal of computational effort can be wasted. One of the data structures to be used within this thesis is called the Partial Support Tree (P-tree). The P-tree aims to take advantage of any duplication found in the dataset, by pre-processing the data and storing it in an structured manner. The advantages gained when processing duplicated data will be illustrated later in this chapter. If the data contains no duplicate records, the summation part of the process should be enhanced because of the structural advantages that the P-tree delivers.

The general concept is to read the database once and dynamically create a data storage structure that could be contained in memory and therefore reduce the number of accesses to a disk resident database.

In order to understand why and how the P-tree was developed it is instructive to examine some of the experiments carried out earlier as part of this research using data structures that would allow the database to read, and the support for itemsets computed, in one pass.

4.2 Exhaustive Algorithms

As noted in the previous chapter (section 3.3.1) it is infeasible to compute association rules for any dataset containing more than 30 attributes using an exhaustive approach. To achieve a degree of understanding of some of the implementation details and to provide a start point for this research two exhaustive algorithms were developed. These, like all other work in this thesis, were programmed using ANSI C.

4.2.1 Brute Force 1 Algorithm (BF1)

The data structure used for this algorithm was a linked list of structures. Each structure contained a character string, an integer code and an integer representing the support for that code. The character string was restricted to the 26 letters of the alphabet. The integer code was derived from the bit pattern associated with an attribute. For example, when reading a bit pattern from left to right, A = 10000000 = (integer code 1), B = 01000000 = (integer code 2) and AB = 11000000 = (integer code 3). The support was represented as an integer. The algorithm proceeded in the following manner:

- Read the first line of the transaction dataset.
- From this line determine the number of columns/attributes in the dataset. Use this information to generate a linked list of structures, one structure for each combination of one or more attributes.

- Read the rest of the dataset one record at a time until the end of the dataset is reached. As each record is read compare its contents with the contents of the linked list by traversing the list. If the record contained a subset or exact match for an element in the list increment that element's support count by 1.
- On completion the linked list structure will contain details of all the support that the dataset provides for every possible combination of the attributes described by the table columns.

Test Data

The test data consisted of several datasets each containing 100,000 records generated by a synthetic data generator. The number of attributes in the datasets was 10, 11, 14 and 15. The generator created records randomly with an average percentage of binary 1's in each record set to either 20% or 80%.

Test Environment

All the tests within this thesis were conducted using a stand alone IBM PC clone with an Intel Celeron 300MHz processor, 128Mb of main memory and a 20Gb IDE secondary storage device. The operating system was Mandrake Linux version 7.2, kernel release 2.2.17-21.

Test Results

Table 4.2 provides a key for table 4.3.

From the results presented in table 4.3 it can be observed that as would be expected the memory requirements increase exponentially each time one attribute is added to the dataset. The table differentiates between node increments and node comparisons. The former occurs when a node is found to contain a subset or exact match of

A	Number of Attributes
P	Record density %
N	Nodes in list
M	Memory required (bytes)
I	Node increments (rounded)
C	Comparisons (rounded)
T	Execution time (secs.)

Table 4.2: Key for table 4.3

A	P	N	M	I	C	T
10	20	1023	45012	$10^6 \times 0.297$	$10^8 \times 1.023$	9
10	80	1023	45012	$10^8 \times 0.258$	$10^8 \times 1.023$	9
11	20	2047	90068	$10^6 \times 0.298$	$10^8 \times 2.047$	26
11	80	2047	90068	$10^8 \times 0.258$	$10^8 \times 2.047$	35
14	20	16383	720852	$10^6 \times 0.299$	$10^9 \times 1.638$	623
14	80	16383	720852	$10^9 \times 0.206$	$10^9 \times 1.638$	648
15	20	32767	1441748	$10^6 \times 0.698$	$10^9 \times 3.276$	962
15	80	32767	1441748	$10^9 \times 0.411$	$10^9 \times 3.276$	973

Table 4.3: Results from algorithm BF1

a record, the latter is a test performed on all nodes for all records. The results show that the more dense datasets have slightly increased execution times, this is because of the greater number of node increments. To ensure that the number of transaction records was not the limiting factor several tests were run with the above datasets containing 200,000 records. These tests showed a linear increase in the number of node increments and comparisons and a sub-linear increase in execution times.

4.2.2 Brute Force 2 Algorithm (BF2)

The limitations of the BF1 algorithm were apparent prior to its implementation, testing confirmed its poor performance and scalability. The purpose of this implementation was only to demonstrate the inherently exponential scaling of the problem. Although BF2 was also never expected to be a practical algorithm in terms of support counting over large numbers of attributes, it was felt that some improvements could be made to BF1. BF2 offered two advantages over BF1:

- The amount of memory required was reduced.
- The comparison of bit patterns was no longer necessary.

The first advantage was achieved by dispensing with the linked list data structure in which results were stored, using instead a 1-D array. The fact that each possible combination of attributes could be defined as a binary sequence then translated into an integer (as described in section 4.2.1) enables direct indexing into the array. Each index represents a particular configuration of attributes and the content, that configuration's support. The second advantage is a direct result of the first, because of the direct indexing it is no longer necessary to carry out comparisons between the records in the table and the elements of the data structure. The replacement method for this is to calculate each possible subset of each transaction record as an integer and use this to index into the array and increment that element's support. Computing

A	Number of Attributes
P	Record density %
S	Array size
M	Memory required (bytes)
I	Array updates (rounded)
T	Execution time (secs.)

Table 4.4: Key for table 4.5

all subsets of each transaction record replaces the need to search the linked list (as in BF1) but requires additional computation. BF2 works in the same manner as BF1 described above with the exception of the data structures and indexing.

Test Data

The same synthetic test data generation program was used to produce the datasets for testing BF2 as for BF1. As with BF1 attribute numbers were small but BF2 could process slightly higher numbers: the attribute numbers were 10, 11, 15, 20 and 21 (the first three enable direct comparison with BF1).

Test Results

Table 4.4 provides a key for table 4.5.

The results presented in 4.5 show, as expected, an exponential increase in the size of the array required to hold the results and a corresponding increase in memory requirement. As with the tests on BF1, several datasets were tested which contained 200,000 records these displayed a linear increase in the number of array increments and a sub-linear increase in execution times. BF2 showed a moderate improvement over the results for the the BF1 algorithm (table 4.3) in that, the number of attributes processed by BF2 was 21 compared with 15, and the execution time of BF2 was faster. An interesting observation when comparing the two algorithms is

A	P	S	M	I	T
10	20	1024	4096	$10^6 \times 0.297$	<1
10	80	1024	4096	$10^8 \times 0.258$	8
11	20	2048	8192	$10^6 \times 0.298$	<1
11	80	2048	8192	$10^8 \times 0.258$	19
15	20	32768	131072	$10^6 \times 0.698$	1
15	80	32768	131072	$10^9 \times 0.411$	136
20	20	1048576	4194034	$10^7 \times 0.149$	1
20	80	1048576	4194034	$10^{10} \times 0.226$	172
21	20	2097152	83886084	$10^7 \times 0.149$	2
21	80	2097152	83886084	$10^{10} \times 0.229$	236

Table 4.5: Results from algorithm BF2

the increase in execution time between the sets containing 20% density and 80% density. Algorithm BF1 had a slight increase in execution time between the two sets whereas BF2 displayed a significant increase in execution time. This was because the more dense dataset produced many more subsets and more subsets result in a greater number of array accesses and increments, BF1 did not suffer from this drawback because all the elements in the list were tested for the subset/exact match property regardless of the dataset density. BF2 scales better: it is only slightly faster for 10-11 attributes (80% density) but much faster for 15.

4.3 Partial Support

One of the main problems with both the brute force algorithms is that they compute and store every possible combination of the attribute set. Another is that an increase in the number of attributes by 1 doubles the potential search space and therefore the memory requirement. It may be observed that within any attribute set there may be a great many combinations of attributes that contain no support. Calculating the code combinations and allocating memory for these is a wasted effort. The exponential

scaling places a low upper bound on the number of attributes that can be considered.

The algorithms described in Chapter 3 all depend, to some degree, on a reduction of the search space by reducing the candidates to a number which can be retained in memory and counted feasibly. In most cases, each pass of the database proceeds by examining each record to identify all the members of the candidate set that are subsets of the record. This can be computationally expensive, especially when records are densely populated. In principle, however, it is possible to reduce this cost by exploiting the relationships between sets of attributes illustrated in the lattice (figure 3.2). For example, in the simplest case, a record containing the attribute set ABD will cause incrementation of the support-counts for each of the sets ABD, AB, AD, BD, A, B and D. Strictly, however, only the first of these is necessary, since a level of support for all the subsets of ABD can be inferred from the support-count of ABD. Let i be a subset of the set I (where I is the set of n attributes represented by the database). P_i is defined as, the *partial support* for the set i, to be the number of records whose contents are identical with the set i. Then T_i, the *total support* for the set i, can be determined as:

$$T_i = \sum P_j \qquad (\forall j, j \supseteq i)$$

This allows us to postulate an exhaustive algorithm for computing all (total) supports:

Algorithm A:

Stage 1: for all records j in database do
begin add 1 to $P(j)$
end;

Stage 2: for all distinct j found in database do
begin for all $i \subseteq j$ do

begin add $P(j)$ to $T(i)$

end

end

For a database of m records, the algorithm performs m support-count incrementations in a single pass (stage 1), to compute a total of m' partial supports, for some $m' \leq m$. Stage 2 involves, for each of these, a further 2^i additions, where i is the number of attributes present in the set being considered. If the database contains no duplicate records, then the method will be less efficient than an exhaustive computation such as BF2 which enumerates subsets of each record as it is examined. Computing via summation of partial supports will be superior, however, in two cases. First, when n is small ($2^n \ll m$), then stage 2 involves the summation of a set of counts which is significantly smaller than a summation over the whole database, especially if the database records are densely-populated. Secondly, even for large n, if the database contains a high degree of duplication ($m' \ll m$) then the stage 2 summation will be significantly faster than a full database pass.

Computing partial supports as described above allows, in one pass of the database, to capture all the relevant information in a form which enables efficient computation of the totals, exploiting the structural relationships inherent in the lattice of partial supports.

4.3.1 Linked List Partial Support Algorithm (LLPS)

Both the brute force algorithms perform the set enumeration function and the support count function within the pass of the dataset; the idea behind partial support is that only some of the supports are counted during the initial pass through the dataset, section 1 of algorithm A. The remainder of the supported sets are computed from the linked list after it has been constructed, i.e. it is the duplicates within the

linked list that provide the support counts. Whilst this will require more computation the linked list (held in main memory), not the dataset will provide the support counts. The objectives of the algorithm were as follows:

1. Read the dataset in one pass and store the result in main memory.
2. Only store results for which records are present i.e. there is a transaction record in the dataset corresponding to a record in the structure.
3. Consider datasets of > 20 attributes.

The algorithm proceeded as follows:

- Read the first record in the table describing a transaction.
- From this determine the number of columns/attributes in the table and use this information to generate the initial structure in the linked list.
- Read the remainder of the table and for each transaction record:
  - If the transaction record is already represented by a node in the linked list increment the support for that node.
  - Otherwise create a new structure and add this to the list in such a way that the list is ordered according to the identification codes associated with the transaction records.

The brute force algorithms could not cope with attribute numbers greater than 20 and therefore any combination that required representation could be identified as a single 32 bit integer as described in section 4.2.1. The LLPS algorithm would be required to process numbers of attributes greater than 32 and therefore a single unsigned integer would not be sufficient. The codes were stored as arrays of unsigned integers. The number of elements in the array is dependent on the number of

attributes in the dataset and is calculated as follows (k = number of array elements):

$$k = \frac{number\ of\ attributes}{32} + \frac{31}{32}(rounded\ up\ to\ the\ nearest\ integer)$$

For example, if the following transaction record was read;

01001011010000000010100100001010 11001100

first 32 bits second 8 bits

The number of attributes in the transaction record above would be 40. Taking the leftmost digit to be the least significant, this would translate to the integer code 2703754658 51 (the spaces are for clarity) these two integers could then be stored in the array, (of unsigned integers). The remaining elements of each linked list structure was an integer for the support count and a "next" pointer, each requiring 4 bytes. Therefore the total storage for the algorithm is defined as:

$$u(8+k)bytes$$

where u is the number of unique transaction records in the dataset. In the worst case u will be equal to m (all records unique) and in the best case u would equal 1 (all records the same). The brute force algorithms took no account of any of the properties of the dataset except the number of attributes, because all possible combinations of the attribute set were calculated regardless of their support. This factor changed significantly with the LLPS: First, because the number of unique elements in the dataset was significant, as described above, and secondly, the ordering of the transaction records influenced the number of comparisons. For example, consider the following two datasets, both binary encodings of all the values between 1 and 15 (or an alternative description, the complete enumeration of the attribute set $ABCD$). The dataset on the left enumerates in ascending order, on the right descending order. When the LLPS algorithm constructs the linked list to store the transaction records the number of comparisons made to enable the new structure

A	B	C	D
1	0	0	0
0	1	0	0
1	1	0	0
0	0	1	0
1	0	1	0
1	1	0	0
1	1	1	0
0	0	0	1
1	0	0	1
0	1	0	1
1	1	0	1
0	0	1	1
1	0	1	1
0	1	1	1
1	1	1	1

A	B	C	D
1	1	1	1
0	1	1	1
1	0	1	1
0	0	1	1
1	1	0	1
0	0	1	1
1	0	0	1
0	0	0	1
1	1	1	0
0	1	1	0
1	0	1	0
0	0	1	0
1	1	0	0
0	1	0	0
1	0	0	0

Figure 4.1: Two datasets enumerating 1 to 16

to be added to the existing list is, 20 for the descending dataset but 105 for the ascending dataset.

Test Data

The LLPS algorithm was tested on files generated with the same synthetic test data generator as the brute force algorithms. The LLPS algorithm datasets were considerably smaller in terms of the number of transaction records. The algorithm was not able to process 100,000 records; 10,000 was found to be the maximum number for the various attribute numbers. The number of attributes, however, was increased to 50. As with the previous datasets for each attribute number, two data densities were generated, 20% and 80%. The attribute numbers increased in increments of ten from 10 to 50.

Test Results

Table 4.6 provides a key for table 4.7.

From the test results provided in table 4.7 it can clearly be seen that, unlike the brute force algorithms, increasing the number of attributes does not result in any

A	Number of Attributes
P	Record density %
N	Nodes in list
M	Memory required (bytes)
I	Node increments (rounded)
C	Comparisons (rounded)
T	Execution time (secs.)

Table 4.6: Key for table 4.7

A	P	N	M	I	C	T
10	20	55	660	9945	$10^5 \times 0.289$	2
10	80	887	10644	9113	$10^7 \times 0.406$	4
20	20	4886	58632	5114	$10^8 \times 0.154$	27
20	80	9939	119268	61	$10^8 \times 0.119$	30
30	20	9917	119004	83	$10^8 \times 0.246$	44
30	80	10000	120000	0	$10^8 \times 0.249$	45
40	20	10000	160000	0	$10^8 \times 0.253$	54
40	80	10000	160000	0	$10^8 \times 0.252$	56
50	20	10000	160000	0	$10^8 \times 0.251$	67
50	80	10000	160000	0	$10^8 \times 0.250$	67

Table 4.7: Results from algorithm LLPS

exponential growth. The field described as "node increments" serves two functions, it illustrates the number of node increments but also serves to show how many duplicate transaction records were in the dataset. It will be recalled from the algorithm description that when a transaction record is read either a new node is created *or* support for an existing node is increased. The table shows that when lower numbers of attributes were used there was a higher probability of duplication occurring within the dataset so the algorithm is efficient for these cases. As the chance of duplication diminishes the algorithm becomes less efficient and turns out to be a repeat of the original dataset in an ordered format but occupying much more storage space. Direct comparison with algorithms BF1 and BF2 is unjustified because LLPS only calculates partial support as opposed to total support.

The datasets for table 4.7 were all randomly generated. The following figure 4.2 illustrates the execution times when two datasets are presented as a random sets, sets sorted in ascending order and in descending order. The data sets selected contained 30 attributes, 10,000 transaction records and densities of 20% and 80%.

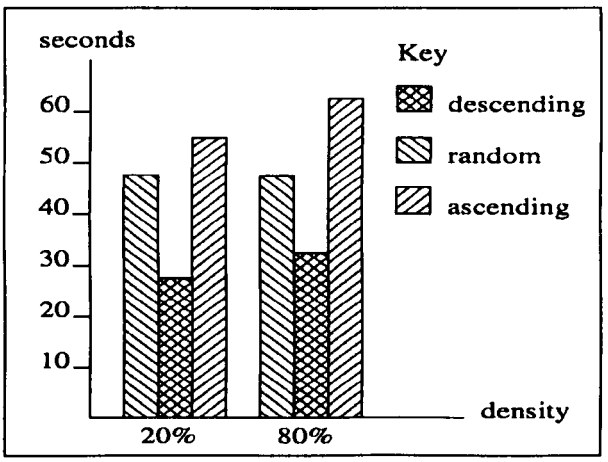


Figure 4.2: Graph illustrating random, ascending and descending ordering

As can be seen from the graph (4.2) the ordering of the dataset does, as expected, have an influence on the running time of the algorithm. Whilst the LLPS algorithm enables a greater number of attributes to be processed its limitations in terms of

number of transaction records mean it is not a viable method for processing large datasets. The LLPS algorithm computes stage 1 of Algorithm A earlier. The limitations in its performance illustrate that it is impractical to implement stage 2 of Algorithm A.

For effective implementation of both stages of Algorithm A, a method is required for storing the partial supports in a form which facilitates efficient stage 2 computation.

4.3.2 Developing The P-tree

The concept of the $P-$tree was developed to overcome the shortfalls of the BF and LLPS algorithms. Both algorithms contained good and bad points; BF could process the number of records required, but was limited to small attribute numbers and wasted much computational effort, LLPS could deal with the larger attribute numbers but not the required number of records.

To address these issues the set enumeration framework proposed by Rymon [1992] was explored. Figure 4.3 shows the sets of subsets of I, for $I = \{A, B, C, D\}$, in this form. The tree is generated in its entirety with all counts set to zero. In this structure, each subtree contains all the supersets of the root node which follow the root node in lexicographic order.

Using this tree as a storage structure for support-counts in stage 1 of Algorithm A is straightforward and computationally efficient: locating the required position on the tree for any set of attributes requires at most n steps. Furthermore advantage can be taken of the structural relationships implied by the tree to begin the computation of total supports. This is because, in locating a node on the tree, the traversal will pass through a number of nodes which are subsets of the target node. In doing so, it is inexpensive to accumulate *interim* support-counts at these nodes. A (stage 1) algorithm for this has the following form:

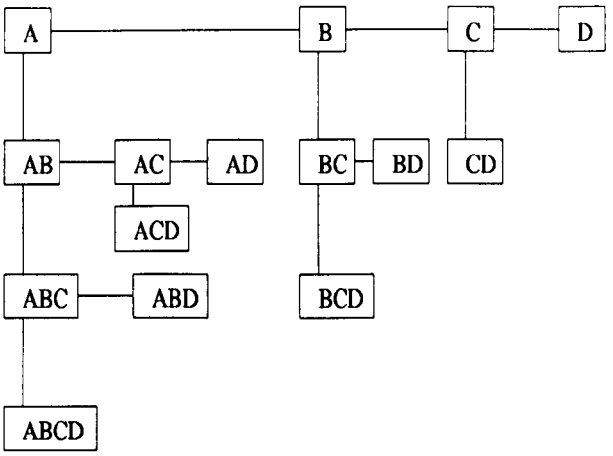


Figure 4.3: Tree storage of subsets of $\{A, B, C, D\}$

Algorithm B:

for all records j in database do

starting at root node of tree:

begin if $j \supseteq$ node then increment Q(node);

if j = node then exit

else if $j \supseteq$ node then recurse to child node

else recurse to next sibling;

end

This algorithm will, in stage 1, compute interim support-counts Q_i for each subset i of I, where Q_i is defined thus:

$$Q_i = \sum P_j \qquad (\forall j, j \supseteq i, j \textit{ follows } i \textit{ in lexicographic order}) \tag{4.1}$$

It then becomes possible to compute total support using the equation:

$$T_i = Q_i + \sum P_j \quad (\forall j, j \supset i, j \textit{ precedes } i \textit{ in lexicographic order}) \tag{4.2}$$

The following example shows how Algorithm B would populate a set enumeration tree from the example dataset in Figure 4.4 (note, the TIDs are to clarify the example). Initially the tree is generated in its entirety with all support counts set to zero, (in the example support counts are shown as the integers to the right of each node).

TID	A	B	C	D
10	1	0	1	1
20	0	1	1	0
30	1	1	1	0
40	1	0	0	1
50	1	1	1	1

Example dataset

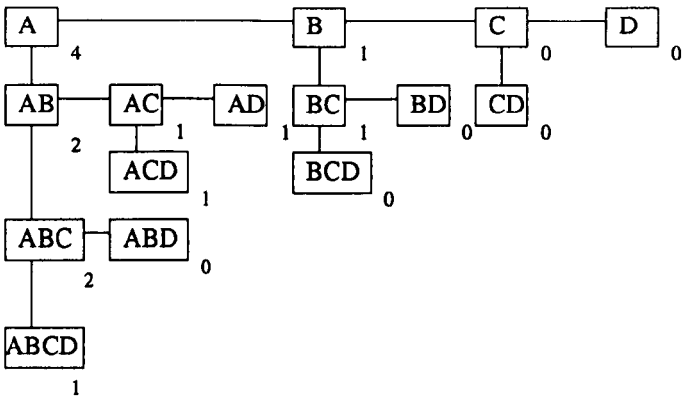


Figure 4.4: Tree storage of example dataset

- Transaction record (R) 10 is read:

  1. The initial element of the transaction is used to locate the root of the subtree to be traversed, in this case subtree root A.
  2. As R passes A the support for node A is incremented by 1.
  3. R then moves down to node AB but this is not a subset so it is not incremented and its sibling is inspected.
  4. The sibling, AC is a subset of R so this node's support is incremented by 1.

5. The child of AC is inspected this is an exact match for R so its support is incremented by 1 and the search terminates.

- The next R, 20, contains as its initial element B, therefore the root of the subtree to be traversed is at node B.

  1. The R contains B as a subset so the support for B is incremented by 1.
  2. The child of B, BC, is an exact match so its support is incremented by 1 and the search terminates.

- R 30 has its root at A.

  1. Node A is a subset of R, increment support by 1.
  2. Inspect child, AB is a subset of R increment support by 1.
  3. Inspect child ABC is an exact match of R increment support by 1 and exit.

- R 40, has its root at A.

  1. Node A is a subset of R, increment support by 1.
  2. Inspect child, AB is not a subset.
  3. Inspect sibling (AC) not a subset.
  4. Inspect sibling AD is an exact match of R increment support by 1 and exit.

- Finally, R 50 has its root at A increment and inspect child.

  1. AB is a subset of R increment and inspect child.
  2. ABC is a subset of R increment and inspect child.
  3. $ABCD$ is an exact match for R, increment support by 1 and exit.

As can be seen from the example (figure 4.4) some of the nodes are never incremented and storage of the complete tree of subsets of I, of course, has a requirement of order 2^n, (where n is the number of attributes in I) which will in general be infeasible. This can be avoided, however, by observing that for large n it is likely that most of the subsets i will be unrepresented in the database and will therefore not contribute to the partial-count summation.

An alternative version of Algorithm B that exploits the fact that many subsets are unrepresented builds the tree dynamically as the transaction records are processed storing partial totals only for transaction records which appear in the database. This tree structure could, if allowed, degenerate into another form of the LLPS algorithm, however the tree structure is maintained by inserting "dummy" nodes when certain criteria are found. Nodes are created only when a new subset i is encountered in the database as with LLPS, or when two siblings i and j share a leading subset which is not already represented. The latter provision maintains the structure of the tree as it grows. The summation of the interim supports that are present in the tree allows the total supports to be obtained using the formulae numbered 4.2.

Building the tree dynamically enables a storage requirement of order m, the number of records, rather than 2^n. This will be reduced further, perhaps substantially, if the database contains a high incidence of duplicates. In the following sections the algorithm for building the partial support tree is described. Completing the support-count computation using the tree is described in Chapter 5.

4.4 Generating the P-tree

A complete set enumeration tree, as illustrated in 4.3, would require 2^n nodes for large n (n being the number of attributes) this would be impractical. The P-tree, because of its dynamic construction, *only* has nodes added in the following cases: when a new subset i is encountered in the database, or when two siblings i and j

share a leading subset which is not already represented. In this section the algorithm to produce a partial support tree with dummy nodes is described. The nodes in the tree represent either unique transaction records within the transaction dataset, or dummy nodes inserted to prevent the tree from degenerating into a "linked list". The nodes are arranged so that parent nodes are subsets of their descendent nodes. Sibling nodes are ordered lexicographically, consequently the conception of "elder" and "younger" siblings can be used. Examples are given later in the chapter. The algorithm operates by passing through the transaction dataset in an iterative manner reading each transaction record. On each iteration if the transaction record under consideration is already represented by a node in the tree the support associated with this node is incremented by one. Otherwise a new node is created for the record and inserted in the appropriate location in the tree. This requires a search through the tree as a consequence of which the new node is inserted as a parent, child, elder sibling or younger sibling of some existing node. On route supports associated with existing nodes may be incremented. Where necessary an additional (dummy) node may also be created to preserve the desired tree structure. Alternatively, when inserting a new node it may be necessary to break certain portions of the tree and reconnect them so as to conform to the defined structure of the tree. To achieve this the tree is traversed in an iterative manner. On each iteration the nature of the search is defined by a set of five basic rules. These rules are described in detail below (where R represents the bit pattern of the transaction record currently under consideration, and B a bit pattern attached to an existing tree node under consideration). The search commences at the root of the tree and may progress down either the sibling branch or child branch of this node. So that knowledge of the nature of the current branch is maintained a flag is set with the values 0 (root), 1 (child) or 2 (sibling). Note; the $<$ and $>$ operators are used to define lexicographic not numeric ordering.

Rule 1($R = B$, an identical node is found) : Simply increment the support asso-

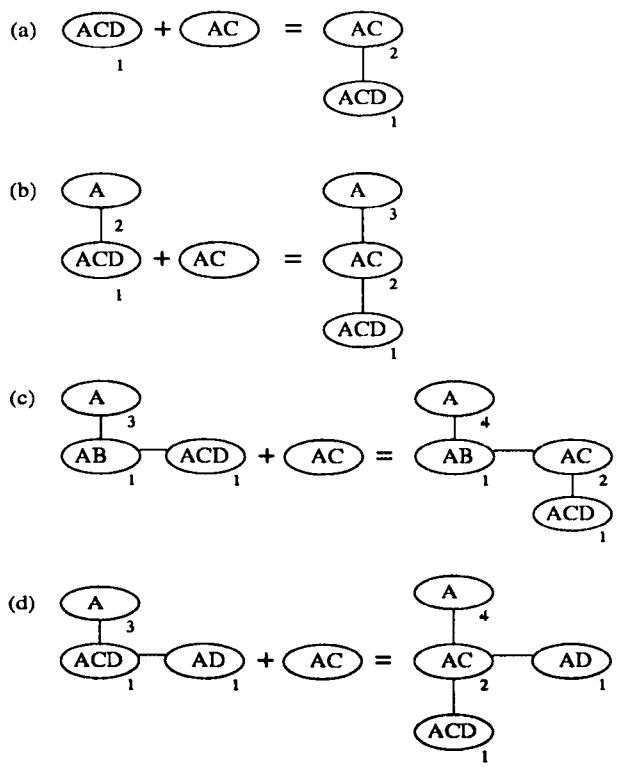


Figure 4.5: Application of Rule 2

ciated with B and return.

Rule 2($R < B$ and $R \subset B$, new transaction record is a parent of current node)

:

1. Create a new node for R and place the node associated with B on the new node's child branch.
2. Place the new node either as a new root node, or add it to the child or sibling branch of the previously investigated node (as indicated by the flag). This is illustrated in Figure 4.5(a), (b) and (c). In Figure 4.5 (a) the new node is added as a new root node, in (b) as a child of the previously investigated node and in (c) as a elder sibling.
3. If necessary move one or more of the younger siblings of the previously existent node up to become younger siblings of the newly created node.

This is illustrated in Figure 4.5(d) where the node AD, which was an elder sibling of ACD is moved up to become an elder sibling of the newly created node AC.

Rule 3($R < B$ and $R \not\subset B$, new transaction record is an elder sibling of existing node) :

1. If R and B have a leading sub string S and S is not equal to the code associated with the parent node of the node representing B then:
   - Create a dummy node for S with support summation of that associated with the node for B and any of B's siblings.
   - Place the new dummy node either as a new root node, or add it to the child or sibling branch of the previously investigated node (as indicated by the flag). A number of examples illustrating this are given in Figure 4.6. In Figure 4.6(a) a dummy node is created (B) as a new root node, in 4.6(b) a dummy node is created (BC) as child node of node B, and in Figure 4.6(c) a dummy node is created (BC) as an elder sibling node of AB.
   - Then create a new node for R and place this so that it is a child of the newly created dummy node.
   - Finally place the previously existent node for B as a younger sibling of the node for R.
2. Else create a new node for R and place the node associated with B on the new node's sibling branch. The new node is then placed either as a new root node, or is added to the child or sibling branch of the previously investigated node (as indicated by the flag). Examples are given in Figure 4.6 (d), (e) and (f).

Rule 4($R > B$ and $R \supset B$, new transaction record child of current node) : Increment the support for the current node (B) by one and:

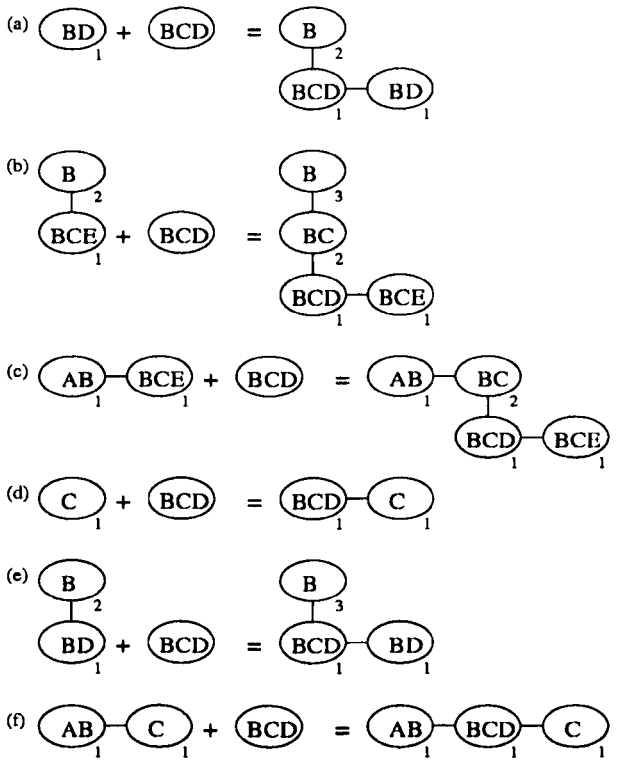


Figure 4.6: Application of Rule 3

1. If node associated with B has no child node create a new node for R and add this to the existing node's child branch.
2. Otherwise proceed down child branch (with flag set appropriately) and apply the rules to the next node encountered.

Rule 5($R > B$ and $R \not\supset B$, new transaction record is a younger sibling of current node)

:

1. If node associated with B has no sibling node, and:
   - If current node (node for B) does not have a parent node and R and B have a leading sub string S then:
     - Create a dummy node for S with support equivalent to that associated with the node for B.

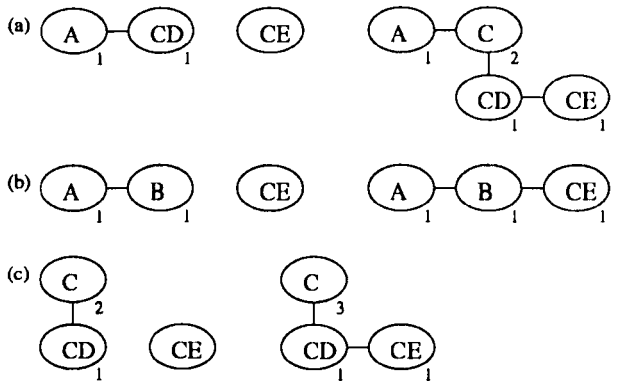


Figure 4.7: Application of Rule 5

- Place the new dummy node either as a new root node, or add it to the child or sibling branch of the previously investigated node (as indicated by the appropriately set flag). This is illustrated in Figure 4.7(a) where the dummy node C is created.
- Create a new node for R and place this so that it is a younger sibling of B.
- Place B as the child of the dummy node S.

• Otherwise create a new node for R and add this to the existing node's sibling branch (Figure 4.7(b) and (c)).

2. Otherwise proceed down sibling branch and (with flag set appropriately) and apply the rules to the next node encountered.

Thus given a bit Pattern R and a bit pattern B associated with an existing node there are five broad possible outcomes; each with a number of variations according to the nature of the tree so far, and the relationship between R and B.

4.4.1 Test Data

The P-tree algorithm was tested on three different groups of files:

1. Test Group A: Files generated with the synthetic data generator, as used for generating the test files for the previous algorithms.

2. Test Group B: Files generated by the synthetic data generator available from the IBM Quest Data Mining Project [Quest 2000].

3. Test Group C: Files Generated from the Fleet Car database and the Print Stock database at the Royal & SunAlliance facilities management operation

Test Group A Description

The benchmark number of randomly generated transaction records within this test group was 100,000. Some of the test files however, contained 200,000 randomly generated transactions and others 100,000 transactions containing 50,000 duplicates, i.e. each record was duplicated once. The number of attributes was between 200 and 1000 increasing in increments of 200. Transaction record density was set as before at 20% and 80%.

Test Group A Results

Table 4.8 provides a key for table 4.9.

A	Number of Attributes
P	Record density %
N	Total nodes in tree
D	Dummy nodes in tree
M	Memory required (bytes)
I	Node increments (rounded)
C	Comparisons (rounded)
T	Execution time (secs.)

Table 4.8: Key for tables 4.9, 4.10 and 4.11

A	P	N	D	M	I	C	T
200	20	137849	37849	$10^7 \times 0.551$	$10^6 \times 0.398$	$10^7 \times 0.926$	33
200	80	170715	70715	$10^7 \times 0.682$	$10^7 \times 0.109$	$10^7 \times 0.622$	19
400	20	137691	37691	$10^7 \times 0.881$	$10^6 \times 0.398$	$10^7 \times 0.928$	43
400	80	170665	70665	$10^8 \times 0.109$	$10^7 \times 0.109$	$10^7 \times 0.622$	27
600	20	137568	37568	$10^8 \times 0.121$	$10^6 \times 0.410$	$10^7 \times 0.936$	53
600	80	170789	70789	$10^8 \times 0.150$	$10^7 \times 0.109$	$10^7 \times 0.622$	36
800	20	137732	37732	$10^8 \times 0.154$	$10^6 \times 0.399$	$10^7 \times 0.931$	60
800	80	170587	70587	$10^8 \times 0.191$	$10^7 \times 0.109$	$10^7 \times 0.621$	46
1000	20	137693	37693	$10^8 \times 0.192$	$10^6 \times 0.398$	$10^7 \times 0.931$	71
1000	80	170826	70826	$10^8 \times 0.239$	$10^7 \times 0.109$	$10^7 \times 0.621$	55

Table 4.9: Results from algorithm P-tree (Dataset x)

The three test datasets are referred to as Datasets x, y and z to enable comparisons to be made. From the test results provided in table 4.9 and the graphs, figures 4.8 and 4.9, it can be seen that the P-tree can easily process datasets containing 1000 attributes and 100,000 transaction records. The increase in attribute numbers show that both the execution time and the memory requirement scale in a sub-linear manner. It can be seen that the less dense datasets require more processing time because of the increased number of comparisons made during tree generation. Each dataset contains no duplication: this can be seen because the total number of nodes minus the number of dummy nodes equals the number of transaction records in the original dataset. The greater number of dummy nodes in the more dense data can be accounted for by examining the number of combinations within each transaction record of the dataset. For example, the dataset containing 1000 attributes and 80% density contains 800! combinations whereas the same number of attributes at 20% density (only) contains 200! combinations. It is therefore more likely that in the tree constructed from the 20% density data that two sets will share a leading subset that is represented and a dummy node will not be required. This fact is also confirmed by the number of comparisons made. The total number of comparisons is calculated from the number of "before" tests, i.e. the current transaction

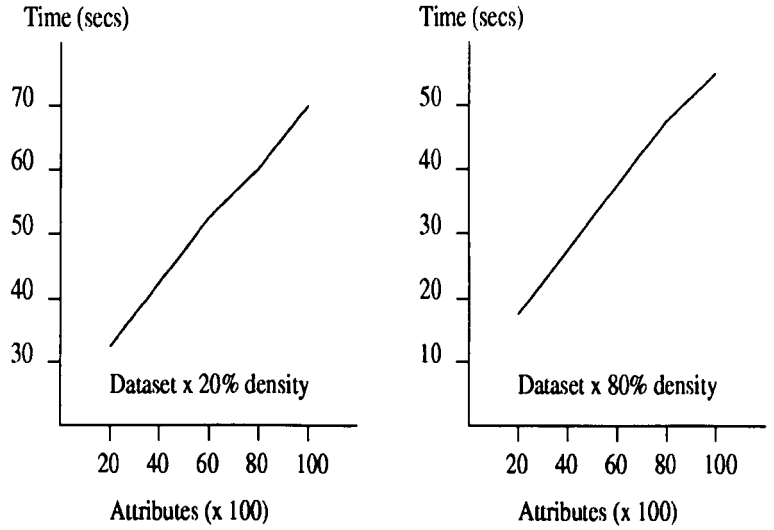


Figure 4.8: Increase in attributes vs time (Dataset x)

record is before (lexicographically) the node under inspection; "subset" tests, i.e. the current transaction record is a subset of the node under inspection; "equals" tests, i.e. the current transaction record is equal to the node under inspection; and "substring" tests, i.e. the current transaction record contains a substring node under inspection. The difference between "subset" and "substring" is illustrated by the following example. ABC and ABD are both subsets of $ABCDE$ however, they contain different substrings and therefore need to be placed in different parts of the tree. The greater number of comparisons shows that the nodes in the tree and transactions being read from the dataset share a greater number of similarities in their elements and therefore require more comparisons to locate their correct position on the tree.

To ensure the scalability in terms of transaction records each of the datasets listed in table 4.9 that contained 20% density were produced containing 200,000 transaction records. Table 4.10 displays the results of these tests and figure 4.10 illustrates the execution times and memory requirements graphically.

As can be seen from the table 4.10 and the graphs in figure 4.10, both the execution

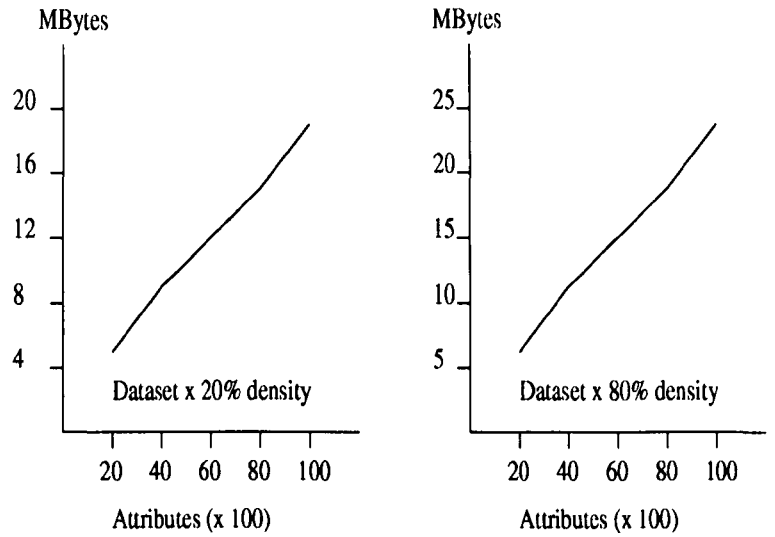


Figure 4.9: Increase in attributes vs storage requirement (Dataset x)

A	P	N	D	M	I	C	T
200	20	275892	75892	$10^8 \times 0.110$	$10^6 \times 0.859$	$10^8 \times 0.199$	73
400	20	275992	75992	$10^8 \times 0.176$	$10^6 \times 0.863$	$10^8 \times 0.199$	92
600	20	277013	76013	$10^8 \times 0.242$	$10^6 \times 0.914$	$10^8 \times 0.200$	122
800	20	275910	75910	$10^8 \times 0.309$	$10^6 \times 0.865$	$10^8 \times 0.200$	139
1000	20	275712	77712	$10^8 \times 0.385$	$10^6 \times 0.863$	$10^8 \times 0.200$	162

Table 4.10: Results from 200,000 transactions (Dataset y)

time and the memory requirements increase in an almost linear manner.

To illustrate the effect of duplication on the algorithm each of the datasets listed in table 4.9 that contained 20% density were produced containing 100,000 transaction records, 50,000 of which were duplicates. Table 4.11 displays the results of these tests and figure 4.11 illustrates the execution times and memory requirements graphically.

As can be seen from the results presented the memory requirements are what would be expected, the duplicated data set requires roughly half the memory space. The time differences between the two sets of data shows very little difference. This is

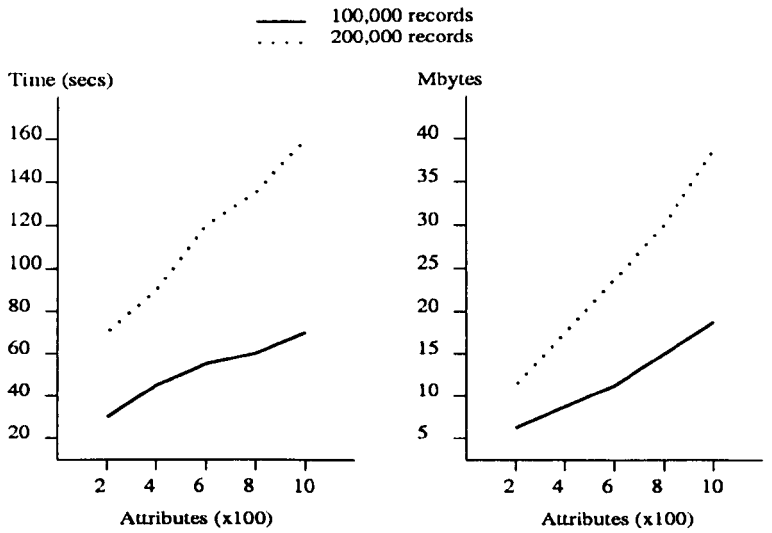


Figure 4.10: Comparison at 20% density of Datasets x and y

A	P	N	D	M	I	C	T
200	20	68910	18910	$10^7 \times 0.275$	$10^6 \times 0.418$	$10^7 \times 0.851$	34
400	20	68855	18855	$10^7 \times 0.440$	$10^6 \times 0.419$	$10^7 \times 0.851$	42
600	20	68711	18711	$10^7 \times 0.604$	$10^6 \times 0.424$	$10^7 \times 0.860$	51
800	20	68791	18791	$10^7 \times 0.770$	$10^6 \times 0.418$	$10^7 \times 0.855$	61
1000	20	68824	18824	$10^7 \times 0.963$	$10^6 \times 0.418$	$10^7 \times 0.855$	67

Table 4.11: Results from 100,000 transactions (50,000duplicates) (Dataset z)

because of the number of comparisons carried out during the construction of the trees, each tree is placing 100,000 transaction records, the fact that a record is a duplicate of a previous one does not reduce the number of comparisons that need to be made to place the record in its correct location.

Test Group B Description

The test files making up Test Group B were generated using the synthetic data generator downloaded from the IBM Quest Data Mining Project web site. The datasets generated in Test Group A were randomly generated with the only guidance being

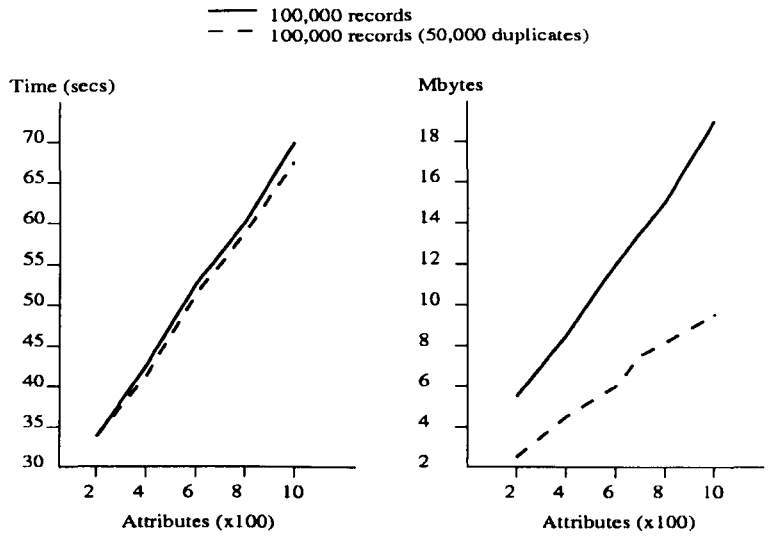


Figure 4.11: Comparison at 20% density of Datasets x and z

the average density of each transaction record. The following description of the Quest data generation method is from [Agrawal and Srikant 1994]: The Quest data is aimed at mimicking transactions taking place in a retail environment. To this end it models the fact that people tend to buy sets of items together. Transaction sizes are typically clustered around a mean and a few transactions have many items. Typical sizes of large itemsets are also clustered around a mean with a few large itemsets having a large number of items. The following two tables 4.12 and 4.13 provide a key to the parameters and the parameter settings used in the dataset generation. (Note, because of the different dataset properties a different labelling system is used.)

For each dataset the number of transaction records (D) was set to 100,000, the number of attributes (N) was set to 1000 and the number of maximal potentially large itemsets (L) was set to 2000. These parameters mirror those used in [Agrawal and Srikant 1994].

D	Number of transactions
T	Average size of transactions
I	Average size of the maximal potentially large itemsets
L	Number of maximal potentially large itemsets
N	Number of items
DE	Data Density

Table 4.12: Parameters used in dataset generation

Dataset	T	I	DE
A	5	2	0.5
B	10	2	1.0
C	10	4	1.0
D	20	2	2.0
E	20	4	2.0
F	20	6	2.0

Table 4.13: Parameter settings

Test Group B Results

Table 4.8 provides a key for table 4.14. The results illustrated in table 4.14 show that the overall performance of the P-tree algorithm is fairly consistent regardless of the numbers of items within each transaction record. The dominant factor in the running time of the algorithm, (as with Test Group A) is the number of comparisons that need to be made for the tree to be correctly constructed. As can be seen it is the smallest test group (A) in terms of numbers of items per transaction record, that takes the longest time to execute. Datasets A', B' and D' were created with 200,000 records and the results are shown in table 4.15 Testing the datasets with 200,000 transaction records reveals the linear scaling features of the algorithm.

A subset of datasets within Test Group B was generated to illustrate the algorithm's

Dataset	*N*	*D*	*M*	*I*	*C*	*T*
A	113673	13673	$10^8 \times 0.159$	$10^6 \times 0.420$	$10^8 \times 0.645$	144
B	142507	42507	$10^8 \times 0.199$	$10^6 \times 0.613$	$10^8 \times 0.511$	135
C	135831	35831	$10^8 \times 0.190$	$10^6 \times 0.564$	$10^8 \times 0.458$	121
D	151749	51749	$10^8 \times 0.212$	$10^7 \times 0.108$	$10^8 \times 0.345$	120
E	151885	51885	$10^8 \times 0.212$	$10^6 \times 0.871$	$10^8 \times 0.351$	121
F	152973	52973	$10^8 \times 0.214$	$10^6 \times 0.965$	$10^8 \times 0.358$	121

Table 4.14: Results from Test Group B (100,000 records)

Dataset	*N*	*D*	*M*	*I*	*C*	*T*
A'	220805	20805	$10^8 \times 0.319$	$10^7 \times 0.132$	$10^9 \times 0.135$	331
B'	285084	85084	$10^8 \times 0.399$	$10^7 \times 0.205$	$10^9 \times 0.107$	318
D'	303357	103357	$10^8 \times 0.424$	$10^7 \times 0.379$	$10^8 \times 0.750$	298

Table 4.15: Results from Test Group B (200,000 records)

behaviour with clustered data. Test Group A contained only random data within any transaction record and the initial datasets of Group B i.e. those detailed in tables 4.14 and 4.15 contain clustering within any one transaction but not inter-transaction clustering. To generate clustering "around" a particular transaction the synthetic data generator was adapted to produce a number of closely related transaction records. For example, a file containing 25,000 transaction records generated by the Quest Generator, would be read in to the program, each record would then have three new records created which were similar to the original but not exact matches. The results for Test Group B, clustered datasets, are shown in table 4.16.

The clustered datasets produced similar memory requirements and node numbers. The execution time was slightly faster for each of the clustered datasets because the number of comparisons made and the number of node increments were reduced.

Dataset	*N*	*D*	*M*	*I*	*C*	*T*
A	109389	9389	$10^8 \times 0.153$	$10^6 \times 0.396$	$10^8 \times 0.617$	131
B	148910	48910	$10^8 \times 0.208$	$10^6 \times 0.352$	$10^8 \times 0.443$	118
C	134192	34192	$10^8 \times 0.187$	$10^6 \times 0.348$	$10^8 \times 0.424$	112
D	153376	53376	$10^8 \times 0.214$	$10^6 \times 0.489$	$10^8 \times 0.295$	105
E	151793	51793	$10^8 \times 0.212$	$10^6 \times 0.443$	$10^8 \times 0.306$	106
F	148988	48988	$10^8 \times 0.208$	$10^6 \times 0.453$	$10^8 \times 0.312$	108

Table 4.16: Results from Test Group B (100,000 clustered records)

Test Group C Description

The final test data sets were obtained from the Royal & SunAlliance facilities management operation. These were the car fleet dataset and the print stock dataset. The fleet data was stored on a personal computer in Microsoft Excel format. The print data operating software was the Shuttleworth "Prima" Stock Control System version 17. This ran on an SCO UNIX system. Both systems allowed the creation of *flat files* i.e. files containing asci code only. Once these had been obtained they were pre-processed to transform each record into a binary vector.

A full listing of the attributes and how they were preprocessed is detailed in Appendix A. After processing the Fleet Dataset contained 9000 records and 195 attributes each record contained 17 binary 1's, the Print Dataset contained 6800 records and 459 attributes each record contained 24 binary 1's.

Test Group C Results

As would be expected after the performance characteristics illustrated by the previous test datasets the fairly small facilities management datasets were processed without any problems. Table 4.17 details the performance of the algorithm on these datasets. Table 4.8 provides a key for table 4.17.

Dataset	*N*	*D*	*M*	*I*	*C*	*T*
FM Fleet	11545	2545	$10^6 \times 0.461$	$10^5 \times 0.514$	$10^6 \times 0.690$	3
FM Print	10799	3999	$10^6 \times 0.777$	$10^5 \times 0.199$	$10^7 \times 0.274$	4

Table 4.17: Results fron Test Group C (FM Dataset)

4.5 Evaluation

In the case of a very small dataset the brute force type algorithm would provide an efficient and rapid means of calculating support. However, as illustrated by the two examples, the brute force method has severe limitations. The Linked List Partial Support algorithm overcomes, to a degree, some of the limitations in terms of attribute numbers encountered by BF algorithms, however only partial supports are gained from the LLPS algorithm. When executing the Fleet Dataset the LLPS algorithm took in excess of two hours to terminate, (compared with 3 seconds for the P-tree). LLPS is also limited by the small number of transaction records it can process.

The P-tree storage structure has shown itself to be an effective method of resructuring the original dataset in the following ways:

- The increase in terms of record numbers in the test datasets results in approximately linear scaling in terms of time and space.
- Execution time is almost independent of attribute numbers.
- Data density affects the size and build time for the tree. However, it is *decreasing* the density that leads to an increase in execution time.
- The presence of duplicates transaction records reduces memory requirement.
- In building the P-tree some of the work required for the computation of total support is completed.

Whilst it is obvious that the number of nodes in the P-tree is greater than the number of transaction records in the original dataset in all the test data, this is not seen as a problem. When summing total supports the advantage of the tree structure will be that it allows the elimination of large areas of the search space.

4.6 Summary

The first part of this chapter dealt with the construction and evaluation of three algorithms, Brute Force 1, Brute Force 2 and Linked List Partial Support. The Brute Force algorithms were expected to be of limited pracrtical use for mining association rules, however the implementation of them provided a start point for further development. The LLPS algorithm differed from the BF algorithms both in terms of its goals and its data processing ability. Whilst the BF algorithms provided a complete solution i.e. their output was a list of supported sets, the LLPS algorithm only made a partial contribution to locating these.

The second part of the chapter detailed the construction, implementation and testing of the P-tree. Using the foundation of the set enumeration tree and the dummy node concept the dynamic construction of the P-tree was achieved. The test data files showed the P-tree more than capable of processing moderately large datasets with various configurations of attributes, transaction record numbers and transaction record density. As with the LLPS algorithm the P-tree algorithm in its single pass of the dataset goes some way to calculating the supported itemsets. The algorithm and data structure for delivering the itemsets that meet the user defined support threshold will be detailed in the next chapter.

Chapter 5

Summation Of Partial Support Tree

5.1 Introduction

The previous chapter described the idea and implementation of the P-tree. This chapter describes how the P-tree structure can be used to produce lists of supported itemsets from which the association rules can be produced. The P-tree is exploited by a data structure called the Total Support Tree or T-tree. This structure allows maximum benefit to be derived from the partial totals already gathered in the P-tree and resident in main memory. Using the T-tree to produce total support for itemsets has the following advantages over existing "repeated pass algorithms":

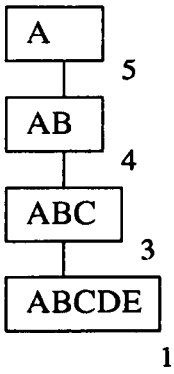
1. The T-tree is tailored to take advantage of the pre-processing achieved by the P-tree.

2. In building the T-tree only the areas of the P-tree necessary to be searched are examined: the ordering of the nodes in both trees reduces the potential search area.

The summation of the partial supports contained within the P-tree is achieved by a depth first traversal of the structure. The T-tree however, is built in a breadth first manner as the total supports are retrieved from the P-tree.

5.2 Summation of Partial Supports

As observed in Chapter 4, the P-tree is essentially a restructuring of the database in to a more useful form while also beginning the process of computing support counts for its attribute sets. In principle it is possible to apply any existing algorithm for computing association rules to this revised structure. The T-tree solution is based on the Apriori breadth first construction method. The advantage offered is that because of the partial summation already carried out fewer subsets need to be considered as each node of the P-tree is examined. The following example (Figure 5.1) illustrates this property.



TID	A	B	C	D	E
10	1	0	0	0	0
20	1	1	0	0	0
30	1	1	1	0	0
40	1	1	1	0	0
50	1	1	1	1	1

Figure 5.1: P-tree and dataset

In Figure (5.1) the P-tree on the left is constructed from the dataset on the right (the dataset TID numbers are for clarity). When support is being calculated using the P-tree the number of subsets that need investigating is less than when reading the dataset. If the support for level 2 subsets, i.e. pairs was being determined traversing the P-tree would proceed in the following manner:
N = number of subsets examined.

1. Examine root, node contains only one attribute proceed to child, $N = 0$
2. AB contains only one pair, $N = N + 1$

3. Examine ABC; because the support for AB has been determined by the examination of the previous node only the support for AC and BC needs finding, $N = N + 2$

4. The final node only differs from the previous node by the items D and E so support needs to found for sets AD, BD, CD, AE, BE, CE and DE a total of 7 pairs $N = N + 7$

Total: 10 pairs examined.
Contrast this with the numbers of subsets calculated from the dataset:

1. TID 10, one attribute, therefore no pairs $N = 0$

2. TID 20, one pair $N = N + 1$

3. TID 30 and 40, three attributes 3 pairs each, $N = N + 6$

4. TID 50 requires all pairs to be examined $N = N + 10$

Total: 17 pairs examined.
As can be seen from the example in the best case only r - 1 subsets of a record of r attributes need to be considered, rather than $r(r-1)/2$ required by the original Apriori.

Storing The Total Supports

The T-tree stores the total support counts for sets identified as interesting. An example of a complete T-tree is illustrated in Figure 5.3. This has been generated from the P-tree shown in Figure 5.2.

From the figures it can be observed that the T-tree is essentially the obverse of the P-tree, i.e. each subtree contains only predecessors of its parent. The singleton

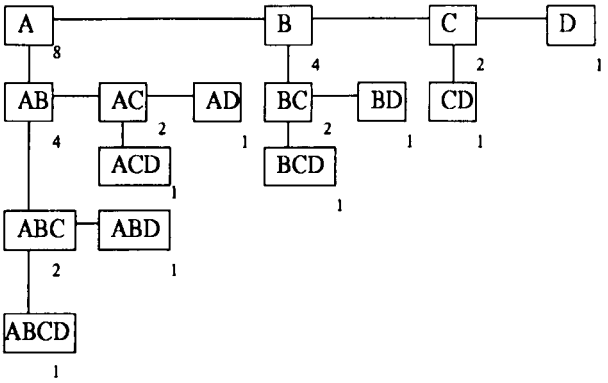


Figure 5.2: Example P-tree

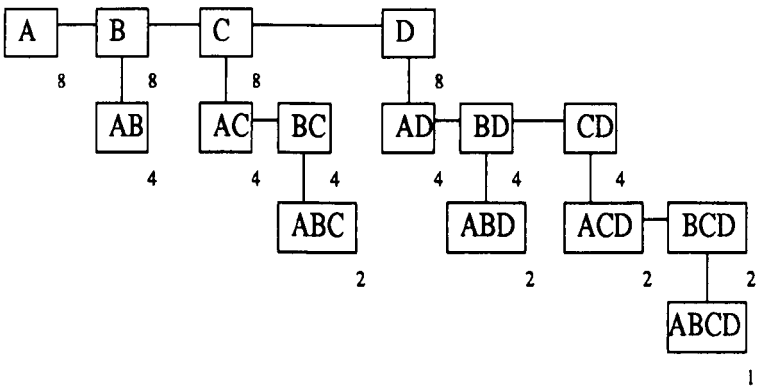


Figure 5.3: Complete T-tree founded on P-tree presented in 5.2

nodes are ranged along the top from which the pairs, triples descend as appropriate. Whereas in the P-tree the (say) "B-branch" contains only those nodes whose identifier commences with the digit B, in the T-tree the "B-branch" contains only those nodes whose identifier ends in B. The significance of this arrangement is that, when counting total support deriving from any particular set, the subsets which need to be considered are grouped together in the tree. This advantage can best be illustrated by considering the T-tree generation process.

5.3 T-tree Generation

At its most abstract the process of generating the T-tree, using the Apriori-type methodology, can be viewed as an iterative algorithm comprising of the following steps:

1. Build level k in the T-tree.
2. Traverse the P-tree and apply appropriate partial supports associated with individual P-tree nodes to the level k nodes established in (1).
3. Remove any level k T-tree nodes that fall below the support threshold.
4. Repeat steps (1),(2) and (3); until a level k is reached where no nodes exceed the support threshold.

Consider the dataset and its derived P-tree as shown in Figure 5.4.

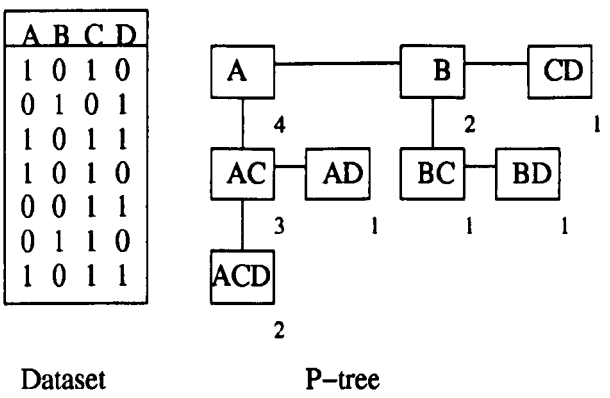


Figure 5.4: dataset and P-tree

Initially, the first level to be established in the T-tree will comprise of the singleton nodes displayed along the top sibling branch of the P-tree (compare the top lines of Figures 5.4 and 5.5). Note the P-tree node containing CD is broken down into its subsets to enable support to be derived for the attributes C and D. The support is

then applied as follows:

$$\forall P \in Ptree$$
$$\forall T \in Ttree \; where \; (level(T) \equiv 1)$$
$$if \; (T \subset P) \; \& \; (T \not\subset P_{parent}) \; \; T_{sup} = T_{sup} + P_{sup}$$

In the above the *level* function returns the level in the T-tree at which its argument is located; for example $level(ABC)$ will return 3 (note that with respect to the T-tree the level is equivalent to the number of digits present in the node identifier). The reason for ensuring that a particular T-tree node T is a subset of the P-tree node P, but not of the parent of the P-tree node is that individual supports are not calculated twice. For example if:

$$P = \{A, B, C, D\} \; with \; support \; 2$$
$$P_{parent} = \{A, B\} \; with \; support \; 4$$

the support of 4 associated with $\{A, B\}$ already includes the support contribution associated with $\{A, B, C, D\}$. Note, however, that the P-tree in this case does not include the node ABC. Thus, the above algorithm (assuming for the moment inspection of the top level only) will add a support of 2 to the supports calculated so far for nodes C and D but not to those associated with nodes A and B because these will have been considered earlier during the tree traversal. An alternative, and more succinct formal definition of the algorithm is:

$$\forall P \in Ptree$$
$$P' = P \setminus P_{parent}$$
$$\forall T \in Ttree \; where \; (level(T) \equiv 1)$$
$$if \; (T \subset P') \; \; T_{sup} = T_{sup} + P'_{sup}$$

In the above, the *search node* P' is made up of those digits that are in P but not in P_{parent}. In some cases P_{parent} may equate to *null* (i.e. no parent) in which case

P' will be equal to P. In this manner the supports for the top-level nodes in the T-tree can be determined. This first level is then pruned by removing any nodes that do not display the required level of support. Thus, considering the example P-tree presented in Figure 5.2 and assuming a minimum required support level of 2, Figure 5.5(a) represents the initial T-tree candidate set.

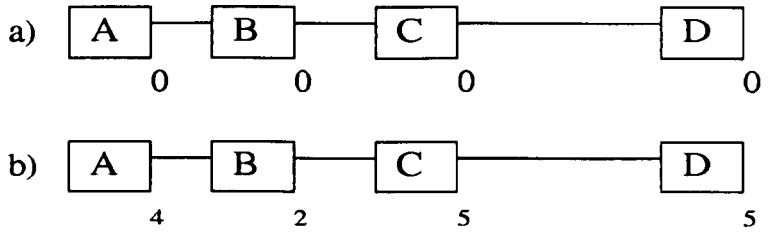


Figure 5.5: Level 1 T-tree

Applying the above algorithm to the P-tree and this candidate T-tree set will give a revised T-tree with total supports given in Figure 5.5(b). This will then be pruned; however in this example, as all the total supports exceed the minimum support the tree will remain unaltered after pruning. From this first T-tree level (the top level) the second level candidates are now generated. The algorithm is as follows:

$$\forall T \in Ttree \; where \; (level(T) = nextLevel - 1)$$
$$\forall T' \in Ttree \; where \; (level(T') = 1)$$
$$if \; (T' < T)$$
$$add \; (T' \cup T) \; as \; child \; of \; T$$

Note that the $<$ diadic infix operator used above should be interpreted as a *lexicographic before*; thus $\{B, C, E\} < \{C, D\}$ would return *true*. As a result of the above, and given the T-tree so far (Figure 5.5(b)) the level 2 candidate nodes ("doubles") of the T-tree shown in Figure 5.6 are generated.

To determine the total supports for these level 2 T-tree nodes the P-tree is traversed for a second time and, for each P-tree node, add the interim support associated with the P-tree node to the appropriate (level 2) T-tree node(s). To achieve this

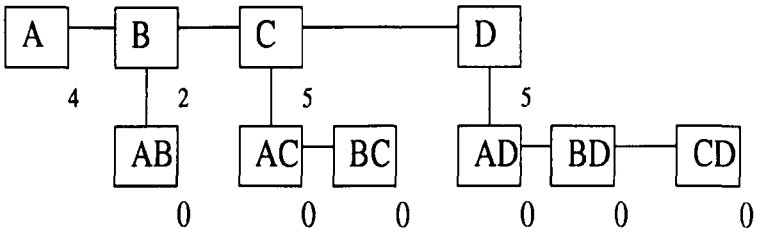


Figure 5.6: Level 2 T-tree

efficiently it is desirable to use some technique to limit the search space. As noted above, a feature of the T-tree is that the identifier for each node in a subtree emanating from a top level node contains the same end digit. This is important because this allows the identification of those particular subtrees in the T-tree that need to be considered when determining total supports for lower level T-tree nodes. The nodes of interest are the complement of any discovered P-tree node in its parent node. For example if $P = \{A, B, C, D\}$ and $P_{parent} = \{A, B\}$ then only the branches emanating from the C and D top level T-tree nodes ($\{A, B, C, D\} \setminus \{A, B\} = \{C, D\}$ need to be considered. This provides an effective implementation of the reduced subset-count derived from the partial computation already completed. The derivation of the supports for lower level T-tree nodes, taking into consideration the above technique, thus requires some addition/modification to the basic algorithm as presented so far. The revised algorithm can be considered to comprise two parts:

Part 1 Process the top level of the T-tree and identify those nodes which merit consideration (by virtue of being a subset of the search node P'). If, at this stage, the required level is equal to 1 (i.e. the top level) apply the appropriate additional support levels as already described, and stop; otherwise proceed to Part 2.

Part 2 Proceed down the child branches of each of the nodes identified in Part 1 until the required level is reached and then apply the appropriate support updates.

More formally this can be expressed as follows:

$$\forall P \in Ptree \ where \ (numDigits(P) \geq requiredLevel)$$

$$P' = P \setminus P_{parent}$$

$$\forall T \in Ttree \ where \ (level(T) \equiv 1)$$

$$if \ (T \subset P')$$

$$if \ (requiredLevel \equiv 1) \ T_{sup} = T_{sup} + P_{sup}$$

else proceed down child branch of T *until required level is reached and then apply appropriate support updates*

Note that the *numDigits* function returns the number if digits (binary '1's) in the identifier for a P-tree node; this will equate to a T-tree level. Of course, it is desirable to avoid exhaustive searches of both the top level of the T-tree and any sub-branches which require attention. This can be achieved by making use of the lexicographic ordering imposed on the T-tree. Thus considering the first stage in the above: given some search node P' begin at the start of the T-tree (node A in the examples given in Figures 5.5, 5.6 and 5.7) and test whether A is *before*, *equal to*, *subset of* or *after* P' and then continue as follows:

before Simply proceed to the next T-tree node.

equal to If level 1 is currently under inspection update the current T-tree node and stop; otherwise proceed to Part 2. In neither case is there any need to continue processing the top level of the T-tree because for P' to be equal to a top level T-tree node it must be a singleton, in which case it can only match one T-tree node.

subset of Proceed in a similar manner as in the equals case but this time continue processing the top level of the T-tree having removed the first digit of the identifier for P'.

after Proceed to next node in T-tree having first removed the first digit of the identifier for P'.

Continue in this manner until there are no digits left in the search node (P'). To identify the first digit of a node identifier a function $firstDigit$ is defined which returns the first digit of its argument (which must be a node identifier). Thus:

$$firstDigit\,(\{B, C, D\})$$

will return B. To remove this first digit from P' perform a complement operation: i.e. $\{B, C, D\} \setminus firstDigit\,(\{B, C, D\}) = \{B, C, D\} \setminus \{B\} = \{C, D\}$. More formally the above can be described as follows:

loop while $P' \neq null$

  if $T_{1,j} < P'$

    $j++$

  if $T_{1,j} \equiv P'$

    if $(requiredLevel \equiv 1)$ $T_{sup} = T_{sup} + P_{sup}$

    else **Part 2**

    $P' = null$

  if $(T_{1,j} \subset P')$

    if $(requiredLevel \equiv 1)$ $T_{sup} = T_{sup} + P_{sup}$

    else **Part 2**

    $P' = P' \setminus firstDigit(P') \cdot j++$

  if $T_{1,j} > P'$

    Stop

where the double subscript of T is interpreted as the level number and location along that level respectively. Thus the T-tree node $T_{1,3}$ is in the top level (level 1) and is the third node along in this level, i.e. node C in Figure 5.7. The ++

postfix unary operator should be interpreted as "increment by 1". Suppose the P-tree node $\{A, B, C\}$ of Figure 5.2 is being processed. P' in this case is equal to C ($\{A, B, C\} \setminus \{A, B\} = \{C\}$). Commencing at the top level of the T-tree of Figure 5.7 with the node A. Node A is before C (P') so proceed to T-tree node B. Node B is also before C (P') so proceed to T-tree node C. T-tree node C is equal to C (P'); level 1 is of no interest, so proceed to Part 2. After returning from Part 2 stop. Putting all of the above together the Part 1 algorithm can be expressed as presented below.

$\forall P \in Ptree \ where \ (numDigits(P) \geq requiredLevel)$

$P' = P \setminus P_{parent}$

$\forall T \in Ttree \ where \ (level(T) \equiv 1)$

loop while $P' \neq null$

if $T_{1,j} < P' \ j++$

if $T_{1,j} \equiv P'$

if $(requiredLevel \equiv 1) \ T_{sup} = T_{sup} + P_{sup}$

else **Part 2**

$P' = null$

if $(T_{1,j} \subset P')$

if $(requiredLevel \equiv 1) \ T_{sup} = T_{sup} + P_{sup}$

else **Part 2**

$P' = P' \setminus firstDigit(P') \cdot j++$

if $T_{1,j} > P'Stop$

Table 1: Total Support Algorithm (Part 1)

Part 2 is only invoked when the required level is not level 1 and some level 1 T-tree node is equal to or a subset of P'. Again, to minimise the work required to search the subtree; use is made of the lexicographic ordering of the nodes. Proceeding in a similar manner to that described for identifying individual branches but using a termination search node equivalent to the last n digits of the discovered P-tree node (where n is equivalent to the current level in the branch of the T-tree under investigation). Thus if P is equivalent to ABC and level two is the current level (the "doubles" level) the termination search node (P'') will be equivalent to BC. A function *endDigits* is defined, that takes two arguments P' and n (where n is the current level) and returns an identifier comprising the last n digits of P'. For example:

$$endDigits(ABC, 2)$$

will return BC. The significance of this is that BC is the last subset of ABC at level 2 that need be considered. Each discovered T-tree branch is processed in an iterative manner commencing with the first child node of the branch, until arriving at the required level. While descending the tree, a termination node P'' is computed at each level, and determines whether the T-tree node is: *before*, *equal to* or *after* P'' (note that because both P'' and T, as defined here, have the same number of digits at any particular level, T can never be a "proper" subset of P'). As a result of this test proceed as follows:

before If T is a subset of P proceed down child branch. In every case also continue to next T-tree node.

equals If $T \equiv P''$ then $T \subset P$ proceed down child branch, but do not continue to check siblings of T.

after Stop.

More formally:

$$P'' = endDigits(P', currentLevel)$$
$$loop\ while\ T_{i,j} \neq null$$
$$if\ T_{i,j} < P''$$
$$if(T_{i,j} \subset P)\ recursively\ call\ Part\ 2\ commencing\ with\ T_{i++,1}$$
$$j++$$
$$if\ T_{i,j} \equiv P''\ recursively\ call\ Part\ 2\ commencing\ with\ T_{i++,1}$$
$$Stop$$
$$if\ T_{i,j} > P''$$
$$Stop$$

The recursive calls in each case terminate when the required level is reached. At this level step along the nodes updating the supports as appropriate. Again, use the termination search node P'' to bound the search, comparing this node to each T-tree node. There are three possibilities:

before If T is a subset of P update support. Whatever the case also continue along sibling branch.

equals If $T \equiv P''$ then $T \subset P$ update support and stop.

after Stop.

Thus, expressed formally:

$$P'' = endDigits(P', currentLevel)$$
$$loop\ while\ T_{i,j} \neq null$$
$$if\ T_{i,j} < P''$$
$$if(T_{i,j} \subset P)\ T_{sup} = T_{sup} + P_{sup}$$
$$j++$$

$if\ T_{i,j} \equiv P''$

$T_{sup} = T_{sup} + P_{sup}$

$stop$

$if\ T_{i,j} > P''$

$stop$

Combining the above with the sub-branch search algorithm produces the entire part 2 algorithm shown below.

$P'' = endDigits(P', currentLevel)$

$loop\ while\ T_{i,j} \neq null$

$if\ T_{i,j} < P''$

$if(T_{i,j} \subset P)$

$if\ currentLevel \equiv requiredLevel\ T_{sup} = T_{sup} + P_{sup}$

$else\ recursively\ call\ Part\ 2\ commencing\ with\ T_{i++,1}$

$j++$

$if\ T_{1,j} \equiv P''$

$if\ currentLevel \equiv requiredLevel\ T_{sup} = T_{sup} + P_{sup}$

$else\ recursively\ call\ Part\ 2\ commencing\ with\ T_{i++,1}$

$stop$

$if\ T_{1,j} > P''$

$stop$

Table 2: Total Support Algorithm (Part 2)

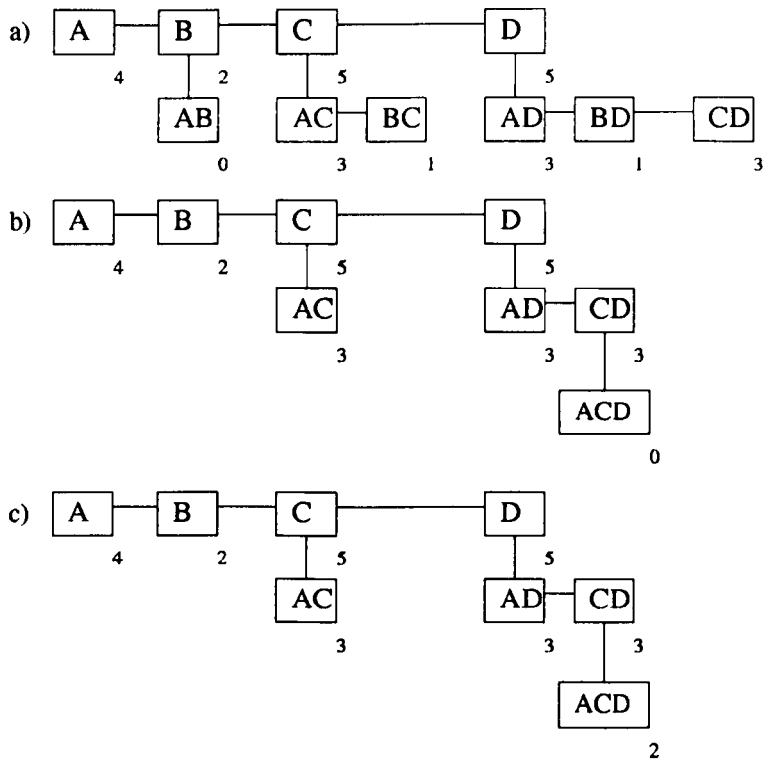


Figure 5.7: Level 3 T-tree

Returning to the example T-tree generated so far (Figure 5.6): using the above algorithm support will be added for all the pairs to give the result presented in Figure 5.7(a). These supports are all above the minimum threshold so pruning will produce no changes. Generate the next level in the T-tree (triples) as shown in Figure 5.7(b). Again adding supports derived from the P-tree nodes will produce a T-tree of the form presented in Figure 5.7(c). A level has now been reached which has no supported T-tree nodes and consequently the end of the T-tree generation process has been reached. The final tree produced will thus be of the form given in Figure 5.7(c). This tree contains all the supported sets and can be used to derive all relevant association rules contained in the data set.

Initial Performance

The candidate generation algorithm used for constructing the T-tree does not strictly adhere to the downward closure property described in Chapter 3 section 3.2. When constructing candidate sets for level three and below the algorithm does not check all the subsets of the candidate set before creating it. The algorithm takes the top level (single) T-tree nodes and appends them to the lower level node that is being created. Thus, if node BC is supported in the T-tree and A is also supported then the node ABC is created without checking for the presence of AB or AC. This means that in some cases candidate nodes will be created that will prove to be false because all their subsets are not present in the T-tree. The false candidate set that is generated will be identified as such during the P-tree traversal to establish support for this level candidate. The trade-off is therefore to either traverse the T-tree to establish the candidate as a "true" candidate or traverse the P-tree to establish that it is a "false" candidate. Using either method will return only the supported candidates at the required level. Unfortunately, the non-availability of implementations of these alternative algorithms means that direct comparison was not possible.

Initial tests for constructing the T-tree from small datasets proved that it could be constructed correctly and quickly. However, once larger datasets were introduced i.e. the level 2 candidate set contained more than 2000 candidates, the time taken to construct the T-tree became unacceptable because traversing the original P-tree structure took too long. To address this problem a new data structure was created, the P-table. This table was a dynamically created two dimensional array consisting of pointers to the P-tree. Once the level 2 candidate sets had been created on the T-tree the P-table was accessed to establish the support levels for the candidates.

5.4 Testing The T-tree

The T-tree test data was the same files which were used to construct the P-tree, as described in section 4.4.1.

1. Test Group 1: Files generated with the synthetic data generator, as used for generating the test files for the previous algorithms, i.e. BF1, BF2 and LLPS.
2. Test Group 2: Files generated by the synthetic data generator available from the IBM Quest Data Mining Project [Quest 2000].
3. Test Group 3: Files Generated from the Fleet Car database and the Print Stock database at the Royal & SunAlliance facilities management operation

During the testing of the T-tree the time taken to construct the P-tree was ignored. Two constraints were placed on the generation of supported itemsets a) that the support threshold must be less than 5% and b) whatever the support level that was being tested the result must be produced in less than 90 minutes. The support restriction was introduced to enable comparison with other association rule generation methods. The time restriction was introduced to emulate a working environment where to wait an inordinate amount of time for results would be unreasonable. Note: A full tabular breakdown of each datasets performance can be found in Appendix B.

Test Group A Results

Test Group A consisted of 3 datasets, two of which (Datasets 1 and 2) were the same datasets as used to generate the P-tree in Chapter 4 (see Test Group A description Chapter 4). The remaining dataset (3) was generated to examine the record density threshold. Each dataset contained 100,000 records, the remaining properties were:

- Dataset 1: 200 attributes with a record density of 20%.

- Dataset 2: As above, but with each of 50,000 records having one duplicate.
- Dataset 3: 1000 attributes with a record density of 6%.

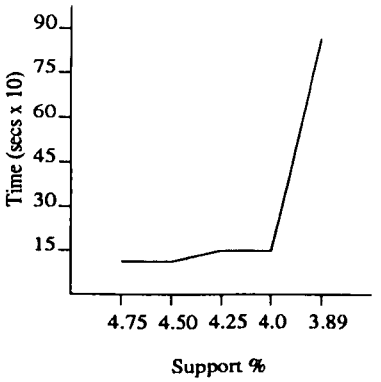


Figure 5.8: Test Group A, Dataset 1: Execution time

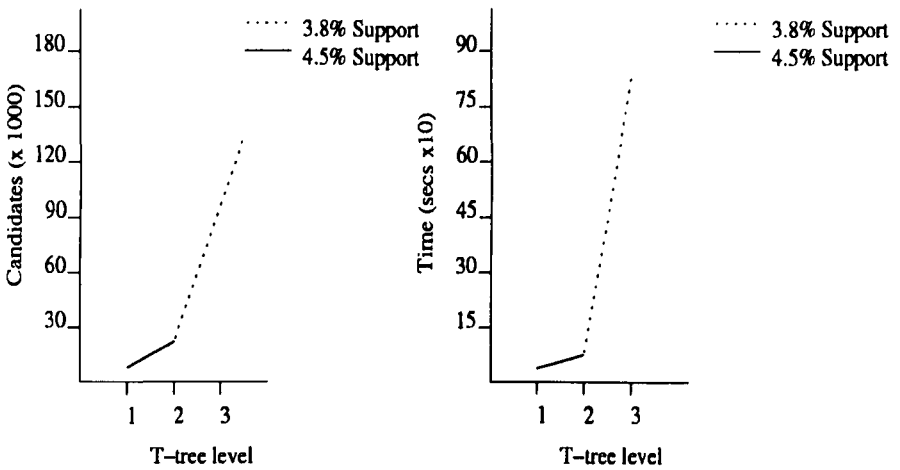


Figure 5.9: Test Group A, Dataset 1: Candidates and time at each level

(Note, in Figures 5.9 and 5.11 the 4.5% support result is superimposed over the 3.8% result up to T-tree level 2.) The graph shown in Figure 5.8 illustrates the increasing time taken to process the dataset as the level of support is decreased. Once the support threshold falls below 4% the number of supported sets at the pairs level increases from 15 to in excess of 2000[1]; as a result of this the candidates for

[1]This rapid increase in the number of candidates would be expected to occur at some point in the experiment as the support figure is lowered. This is because the problem addressed in this thesis is

level 3 increase from 900 to 145,000. The two graphs shown in Figure 5.9 illustrate the effect of the rapid increase in candidate sets. The graphs show the number of candidate sets and the time taken to process each level of the T-tree to be exactly the same for the differing support levels until level 2 is finished, at this point the 4.5% support threshold can find no level 3 candidates so the execution terminates. This can be contrasted with the 3.8% support threshold and the subsequent significant increase in time taken to process level 3 and the number of candidate sets generated.

The three graphs shown in Figures 5.10 and 5.11 illustrate the performance of the

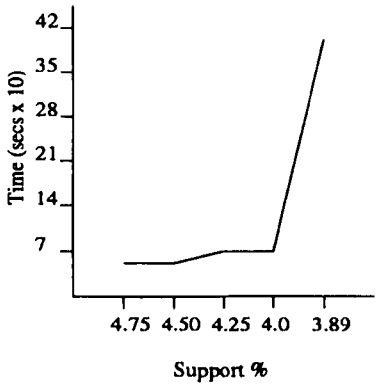


Figure 5.10: Test Group A, Dataset 2: Execution time

dataset containing 100,000 records (50,000 duplicates), 200 attributes and a density of 20%. As can be seen from the graphs the execution times are almost halved as would be expected because the P-tree is half the size of the Dataset 1 P-tree, whilst the number of candidates remains the same as for Dataset 1. The performance of Dataset 3 is illustrated in the graphs in Figures 5.12 and 5.13. The execution time does not increase in the same manner as for Datasets 1 and 2 because the candidate numbers at the given support levels do not show the same significant increases. This is reflected in the graphs illustrating the candidates generated and time spent at each level (Figure 5.13) where the 2% and 0.5% support levels follow each other closely.

inherently exponential.

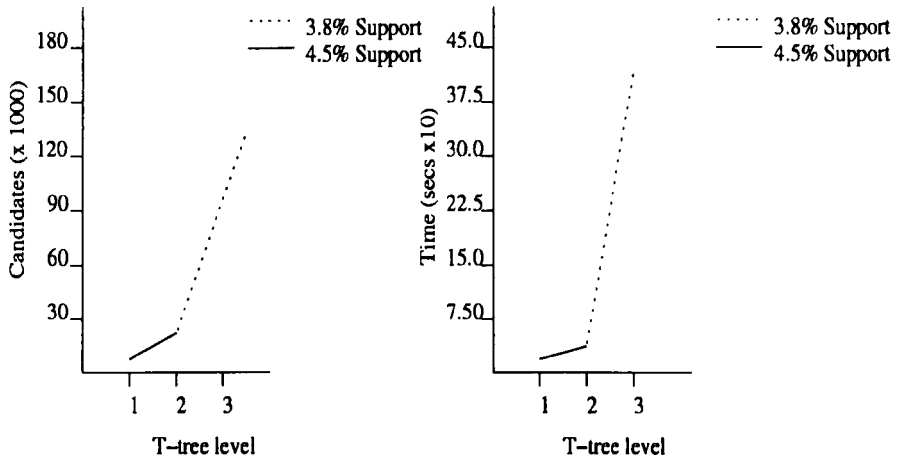


Figure 5.11: Test Group A, Dataset 2: Candidates and time at each level

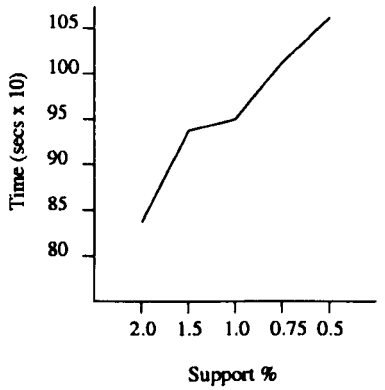


Figure 5.12: Test Group A, Dataset 3: Execution time

Test Group B Results

Four datasets from within Group B (Chapter 4, Test Group B) were used to build the *T*-tree. The test datasets all contained 1000 attributes and had the following properties (as described by Agrawal and Srikant [1994]):

- Dataset 1: 100,000 rows, average of 5 transactions per row.
- Dataset 2: 100,000 rows, average of 20 transactions per row.
- Dataset 3: 200,000 rows, average of 20 transactions per row.

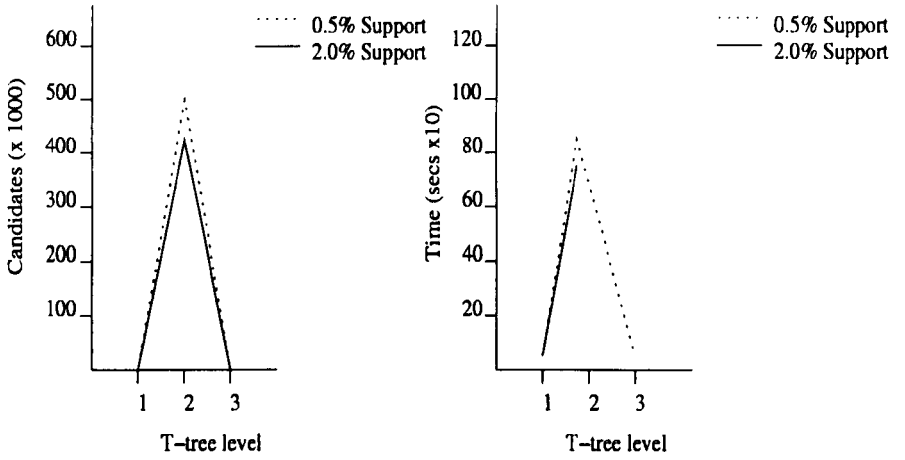


Figure 5.13: Test Group A, Dataset 3: Candidates and time at each level

- Dataset 4: 100,000 rows, average of 20 transactions per row. This dataset contained clustering as described in Chapter 4.

A maximal large itemset is

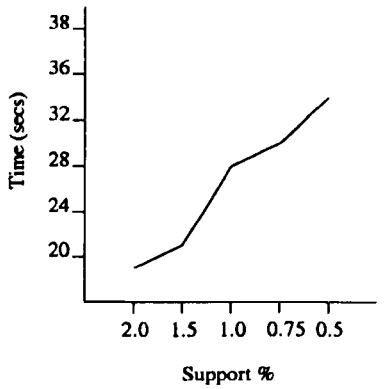


Figure 5.14: Test Group B, Dataset 1: Execution time

The three graphs shown in Figures 5.14 and 5.15 illustrate the performance of Dataset 1. Dataset 1 is very sparse, this is illustrated by two facts;

1. The execution time for the test data is less than 60 seconds even at the 0.5% support level.

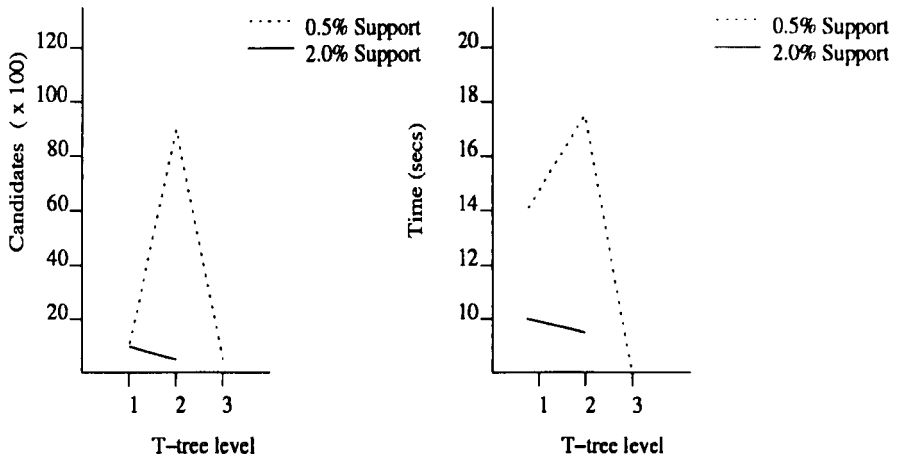


Figure 5.15: Test Group B, Dataset 1: Candidates and time at each level

2. The large number of level 2 candidates generated produces very few level 3 candidates (i.e. very few level 2 candidate itemsets are supported).

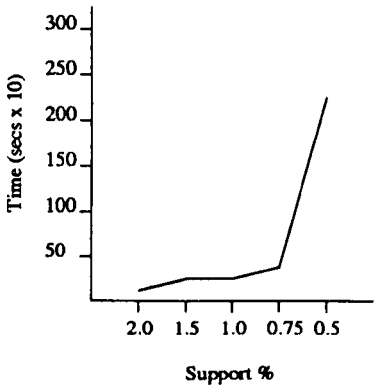


Figure 5.16: Test Group B, Dataset 2: Execution time

The graphs showing the performance of Dataset 2 illustrate the fact that the data is more "dense" than that of Dataset 1, i.e. each record has an average of 20 transactions per row. It can immediately be seen from the graph in Figure 5.16 that the execution time for generating supported sets at the 0.5% level has significantly increased. The graphs in Figure 5.17 show how the increase in time is being used, the number of supported candidate itemsets at the 2 level leads to a very large number of candidate sets being generated at the 3 level.

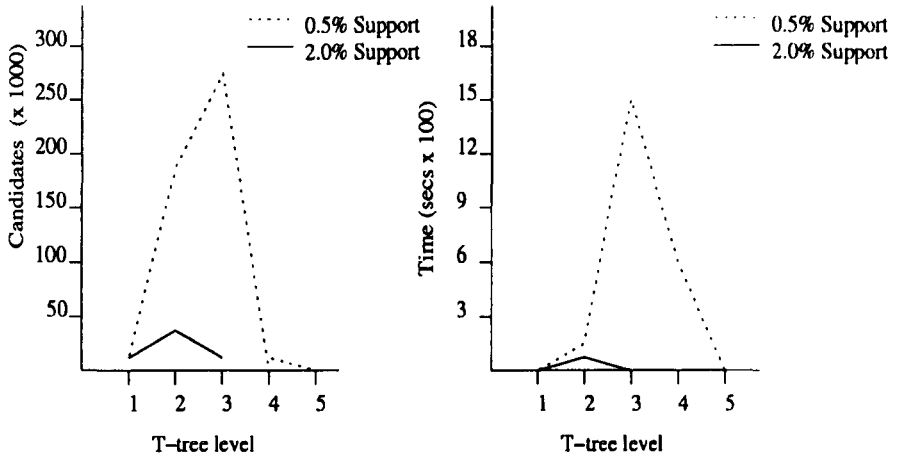


Figure 5.17: Test Group B, Dataset 2: Candidates and time at each level

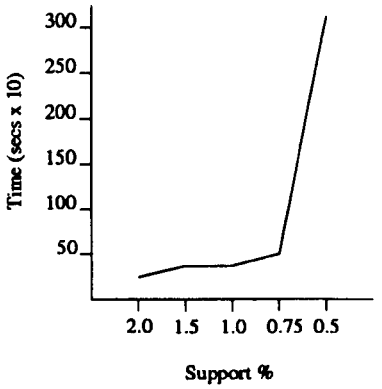


Figure 5.18: Test Group B, Dataset 3: Execution time

Initial inspection of the graphs showing execution times for Datasets 2 (Figure 5.16) and 3 (Figure 5.18) reveal an increased execution time for Dataset 3 but perhaps not as great as would be expected. However Dataset 3 produced no candidate sets below level 4 so the program terminated more rapidly than when running the Dataset 2 which produced candidates at level 5. The number of candidates generated at each level was almost the same as for the smaller dataset (Dataset 2) the increase in both overall execution time and execution time at each T-tree level can be accounted for by the larger size of the P-tree.

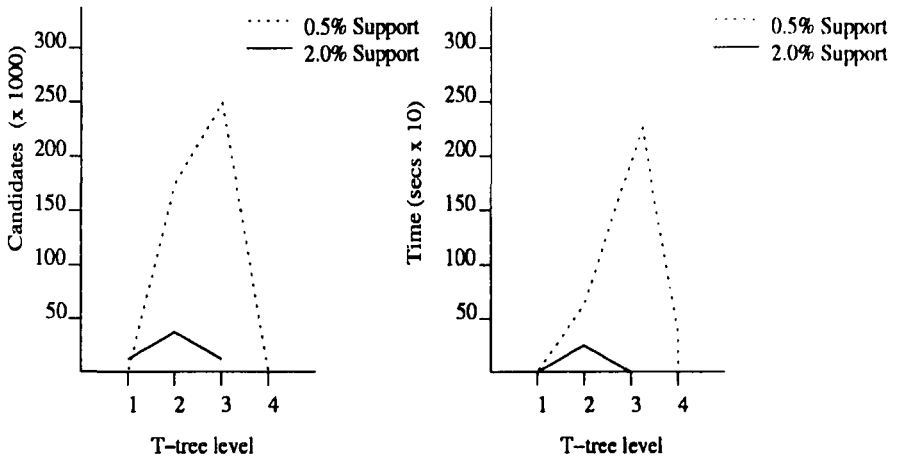


Figure 5.19: Test Group B, Dataset 3: Candidates and time at each level

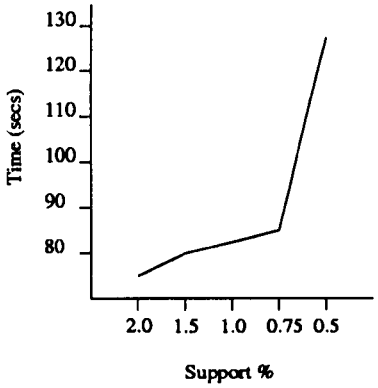


Figure 5.20: Test Group B, Dataset 4: Execution time

Dataset 4 contained a degree of clustering, recall from Chapter 4 (section 4.4.1) that for each record that was generated three similar records were produced. This dataset as can be seen from the graphs in Figures 5.20 and 5.21 produced extremely small numbers of candidates and supported sets at all the support thresholds and therefore executed very quickly.

Test Group C Results

The two facilities management datasets were much smaller than the synthetic datasets and offered the potential to extract "real" supported itemsets that could be related

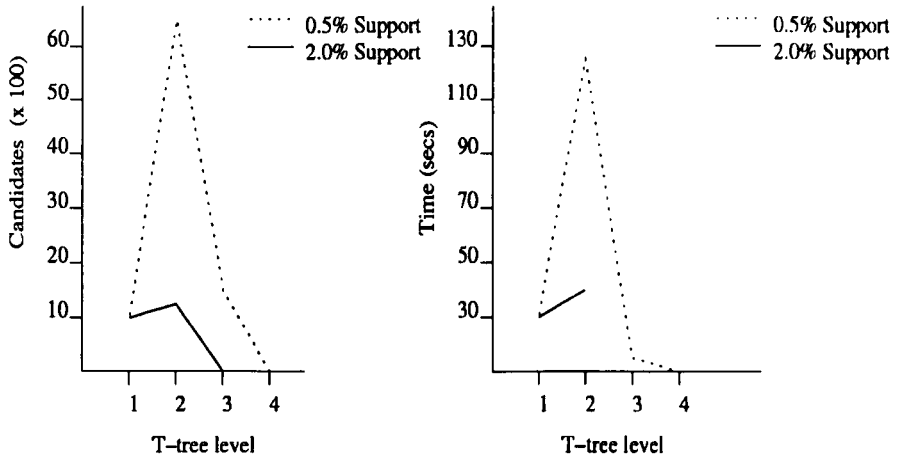


Figure 5.21: Test Group B, Dataset 4: Candidates and time at each level

to the real data files (as illustrated in Appendix A). As can be seen from the graphs

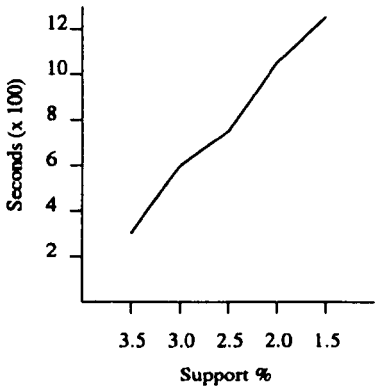


Figure 5.22: Test Group C, Print: Time vs support

(Figures 5.22 and 5.24) execution time and support followed a similar pattern to the previous test groups. For the purposes of this chapter and the imposed "time out" limitation the minimum support achieved was 1.5%. To explore the influence of attribute frequency on the performance of the *T*-tree two attributes were removed from the Fleet Dataset. These were the "previous drivers" and "other drivers" attributes. Both of these consisted of only two binary attributes within the test dataset and therefore removing them would have little influence on the overall density, i.e. down from 8.6% to 7.8%, but would result in far fewer candidates being generated.

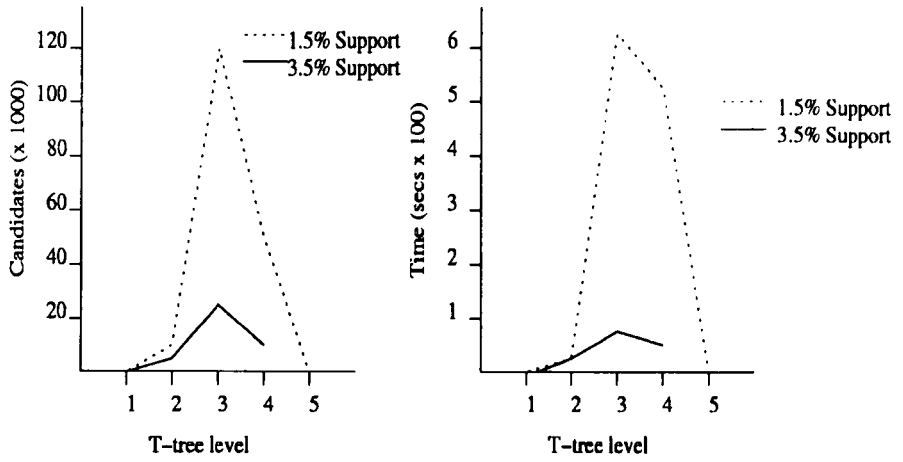


Figure 5.23: Test Group C, Print: Candidates and time at each level

The following graphs (Figures 5.26 and 5.27) illustrate the performance of the T-tree on this dataset. Note, the dataset now consisted of 191 attributes with 15 binary 1's per record, this dataset is called Fleet191 to distinguish it from the original Fleet Dataset.

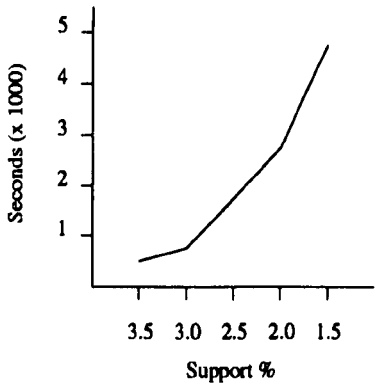


Figure 5.24: Test Group C, Fleet: Time vs support

5.5 Evaluation

Three factors influence the T-tree generation process:

1. The density of the data within the transaction records.

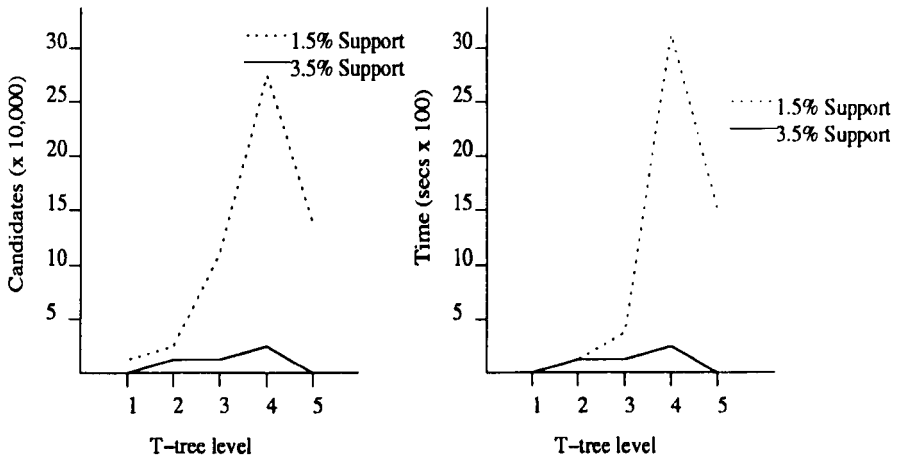


Figure 5.25: Test Group C, Fleet: Candidates and time at each level

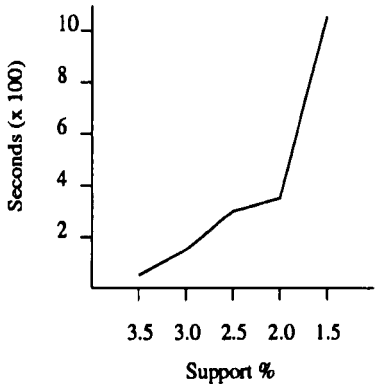


Figure 5.26: Test Group C, Fleet191: Time vs support

2. The number of unique records i.e. the size of the P-tree.

3. The frequency of attributes within the dataset.

The graphs illustrating the performance of Test Group A show how the data density affects the T-tree construction process. Test Group A Datasets 1 and 2 contained 20% density i.e. 40 binary ones in each record but the minimum support that could be examined was 3.89%. By contrast Test Group A Dataset 3 contained 60 binary ones in each record, but it remained feasible to process it at a support level as low as 0.5% because of the low (6%) density. Both Test Group A Datasets 1 and 3 had

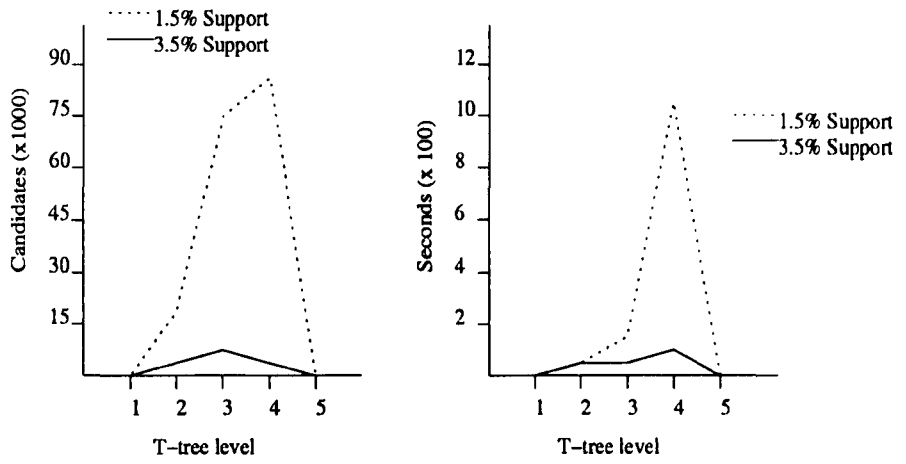


Figure 5.27: Test Group C, Fleet191: Candidates and time at each level

P-trees of similar sizes i.e. approximately 140,000 nodes. When looking at the number of records in the dataset the example of Test Group B Datasets 2 and 3 can be compared. The execution time for the larger dataset (3) is approximately double that for 2 but approximately the same number of candidates are generated at each level, indicating that traversing the larger *P*-tree is taking the extra time.

The facilities management datasets examined in Test Group C could only be examined to 1.5% support within the time limit imposed. Both datasets contained higher densities and a higher percentage of frequently occurring attributes then the "usual" transaction type datasets. When two frequently occurring attributes were removed the execution time and the number of candidates generated at each level significantly decreased, whereas the density was only reduced by 0.8%. This illustrates that a combination of high data density *and* frequently occurring attributes presents a significant challenge for association rule mining.

5.6 Summary

The first part of this chapter dealt with the principle of exploiting the data that was stored in the *P*-tree i.e. the summation of the partial support counts. The idea of

the T-tree was developed to store the total support counts gathered from the partial totals in the P-tree.

The second part of this chapter dealt with testing and evaluating the T-tree when presented with a number of different test files both synthetically generated and extracted from real databases. The tests show that the method is feasible on a variety of different cases, although (as with other methods) execution times rose sharply with very dense data. In the next chapter this will be examined in comparison with existing methods.

Chapter 6

Association Rule Discovery: Case Study

6.1 Introduction

The previous two chapters have dealt with the theory, application and performance assessment of discovering supported itemsets using the P-tree and the T-tree. This chapter has two main themes:

1. To present a case study in discovering association rules from within the Print and Fleet datasets.

2. To compare the use of the P-tree/T-tree combination to discover supported itemsets with a method based on Apriori coupled with the T-tree and establish the advantages and disadvantages of either case.

6.2 The FM Datasets

The two FM datasets supplied by the Royal & SunAlliance were obtained in January 1999. This was after the merger in 1996 of the two insurance companies Royal, predominantly operating in the northern UK and the Sun Alliance predominantly operating in the southern UK. The Fleet Dataset contained all the details of the vehicle management within the enlarged group, however some references still remained to aspects of the vehicles prior to the merger. The Print Dataset contained details of print ordering, supplying and re-stocking this dataset had been integrated

and only contained details of the merged group. The fleet management function (and therefore the dataset) is fairly obvious, i.e. who is driving which vehicle and the vehicle details. The print function of the FM division is different, it acted as a broker between the employees requirements for printed material and the suppliers ability to supply that material at the correct cost. The Print Dataset also recorded stocking levels to ensure over/under ordering was avoided.

6.2.1 Pre-Processing the Data

The two datasets were stored in text format on different computer systems, as outlined in Chapter 4, (section 4.4.1). The initial operation required, to allow the production of association rules from the dataset, was to decide how the attributes were to be split. This is an area of research in itself with authors like Quinlan [1993] advising "caution" when continuous-valued attributes are broken down. Other authors, such as Brin *et al.* [1997], when dealing with continuous-valued attributes like *income*, take the logarithm of the value and round it to the nearest integer to reduce the number of possible answers. For this case study the managers of the Fleet and Print datasets were consulted for their opinions on how to divide the continuous attributes to give the possibility of interesting rules being discovered.
Within the Fleet Dataset three types of attribute were identified:

1. Attributes such as, driver sex, car telephone, fuel card etc. naturally lend themselves to conversion to binary attributes. These were converted into two binary attributes for example, a male would be represented as 1 0. The reason for having two attributes to indicate presence or absence is that to generate supported sets a binary '1' needs to be found.

2. Attributes such as job description or cost centre could be described as "multi-valued" and allowed conversion to binary format because there was only a finite number of potential attributes. For example, an employee's job de-

scription according to the text in the original dataset would be described as, Executive, Senior Manager, Manager or Operations, therefore the attribute "job description" from the original dataset could be broken down into four binary attributes; if the driver's record held "manager" under the job description attribute then that section of the binary dataset would be as follows, 0 0 1 0.

3. Attributes from the original dataset that could be described as "continuous", for example, the "latest mileage" attribute could be represented as 40,000 binary attributes each containing 39,999 zeros and only one binary '1' detailing the exact mileage of the car. Clearly this would be an impractical solution and therefore the "mileage" attribute from the original dataset was divided into a more sensible 20 binary attributes each representing an increment of 2000 miles. A similar approach was adopted towards other "continuous" attributes such as average mpg and licence issue date.

The Print Dataset contained no attributes that naturally lent themselves to binary conversion as described in 1 above. Several attributes such as, unit of sale, unit of purchase and supplier code could be converted using the description of the method from 2 above. The "continuous" type attribute within the Print Dataset was identified for attributes such as, date of last purchase, date of last sale etc.

The use of the FM datasets also showed how the potential results can be affected by the splitting of multi-valued attributes. For example, the original Fleet Dataset contained the attribute *registration date*. The pre-processing of this attribute divided it into thirty six binary values, each representing one month from January 1996 to December 1998. The original attribute could easily have been divided into three binary attributes each representing one year (1996, 1997 and 1998). This would have led to each vehicle record containing one binary '1' in three rather than one binary '1' in thirty six. The fewer number of instances would have *forced* the *registration*

date attribute to become more *visible*. This attribute reduction/expansion could have been investigated for any number of multi-valued attributes and potentially altered the resulting rules. This example reinforces the notion that data mining is not a "one off" exercise but an iterative process where the results from one analysis of the data are interpreted and evaluated, and the the data is mined again using different parameters.

6.2.2 Dataset Characteristics

The binary datasets produced by the pre-processing stage were very different from the "normal" transaction type datasets used in association rule generation research by Agrawal and Srikant [1994], Zaki [1997], Toivonen [1996] etc. The "normal" datasets contain 1000 attributes and at least 100,000 records; however the density of the data is usually about 2%. The FM datasets were both much smaller, the Fleet Dataset contained 194 attributes and 9000 records and the Print Dataset 458 attributes and 6800 records. The density of each of the sets was 8.76% and 5.22% respectively. Furthermore, within the Fleet Dataset several attributes such as driver sex, car telephone and fuel card occurred with great frequency.

6.3 Association Rule Generation

Most of the datasets used for developing and testing the P-tree/T-tree idea have been synthetically generated and as such only contain binary strings that have little "real" meaning. Whilst these are undoubtedly useful for the purposes of research any rules that may be extracted are of no value in "real world" terms. The use of the Royal & SunAlliance's facilities management datasets allows the P-tree/T-tree development to be extended to see if any *useful* or *interesting* rules can be found.

Once the T-tree is constructed it contains all the supported itemsets above the user

defined support threshold. The user may then input a number that represents the number of attributes in the supported sets that they are interested in. For example, if the maximum depth of the T-tree was 7, meaning that the largest supported sets contained 7 attributes, the user may wish to determine all the supported sets that contain 5 attributes above a certain confidence. The following algorithm can be used to derive the confidence for these itemsets:

L = level, N = node under inspection.

$$
\begin{aligned}
&\forall N \in Ttree \\
&\quad if\ attributes(N) \equiv L \\
&\quad \forall n \mid n \subset N \\
&\quad\quad \forall N' \in Ttree \\
&\quad\quad if\ n \equiv N' \\
&\quad\quad n_{sup} = N'_{sup} \\
&\quad\quad confidence = \frac{N_{sup}}{n_{sup}}
\end{aligned}
$$

In the above algorithm the function *attributes* returns the number of attributes in the node under inspection. During the second traversal of the T-tree, i.e. while searching for n, it is sufficient to locate exact match. Any supersets of n that have a contribution to make towards its support will have been included during the T-tree construction. The following example illustrates this; if $n = CDE$ and the node CDE is found in the T-tree the search terminates at this point. Nodes at lower levels such as $CDEF$, $CDEJ$ would be excluded from contributing their support because their support counts would have been included during the construction of level 3 of the T-tree.

6.3.1 Association Rules from the FM Datasets

The above algorithm was applied to the T-trees constructed from the Fleet Datasets and the Print Dataset. In both cases the support level was set to 1.5%, lower support levels taking too long to process. In the case of the Fleet Dataset the confidence was set to 55% and for the Print Dataset the confidence needed to be 38% before any rules were produced. A partial listing of the supported sets from the Fleet Dataset and a full listing of the generated rules from both datasets can be found in Appendix C.

The following are some of the rules generated from the Fleet Dataset:

Group Accounting - London → No other drivers: support = 169/306 confidence = 55%

Male, 30,000 miles → Fuel card: yes: support = 139/248 confidence = 56%

Male, 30,000 miles → Previous vehicle: yes: support = 142/248 confidence = 57%

Male, Pool, Lex → Phone: no: support = 295/518 confidence = 56%

Operations, Fuel card: no, Phone: no → Previous vehicle: no: = support 318/562 confidence = 56%

The first rule states with a confidence of 55% that when the cost centre is Group Accounting - London other drivers are not allowed. The second rule states with 56% confidence that a male with the last recorded mileage of 30,000 miles has a fuel card. The third rule states with 57% confidence that a male with the last recorded mileage of 30,000 miles has owned a previous fleet vehicle. The fourth rule states with 56% confidence that a male who has a pool car from Lex Leasing will not have a car phone. The final rule states with 56% confidence that a person who is in Operations does not have a fuel card and does not have a car phone will not have owned a previous vehicle.

The following are some of the rules generated from the Print Dataset:

Available stock: low, on supplier order: low → sales current month: high: support = 271/691 confidence 39%

Minimum order level: high, On supplier order: high → on customer order: medium: support = 294/763 confidence = 38%

Available stock: low, on customer order: high: → sales last year: low: support = 274/711 confidence = 38%

Available stock: high, on supplier order: high → sales last month: medium: support = 283/736 confidence = 38%

The first rule states with 39% confidence that when available stock is low, and on supplier order is low then sales current month are high. The second rule states with 38% confidence that when minimum order level is high and on supplier order is high then on customer order is medium. The third rule states with 38% confidence that when available stock is low and on customer order is high then sales last year are low. The final rule states with 38% confidence that when available stock is high and on supplier order is high then sales last month are medium.

The rules listed above from the Fleet dataset are easy to read and understand. A person viewing them without domain specific knowledge could easily be expected to follow the implications that the rules display. The Print Dataset, however does not lend itself particularly well to interpretation by a non-domain expert. Whilst the rules generated from the Print Dataset may make sense to someone who is familiar with the ordering and re-stocking system they make little sense as *stand alone* statements. The use of confidence as a mark of rule interestingness was applied for the purposes of these experiments, however, confidence could have been replaced with lift or conviction as discussed in Chapter 3 (section 3.2).

6.4 The P-Tree Advantage

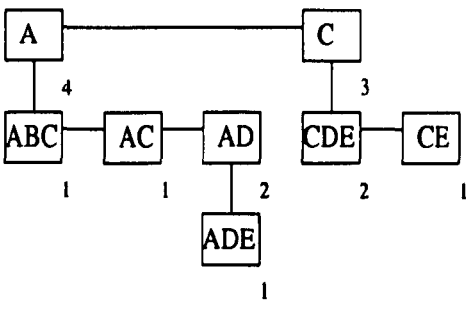
As outlined in the previous chapter (section 5.1) the main advantage of the P-tree is that during total support summation the partially gathered supports and the structure of the P-tree allow the T-tree to be constructed and populated with support totals more easily than if the database itself was being repeatedly read. The following examples compare the construction of a T-tree from repeatedly reading a small dataset with the construction of a T-tree from a P-tree constructed from the same dataset Two operations are defined to be of interest when constructing the T-tree:

1. Node visits (NV): This is the actual number of nodes visited in the T-tree.
2. Node updates (NU): This is the number of nodes that have their support count incremented.

The following Figure (6.1) shows a P-tree constructed from a small dataset. (The partial support counts appear below each P-tree node.) For this example the support threshold is set to 1.

TID	Attributes
10	11100
20	00111
30	00101
40	10010
50	10100
60	10011
70	00111

Datasat



P-tree

Figure 6.1: P-tree and dataset

6.4.1 Support from The Dataset

The following algorithm describes the populating of each level of the T-tree from a dataset, R represents a record in the dataset and T a node in the T-tree. The algorithm terminates when no more candidate sets can be generated.

$\forall R \in Dataset$

$\forall attributes\ X\ in\ R$

$descend\ branch\ X\ in\ Ttree\ (if\ it\ exists)$

$\forall T at\ required\ level$

$if\ T \equiv R$

$T_{sup} = T_{sup} + 1$

$stop$

$if\ T \subset R$

$T_{sup} = T_{sup} + 1$

The top level T-tree candidates are generated and the T-tree resembles Figure 6.2a. Once the initial candidates have been generated the algorithm proceeds as follows. Read initial record (TID 10) R, proceed along the top level of the T-tree until either

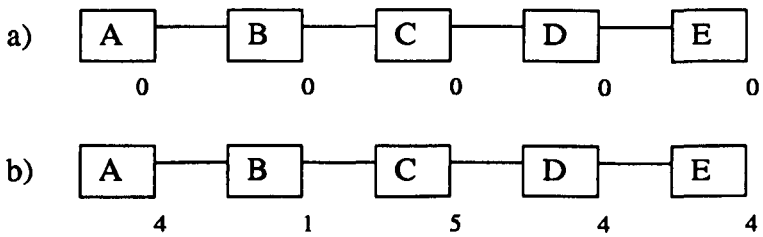


Figure 6.2: Level 1 T-tree

an exact match is found or the T-tree node under inspection is lexicographically after the last attribute in R. In the case of this record the first three nodes are subsets

of R so NU = 3 and NV = 4, the T-tree node D must be checked to ensure completeness.

The second record (TID 20) requires 3 NU (total NU = 6) and all the nodes in the top level to be visited NV = 5 (total NV = 9).

The algorithm proceeds in this manner until all the records in the dataset have been read. The T-tree now resembles Figure 6.2b. Once all the records have been processed the T-tree is pruned (in this case no nodes are removed) and new candidates generated as described in Chapter 5. The total number of NU's for level 1 = 18 and NV's = 33.

The level 2 candidates are now present as illustrated in Figure 6.3a. The NV and NU counters are reset to 0. The initial record from the dataset is re-read and the T-tree is again traversed if the number of binary 1's in the record is less than the current level (i.e. 2) then the next record is read, a record with less attributes than the current level cannot have a contribution to make towards the current level. On this traversal all the nodes present before and including D require a visit (NV = 7) and three AB, AC, and BC require updating (NU = 3).

The next record (TID 20) needs to visit all the nodes descending from the top nodes D and E. Descending the sub-tree emanating from C is not required because the lexicographical ordering of the T-tree ensures that no subset of a record that starts with, for example C, will be found in the C sub-tree. Three nodes are updated, CD, CE and DE, NU = 3 (total NU = 6) NV = 12 (total NV = 19). The total number of node updates at the second level is 15,the total number of node visits is 60 (Figure 6.3b).

The T-tree is pruned, the nodes BD and BE are removed and the new candidate nodes ABC, ACD, ACE, ADE and CDE are inserted. The T-tree now resembles Figure 6.4a.

Level 3 requires a total of 48 node visits and 4 updates. No candidates can be generated below this level therefore the algorithm terminates with the T-tree as shown in 6.4b.

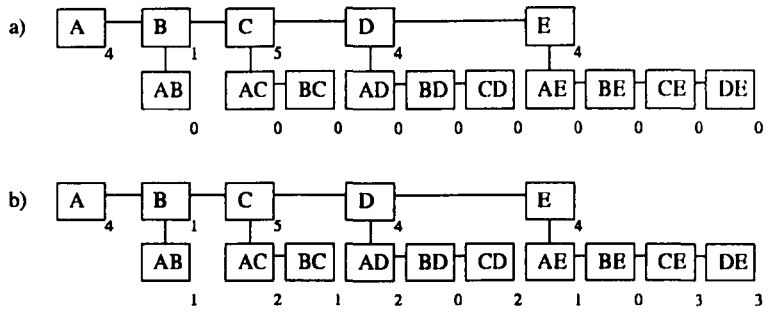


Figure 6.3: Level 2 T-tree

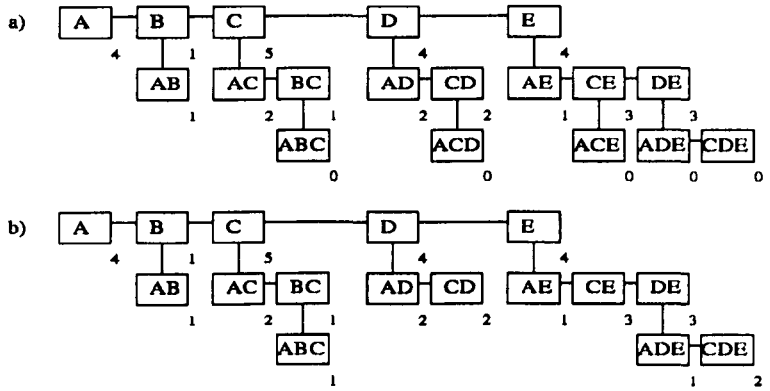


Figure 6.4: Level 3 T-tree

6.4.2 Support from The P-tree

This section uses the same example as the previous section, the only difference being that the T-tree is populated with support counts from the P-tree. The same two counters node updates and node visits will be used. The figures illustrated above, 6.2, 6.3 and 6.4 are exactly the same when produced by both algorithms.

The first node of the P-tree (Figure 6.1) and T-tree are an exact match therefore NV = 1, NU = 1, exact match, stop search.

The next node in the P-tree is ABC; this does not match any of the first level of the T-tree but stops at T-tree node D (because of the lexicographical ordering). Node A in the T-tree does not get updated because $(\{A, B, C\} \setminus \{A\} = \{B, C\}$; node visits = 4 (total NV = 5) node updates = 2 i.e. T-tree nodes B and C are updated (total NU = 3).

The next node in the P-tree is AC, $(\{A, C\} \setminus \{A\} = \{C\}$ therefore there are 4 node visits (total NV = 9) and node update = 1 (total NU = 4). To populate the top level of the T-tree with support counts from the P-tree takes a total of 32 node visits and 11 node updates.

The P-tree is now traversed again but any P-tree nodes with only one element are ignored. The P-tree nodes A and C do not prompt a traversal of the T-tree. The initial P-tree node to prompt a T-tree traversal is therefore ABC. The level 1 T-tree nodes matching the trailing elements of ABC i.e. B and C, are located taking 3 node visits. At this point descend the sub-trees emanating from these nodes, three (level 2) node updates take place because AB, AC and BC are subsets of ABC three node visits also take place. At this point, NU = 3 NV = 7.

The next P-tree node is AC. Locate level 1 node C and descend looking for subsets or exact matches the child of C is AC an exact match therefore update support and stop search. NU = 1, NV = 4. Total node visits 11, node updates 4. To fully populate the second level of the of the T-tree takes 48 node visits and 12 node updates.

Level 3 requires a total of of 27 node visits and 3 updates.

As can be seen from table 6.1 the node updates and node visits at the first level of the T-tree are almost the same between the two approaches. As the T-tree grows larger the differences become greater because the ordering of P-tree allows areas of the T-tree to be excluded from the search space.

	P-tree/*T*-tree			Apriori		
Level	1	2	3	1	2	3
NU	11	12	3	18	15	4
NV	32	48	27	33	60	48

Table 6.1: A comparison of the two methods

6.4.3 Further Comparison

Having established the superiority of the *P*-tree method in the small test case the two facilities management datasets were tested. The support level was set at 3% for both algorithms. The results of these tests are shown in the graphs (Figures 6.5 and 6.6) and tables (Figures 6.2 and 6.3) the table headings NU1 NU2 refer to Node Updates at levels 1 and 2, NV1 and NV2 refer to Node Visits. Although the tables and graphs only illustrate building the *T*-tree to level 3 the differences are significant.

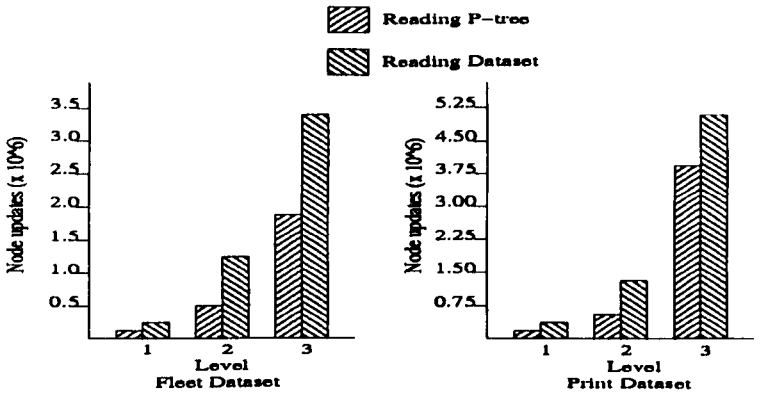


Figure 6.5: Node update comparisons

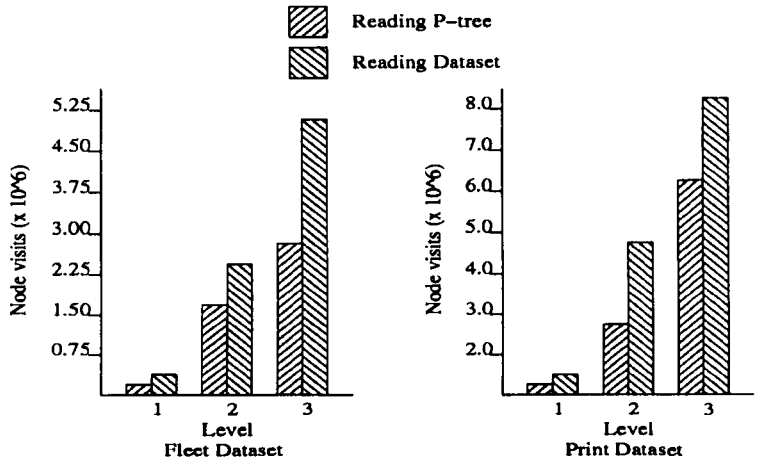


Figure 6.6: Node visit comparisons

File	$NU1$	$NV1$	$NU2$	$NV2$	$NV3$	$NV3$
Fleet data	$10^6 \times 0.153$	$10^7 \times 0.175$	$10^7 \times 0.125$	$10^7 \times 0.249$	$10^7 \times 0.348$	$10^7 \times 0.517$
Print data	$10^6 \times 0.156$	$10^7 \times 0.312$	$10^7 \times 0.142$	$10^7 \times 0.486$	$10^7 \times 0.514$	$10^7 \times 0.815$

Table 6.2: Reading Dataset

6.5 P-tree Advantage Evaluation

From the results provided, both in the small test case and the two facilities management datasets, the P-tree method clearly outperforms the repeated dataset pass method. The P-tree and the T-tree complement each other in a manner that is not reflected when the T-tree alone is used. The two performance indicators, node increments and node updates can be seen as illustrating two of the strengths of the

File	$NU1$	$NV1$	$NU2$	$NV2$	$NV3$	$NV3$
Fleet data	$10^5 \times 0.998$	$10^7 \times 0.138$	$10^6 \times 0.559$	$10^7 \times 0.114$	$10^7 \times 0.197$	$10^7 \times 0.273$
Print data	$10^6 \times 0.112$	$10^7 \times 0.165$	$10^6 \times 0.697$	$10^7 \times 0.268$	$10^7 \times 0.446$	$10^7 \times 0.642$

Table 6.3: Reading P-tree

combined P-tree/T-tree method.

1. Node visits: The lexicographic ordering of the nodes within both trees allows the number of node visits to be significantly reduced because when seeking a T-tree node to update large areas of the T-tree can be ignored.

2. Node updates: The fact that the P-tree contains support counts for subsets means that in some cases only one node update will be required to establish a T-tree nodes support level. Whereas in the repeated pass model each time a particular subset is encountered the relevant T-tree node must be incremented. The number of node updates for the P-tree model would be significantly less if the original dataset had contained duplication.

6.6 Summary

The two themes of this chapter were, the generation of association rules from within the two FM datasets and a comparison between using the P-tree to generate the T-tree and using repeated database passes to generate the T-tree.

Generating association rules from the T-tree is a straightforward task. Once the T-tree contains all the supported sets at the required level the rule generation algorithm extracts the data from the nodes in the T-tree that match the user's requirements, i.e. produce all supported sets at level x. The supported sets are then broken down into their subsets and the support for the subsets compared with the support for the superset. If the user defined confidence level is matched or exceeded then the rule is produced. The FM datasets used in the testing of the rule generation procedure allowed the rules to be interpreted with meaningful results. The generated rules show that some datasets (i.e. Fleet) can produce readily understandable results to a non-expert, whereas datasets such as the Print Dataset would benefit from analysis by a domain expert. The partitioning of the dataset into attribute values assigned

for the tests may be perceived by a domain expert to require revision and a different partitioning strategy may result in different rules being generated.

Using the repeated database pass method to generate the T-tree proved more costly in terms of time, due to the increased number of node updates and node comparisons that needed performing. The use of the lexicographical ordering within both the P-tree and the T-tree enabled the search space for inspecting/updating any one node to be significantly reduced. This benefit was absent from the repeated pass method.

Chapter 7

Conclusion

7.1 Introduction

The broad aim of this research was to explore ways in which supported sets, and therefore association rules could be generated from binary data. The need for more efficient knowledge discovery methods was described in Chapter 1. This need arises because of the ever increasing volumes of data stored in binary format and the increasing demands of database users to fully exploit the potential and untapped value held in their databases. The specific aim of this thesis was to establish that pre-processing data by using specialized data structures could offer significant advantages in terms of memory requirement and execution time.

The generation of association rules from binary databases has been an extremely active area of research since the publication of the AIS algorithm [Agrawal *et al.* 1993]. Two of the fundamental problems encountered in this area have been:

1. Generating all supported sets in one pass of the database.

2. Dealing with "dense" data.

The reduction of the number of database passes has been seen as desirable because constantly reading and re-reading large databases held in secondary storage increases disk I/O and therefore leads to an increased time requirement and load on systems in multi-user environments. The second problem of dealing with "dense"

data can be seen as one of trying to move association rule discovery away from the traditional area of "shopping basket analysis" and explore new sources for rule discovery. The benchmark datasets available from Quest [2000] mirror a shopping transaction dataset; however the "shopper" usually purchases a maximum of 20 items from a range of 1000, i.e. the data has a 2% density. The datasets used by Brin *et al.* [1997] (census data 57% density) and Bayardo [1998] (mushroom classification 17% density) contain a much greater density and therefore present a much greater challenge because the possible number of candidate/supported sets is larger.

One of the issues illuminated by, but not evaluated in, this thesis was the relative data densities for optimum performance by both the P-tree and the T-tree when working together. The P-tree appears to be almost insensitive to the density of the data, processing datasets of 80% density with ease. The T-tree performed well when the data density was lower: this was because it had to generate all the supported combinations present in the P-tree not just store them. It is clear that it is the T-tree that holds the solution to the optimum density question.

7.2 The P-tree/T-tree Advantage

The solution proposed within this thesis to address both the one pass problem and the more dense data problem was the development of the P-tree/T-tree algorithms. The P-tree takes the place of repeatedly reading the dataset. It is a structure that grows in an (almost) linear manner in relation to the number of records and attributes in the dataset. The tree primary advantages of the P-tree are:

1. Compacting the data for more efficient searching.
2. Re-arranging the data from the dataset to take advantage of any duplication.
3. Computation of partial supports are inexpensively achieved during the construction of the tree.

The T-tree in which the supported sets are held was essentially the obverse of the P-tree. The T-tree is constructed in a breadth first manner using an "Apriori" type methodology. During the conversion of the dataset to the P-tree format the lexicographical ordering of the attributes is exploited and this is continued during the construction of the T-tree. The ordering of the attributes in both trees allows large areas of both trees to be excluded from the search space when seeking the support for particular sets.

The test data used ranged from the "benchmark" Quest [2000] datasets to the facilities management datasets provided by the Royal & SunAlliance Insurance Group. The P-tree/T-tree algorithms performed well against all the datasets and when compared with an "Apriori type" repeated pass method used on the FM data.

Using the facilities management data allowed the generation of "meaningful" rules from the datasets. In Chapter 6 the generation and interpretation of these rules was undertaken. The FM data did not reveal any particularly interesting results because to generate any rules the confidence had to be set fairly low. The problem of translating continuous valued attributes into a suitable form for association rule mining was one of the issues illustrated by the use of the FM datasets.

7.3 Wider Issues

During the course of this research the P-tree/T-tree have been linked and used as a complete solution to the supported sets/rule generation problem. However, the use of both trees together is not necessarily the only solution to the problem. The P-tree could easily be seen as working with some other support summation method which would take advantage of the lexicographical ordering and the partial supports. The T-tree construction method in its present form represents a breadth first traversal of the search space but a depth first search may prove more effective in domains of very high density with many repeating attributes.

As observed in Chapter 4, (section 4.4.1) when the dataset contains a high degree of duplication the memory requirement for P-tree storage is considerably reduced. The "best case" for the P-tree algorithm would be in the (unlikely) event of all records being the same i.e. the P-tree would contain one node. The test datasets used in this thesis, with the exception of one (Test Group A, Dataset z), contained no duplication but have shared leading subsets and the P-tree has performed well. Indeed the P-tree at times has almost seemed "data insensitive".

The rules generated from the FM dataset provided a real world example of how association rule generation is more than locating supported sets and then applying the confidence filter to them. One of the more interesting points when dealing with the FM data was the fact that certain combinations of attributes can never occur. For example, within the Fleet Dataset if the attribute "does not have a car phone" was true then the average phone bill attribute would not contain an entry. This type of "if then" relationship between attributes needs to be considered when generating candidate sets to avoid the production of sets that cannot possibly be supported. One could envisage datasets where this type of attribute relationship, if known in advance, could significantly reduce the number of candidate sets generated.

7.4 Current Research

During the writing of this thesis two published works also advocated the use of tree structures for mining association rules.

TreeProjection [Agrawal *et al.* 2000] generates itemsets of frequent sets by successive construction of the nodes of a lexicographic tree of itemsets. Each tree is a subdatabase of the original dataset. Agrawal discusses different search strategies for the tree structure such as depth first, breadth first and a combination of both and examines the trade-offs between the strategies in terms of I/O, memory and computational time requirement.

Recent work published by Han *et al.* [2000] has used a similar structure to the P-tree to achieve dataset compaction. The overall concept is remarkably similar to the P-tree, but the FP-tree (Frequent Path tree), is built in two database passes. The first pass eliminates attributes that fail to reach the required support threshold and orders the remaining attributes by frequency of occurrence. The second pass constructs the tree using a complex series of pointers that join related nodes in the tree structure. Each node in the tree stores a single attribute that is linked to other nodes, each path therefore represents and counts one or more records in the database. The FP-tree also requires that all nodes representing any one attribute be linked into a list. The additional structure of the list facilitates the implementation of an efficient algorithm "FP-growth" which successively generates sub trees from the FP-tree corresponding to each frequent attribute to represent all sets in which the attribute is associated with its predecessors in the tree ordering. The drawback of this method is the additional structural links in the tree, and the repeated accesses to generate the subtrees create problems for efficient implementation for which the tree is too large to fit in main memory. Han however, maintains that by traversing the FP-tree the generation of candidates is avoided and that only"true" frequent sets are produced.

7.5 Future Research

Using the P-tree/T-tree algorithms to generate association rules has proved an effective method with several datasets containing different characteristics. However, several improvements could be made to the system to enhance its performance. Currently the P-tree consists of one complete structure. However, if the tree was constructed as a number of separate trees each having its own root it would be possible to compute interim supports for each tree independently. For example, it would be possible to separate the P-tree presented in Chapter 5 (Figure 5.2) into four sub trees rooted at the nodes A, B, C and D. A record containing ABD for example

would increment support counts within the A-tree, BD within the B-tree and D within the D-tree. The advantage of this is that the completion of the summation to obtain the total supports can be carried out independently within the subtrees used for the interim summation rather than across the whole structure. Partitioning the tree in this manner allows the possibility of using different summation methods for different parts of the structure. a subtree that is large and sparse would invite summation using the T-tree approach, whereas a small dense subtree may benefit from summation via exhaustive methods as described in Chapter 4 (section 4.2.1).

The gain from this approach can be maximized by applying some ordering heuristic similar to those used by Bayardo [1998] and Brin *et al.* [1997]. Suppose the order of frequency of the attributes is known, at least approximately. If the dataset I is ordered so that a_i is the least common attribute and a_n, the most common attributes will be clustered around the right hand side of the P-tree structure. At this end of the P-tree structure exhaustive computation is feasible. Conversely the sparseness of the storage at the left hand side of the P-tree structure is increased.
Essentially this approach is a combination of two heuristics;

1. For a combination of a small set of very frequent attributes exhaustive computation of total supports is efficient.

2. For less frequent sets of attributes, a partitioning of the database corresponding to equivalence classes, i.e. separately rooted P-trees, can be done in a single pass. Each resulting partition can be processed independently.

This dual approach has the advantage that in one full pass of the database computation of the supports of the commonest attributes is completed while at the same tome the database is reorganized into partitions to facilitate the contribution of the least common attributes. A further advantage of this approach is that because the most common attributes have been processed the remaining database partitions may contain relatively few frequent sets enabling the small partitions to be processed in

main memory.

7.6 Summary

The *P*-tree/*T*-tree concept has, in this thesis, proved to be a valid method for producing sets of supported itemsets from datasets with different characteristics. The production of association rules from these sets has been achieved. Whilst no "outstanding" rules were produced from the facilities management sets, the exercise of using these datasets provided a "real world" scenario in which decisions regarding attribute division can have a considerable impact on the results produced.

The research detailed in this thesis can be seen as a starting point from which the *P*-tree/*T*-tree concept could be expanded further. The recent work of Agrawal and Han indicates that other researchers see the idea of data compaction and reorganization as a potential way forward to deal with the ever expanding universe of data.

References

[Adrianns and Zantinge 1996]

P. Adrianns and D. Zantinge. *Data Mining*. Addison Wesley Longman Ltd., 1996.

[Agrawal *et al.* 2000]

R. Agrawal, C. Aggrawal and V.V.V. Prasad. "A tree projection algorithm for the generation of frequent itemsets". In *Journal of Parallel and Distributed Computing*, pages 350–371, 2000.

[Agrawal *et al.* 1993]

R. Agrawal, T. Imielinski and A. Swami. "Mining association rules between sets of items in large databases". In *Proceedings of ACM SIGMOD*, pages 207–216, 1993.

[Agrawal and Srikant 1994]

R. Agrawal and R. Srikant. "Fast algorithms for mining association rules". In *Proceedings of 20th VLDB Conference*, pages 487–499, Morgan Kaufman, 1994.

[Anderson and Moore 1998]

B. Anderson and A. Moore. "Adtrees for fast counting and fast learning of association rules". In *Proceedings of the 4th International Conference on Knowledge Discovery in Databases*, pages 134–138, KDD'98, AAAI Press, 1998.

[Atzeni and DeAntonellis 1993]

P. Atzeni and V. DeAntonellis. *Relational Database Theory.* The Benjamin/Cummings Publishing Company Inc., 1993.

[Bayardo 1998]

R. J. Bayardo. "Efficiently mining long patterns from databases". In *Proceedings of 1998 ACM-SIGMOD International Conference on Management of Data*, pages 85–93, ACM, ACM Press, 1998.

[Bayardo 1997]

R.J. Bayardo. "Brute-force mining of high-confidence classification rules". In *Proceedings of the third international conference on Knowledge Discovery and Data Mining*, pages 123–126, KDD'97, AAAI Press, 1997.

[Bayardo and Agrawal 1999]

R.J. Bayardo and R. Agrawal. "Mining the most interesting rules". In *Fifth ACM Conference on Knowledge Discovery and Data Mining*, pages 145–154, ACM, 1999.

[Bayardo *et al.* 1999]

R.J. Bayardo, R. Agrawal and D. Gunopulos. "Constraint-based rule mining in large, dense databases". In *Proceedings of the 15th International Conference on Data Engineering*, page not sure yet, Not sure yet, 1999.

[Berry and Linoff 1997]

M.J.A. Berry and G. Linoff. *Data Mining Techniques for Marketing, Sales and Customer Support.* Wiley Computer Publishing, 1997. ISBN 0-471-17980-9.

[Bradley *et al.* 1998]

P. Bradley, U. Fayyad and C. Reina. "Scaling cluster algorithms to large databases". In *Proceedings of the Fourth international Conference on Knowledge Discovery in Databases*, pages 9–15, KDD'98, AAAI Press, 1998.

[Bramer 1999]

M. Bramer, editor. *Knowledge Discovery and Data Mining.*. Institute of Electrical Engineers, 1999.

[Brielley and Batty 1999]

P. Brielley and B. Batty. *Knowledge discovery and data mining*, chapter Data mining with neural networks - an applied example in understanding electricity electricity consumption patterns, pages 240–303. Institution of Electrical Engineers, 1999.

[Briemen *et al.* 1984]

L. Briemen, J. Friedman, R. Olshen and C. Stone. *Classification and regression trees*. Wadsworth International Group, 1984.

[Brin *et al.* 1997]

S. Brin, R. Motwani, J. Ullman and S. Tsur. "Dynamic itemset counting and implication rules for market basket data". In *Proceedings of 1997 ACM-SIGMOD International Conference on Management of Data*, pages 225–264, ACM, ACM, 1997.

[Brin and Page 1998]

S. Brin and L. Page. *Dynamic Data Mining: Exploring large rule spaces by sampling*. Technical Report 261, Stanford University, 1998.

[Buchner *et al.* 1999]

A.G. Buchner, J.C.L. Chan, S.L. Hung and J.G. Hughes. *A meteorological knowledge discovery environment*, chapter 10, pages 204–226. Institution of Electrical Engineers, 1999.

[Chaudhuri and Dayal 1997]

S. Chaudhuri and U. Dayal. "An overview of data warehousing and olap technology". In *SIGMOD Record*, pages 65–74, ACM Press, 1997.

[Chen *et al.* 1996a]

M.S. Chen, J. Han and P.S. Yu. "Data mining: An overview from a database perspective". In *IEEE Transactions on Knowledge and Data Engineering*, pages 866–883, 1996.

[Chen *et al.* 1996b]

M.S. Chen, J.S. Park and P.S. Yu. "Data mining for path traversal patterns in a web environment". In *Proceedings of 16th international conference on Distributed computing systems*, pages 385–392, 1996.

[Cheung *et al.* 1996a]

D. Cheung, V. Ng, A. Fu and Y. Fu. "Efficient mining of association rules in distributed databases". In *IEEE Transactions on Knowledge and Data Engineering*, pages 911–922, IEEE Computer Society Press, 1996.

[Cheung *et al.* 1996b]

D. Cheung, V. Ng, A. Fu and Y. Fu. "A fast distributed algorithm for mining association rules". In *Proceedings of 1996 international Conference on Parallel and Distributed Information Systems*, pages 31–42, PIDS'96, IEEE Computer Society Press, 1996.

[Codd 1970]

E. F. Codd. "A relational model for shared databanks". In *Communications of the ACM*, pages 377–387, 1970.

[Codd *et al.* 1993]

E.F. Codd, S.B. Codd and C.T. Salley. *Providing OLAP to User Analysts: An IT Mandate*. Codd and Date Inc., 1993.

[Dilly 1995]

R. Dilly. *Data Mining*. Technical Report, The Queens University Belfast, 1995.

[Dzeroski and Grbovic 1995]

S. Dzeroski and J. Grbovic. "Knowledge discovery in a water quality database". In *Proceedings of the First International Conference on Data Mining and Knowledge Discovery*, pages 81–86, AAAI Press, 1995.

[Ester *et al.* 1996]

D. Ester, G. Gilder, G. Keyworth and A. Toffler. "A magna carta for the knowledge age". *The Information Society*, volume 12, number 3, pages 295–308, 1996.

[Fawley *et al.* 1991]

W.J. Fawley, G. Piatesky-Shapiro and C. J. Matheus. *Knowledge Discovery in Databases: An Overview.* AAAI Press, 1991.

[Fayyad *et al.* 1996]

U. Fayyad, G. Piatetsky-Shapiro and P. Smythe. "Knowledge discovery and data mining: Towards a unifying framework". In *Proceedings of the Second International Conference on Data Mining and Knowledge Discovery*, pages 82–95, AAAI Press, 1996.

[Ganesh *et al.* 1996]

M. Ganesh, Sirvastava J and T. Richardson. "Mining entity-identification rules for database integration". In *Proceedings of the Second International Conference on Data Mining and Knowledge Discovery*, pages 291–294, AAAI Press, 1996.

[Ganti *et al.* 1999]

V. Ganti, J. Gehrke and R. Ramakrishnan. "Mining very large databases". *Computer*, pages 38–45, 1999. IEEE Computer Society Press.

[Gehrke *et al.* 1998]

J. Gehrke, R. Ramakrishnan and V. Gnati. "Rainforrest - a framework for

fast decision tree construction of large datasets". In *Proceedings of 24th International Conference on Very Large Databases*, pages 416–427, VLDB'98, Morgan Kaufman, 1998.

[Gill and Rao 1996]

H. S. Gill and P.C. Rao. *Computing Guide To Data Warehousing*. Que Corp, 1996. ISBN 0-7897-0714-4.

[Han *et al.* 2000]

E.H. Han, G. Karypis and V. Kumar. "Mining frequent patterns without candidate generation". In *Proceedings of ACM SIGMOD*, pages 1–12, 2000.

[Han *et al.* 1997]

E.H. Han, G. Karypis and V. Kumar. "Scalable parallel data mining for association rules". In *Proceedings of 1997 ACM-SIGMOD International Conference on Management of Data*, pages 277–288, ACM, ACM Press, 1997.

[Han *et al.* 1999]

J. Han, L.V.S. Lakshmanan and R.T. Ng. "Constraint-based, multidimensional data mining". *Computer*, pages 45–50, August 1999. IEEE Computer Society Press.

[Hatonen *et al.* 1996]

K. Hatonen, M. Klemettinen, H. Mannila, P. Ronkainen and H. Toivonen. "Knowledge discovery from telecommunication network alarm databases". In *Proceedings of 12th International Conference on Data Engineering*, pages 115–122, IEEE Computer Society Press, 1996.

[Hedberg 1995]

S.R. Hedberg. "The data gold rush". *Byte*, pages 83–88, 1995.

[Hekanaho 1997]

J. Hekanaho. "Ga - based rule enhancement in concept learning". In *Proceed-*

ings of the third international conference on Knowledge Discovery and Data Mining, pages 183–186, KDD'97, AAAI Press, 1997.

[HMSO 1997]

HMSO. *Effective Facilities Management: A Good Practice Guide.* Technical Report, Further Education Funding Council, Her Majesty's Stationery Office, 1997.

[Holsheimer *et al.* 1995]

H. Holsheimer, M. Kersten, H. Mannila and H. Toivonen. "A perspective on databases and data mining". In *First International Conference on Knowledge Discovery and Data Mining*, pages 150–155, 1995.

[Houtsma and Swami 1993]

M. Houtsma and A. Swami. *Set Orientated Mining of Association Rules.* Technical Report RJ 9567, IBM Almaden Research Center, 1993.

[Inmon *et al.* 1997.]

W.H. Inmon, J.D. Welch and K.L. Glassey. *Managing the Data Warehouse.* Wiley Computer Publishing, 1997.

[Jain and Dubes 1988]

A.K. Jain and R.C. Dubes. *Algorithms for clustering data.* Prentice Hall, 1988.

[Johnson 1997]

J. L. Johnson. *Database Models, Languages, Design.* Oxford University Press, 1997.

[Kahng *et al.* 1997]

J. Kahng, W.H.K. Liao and D. McLeod. "Mining generalized term associations: Count propagation algorithm". In *Proceedings of the third International Conference on Knowledge Discovery in Databases*, pages 203–206, KDD'97, AAAI Press, 1997.

[Kamber and Shinghal 1996]

M. R. Kamber and R. Shinghal. "Evaluating the interestingness of characteristic rules". In *Proceedings of the Second International Conference on Data Mining and Knowledge Discovery*, pages 263–266, AAAI Press, 1996.

[Karypis *et al.* 1999]

G. Karypis, E. Han and V. Kumar. "Charmeleon: Hierarchical clustering using dynamic modeling". *Computer*, pages 68–75, 1999. IEEE Computer Society Press.

[Keogh and Smyth 1997]

E. Keogh and P. Smyth. "A probabilistic approach to fast pattern matching in time series databases". In *Third International Conference on Knowledge Discovery and Data Mining*, pages 24–30, KDD'97, AAAI Press, 1997.

[Klemettinen *et al.* 1994]

M. Klemettinen, H. Mannila, P. Rokainen, H. Toivonen and A. I. Verkamo. "Finding interesting rules from large sets of association rules". In *Proceedings of ACM*, pages 401 – 407, ACM, 1994.

[Lakshmanan *et al.* 1996]

L.V.S. Lakshmanan, F. Sadri and I.N. Subramanian. "Schemasql - a language for interoperability in relational multi-database systems". In *Proceedings of 22nd VLDB Conference*, pages 239–250, Morgan Kaufman, 1996.

[Lakshmanan *et al.* 1999]

L.V.S. Lakshmanan, F. Sadri and S.N. Subramanian. "On efficiently implementing schemasql on a sql database system". In *Proceedings of 25th VLDB Conference*, pages 471–482, Morgan Kaufman, 1999.

[Levy 1996]

A. Levy. "Obtaining complete answers from incomplete databases". In *Proceedings of the 22nd VLDB Conference*, pages 402–412, 1996.

[Liu *et al.* 1998]

B. Liu, W. Hsu and Y. Ma. "Integrating classification and association rules". In *Proceedings of the fourth international conference on Knowledge Discovery and Data Mining*, pages 80–86, KDD'98, AAAI Press, 1998.

[MacQueen 1967]

J.B. MacQueen. "Some methods for classification and analysis of multivariate observations". In *Proceedings of the 5th Berkeley symposium on mathematical statistics and probability*, pages 281–297, University of California Press, 1967.

[Manilla 1997]

H. Manilla. "Methods and problems in data mining". In *Proceedings of International Conference on Database Theory*, pages 41–55, Springer-Verlag, 1997.

[Mannila *et al.* 1994]

H. Mannila, H. Toivonen and A. Inkeri Verkamo. *Improved Methods for Finding Association Rules.* Technical Report, Series of Publications C, No. C-1993-65 University of Helsinki, 1994.

[McClean and Scotney 1996]

S. McClean and B. Scotney. *Using Evidence Theory for Knowledge Discovery and Extraction in Distributed Databases.* Technical Report, Number 11 University of Ulster, 1996.

[Mitchel 1999]

T.M. Mitchel. "Machine learning and data mining". *Communications of the ACM*, volume 42, number 11, pages 31–36, 1999.

[Montgomery 1999]

A. Montgomery. *Data Mining: Computer Support for Discovering and Deploying Best Practice in Business and Public Service.* Technical Report 3, BCS SGES, Expert Update, 1999. ISSN 1465-4091.

[Moore and Lee 1998]

A. Moore and M. S. Lee. "Cached sufficient statistics for efficient machine learning with large datasets". *Journal of AI Research*, pages 67–91, 1998.

[Ng *et al.* 1998]

R. T. Ng, L.V.S. Lakshmanan, J. Han and A. Pang. "Exploratory mining and pruning optimizations of constrained association rules". In *Proceedings of 1998 ACM-SIGMOD International Conference on Management of Data*, pages 13–29, ACM, ACM Press, 1998.

[Park *et al.* 1995]

J.S. Park, M.S. Chen and P.S. Yu. "An effective hash-bases algorithm for mining association rules". In *Proceedings of 1995 ACM-SIGMOD International Conference on Management of Data*, pages 175–186, ACM Press, 1995.

[Parthasarathy *et al.* 1998]

S. Parthasarathy, M.J. Zaki and W. Li. "Memory placement techniques for parallel association mining". In *Proceedings of the Fourth international conference on Knowledge Discovery in Databases*, pages 304–308, KDD'98, AAAI Press, 1998.

[Quest 2000]

Quest. *http://www.ibm.almarden.cs/quest/syndata.html*. Data Mining Project, IBM Almarden Research Centre, 2000.

[Quinlan 1993]

J. R. Quinlan. *C4.5: Programs for machine learning*. Morgan Kaufman, 1993.

[Quinlan 1986]

J.R. Quinlan. "Induction of decision trees". In *Machine Learning*, pages 81–106, 1986.

[Ramakrishnan and Gehrke 1998]

R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill, 1998.

[Rastogi and Shim 1998]

R. Rastogi and K. Shim. "Public: A decision tree classifier that integrates building and pruning". In *Proceedings of 24th International Conference on Very Large Databases*, pages 404–415, VLDB'98, Morgan Kaufman, 1998.

[Rymon 1993]

R. Rymon. "An se-tree based characterization of the induction problem". In *Proceedings of the Tenth International Conference on Machine Learning*, pages 268–275, Morgan Kaufman, 1993.

[Rymon 1996]

R. Rymon. "Se-trees outperform decision trees in noisy domains". In *Proceedings of the Second International Conference on Knowledge Discovery in Databases*, pages 331–334, KDD'96, AAAI Press, 1996.

[Rymon 1992]

R. Rymon. "Search through systematic set enumeration". In *Proceedings of the third international conference on the Principles of Knowledge and Reasoning*, pages 539–550, Morgan Kaufman, 1992.

[Savasere *et al.* 1995]

A. Savasere, E. Omiecinski and S. Navathe. "An efficient algorithm for mining association rules in large databases". In *Proceedings of 21st VLDB Conference*, pages 432–444, Morgan Kaufman, 1995.

[Shintani and Kitsuregawa 1996]

T. Shintani and M. Kitsuregawa. "Hash based parallel algorithms for mining association rules". In *Proceedings of the 4th International Conference on*

Parallel and Distributed Information Systems, pages 19–30, IEEE Computer Society Press, 1996.

[Shintani and Kitsuregawa 1998]

T. Shintani and M. Kitsuregawa. "Parallel mining algorithms for generalized association rules with classification hierarchy". In *Proceedings of 1998 ACM-SIGMOD International Conference on Management of Data*, pages 25–36, ACM, ACM Press, 1998.

[Silberschatz and Tuzhilin 1995]

A. Silberschatz and A. Tuzhilin. "On subjective measures of interestingness in kd". In *Proceedings of the First International Conference on Data Mining and Knowledge Discovery*, pages 275–281, AAAI Press, 1995.

[Simoudis *et al.* 1995]

E. Simoudis, B. Livezey and R. Kerber. "Using recon for data cleaning". In *First International Conference on Knowledge Discovery and Data Mining*, pages 282–287, AAAI Press, 1995.

[Smith 1999]

D.S. Smith. "Getting to know clients lifts profits". *The Sunday Times*, pages 17–18, 1999.

[Soderland 1997]

S. Soderland. "Learning to extract text based information from the world wide web". In *Third international conference on knowledge discovery and data mining*, pages 251–254, KDD'97, AAAI press, 1997.

[SPSS 2002]

SPSS. *http://www.SPSS.com.* SPSS Inc. Chicago, IL, USA, 2002.

[Srikant and Agrawal 1995]

R. Srikant and R. Agrawal. "Mining generalized association rules". In *Pro-*

ceedings of the 21st VLDB conference, pages 407–419, VLDB'95, Morgan Kaufman, 1995.

[Srikant *et al.* 1997]

R. Srikant, Q. Vu and R. Agrawal. "Mining association rules with item constraints". In *Proceedings of the third International Conference on Knowledge Discovery in Databases*, pages 67–73, KDD'97, AAAI Press, 1997.

[Stolorz and Musick 1997]

P. Stolorz and R. Musick. *Scalable High Performance Computing for KDD.* Volume 1, Kluwer Academic Publishers, 1997.

[Tamura and Kitsuregawa 1999]

M. Tamura and M. Kitsuregawa. "Dynamic load balancing for parallel association rule mining on heterogeneous pc cluster systems". In *Proceedings of 25th International Conference on Very Large Databases*, pages 162–173, 1999.

[Taylor 1998]

P. Taylor. "Key role for business intelligence". *Financial Times*, pages 1–2, 1998.

[Thomas and Sarawagi 1998]

S. Thomas and S. Sarawagi. "Mining generalized association rules and sequential patterns using sql queries". In *Proceedings of the Third International Conference on Data Mining and Knowledge Discovery*, pages 344–348, AAAI Press, 1998.

[Toivonen 1996]

H. Toivonen. "Sampling large databases for association rules". In *Proceedings of 22nd VLDB Conference*, pages 134–145, Morgan Kaufman, 1996.

[Wang *et al.* 1998]

K. Wang, S. H. W. Tay and B. Liu. "Interestingness-based interval merger for numeric association rules". In *Proceedings of the fourth international conference on Knowledge Discovery and Data Mining*, pages 121–127, KDD'98, AAAI Press, 1998.

[Willet 1988]

P. Willet. "Recent trends in hierarchical document clustering". *Not sure*, volume 24, number 5, pages 577–597, 1988.

[Yoda *et al.* 1997]

K. Yoda, T. Fukuda, Y. Morimoto and S. Morishita. "Computing optimized rectilinear regions for association rules". In *Proceedings of the third international conference on Knowledge Discovery and Data Mining*, pages 96–103, KDD'97, AAAI Press, 1997.

[Zaki 1997]

M.J. Zaki. *Fast Mining of Sequential Patterns in Very Large Databases*. Technical Report, Department of Computer Science, University of Rochester, NY, 1997. Technical Report 668.

[Zaki *et al.* 1997]

M.J. Zaki, S. Parthasarathy, M. Ogihara and W. Li. *New Algorithms for Fast Discovery of Association Rules*. Technical Report, Department of Computer Science, University of Rochester, NY, 1997. Technical Report 651.

Appendix A

The fleet database consisted of the following attributes:

1. Driver name
2. Sex
3. Job description
4. Category
5. Lease agreement
6. Cost centre
7. Make/Model
8. Registration date
9. Latest mileage
10. Last service
11. Fuel card
12. Average mpg
13. Car telephone
14. Average telephone bill
15. Licence issue date
16. Accident
17. Previous vehicle
18. Other drivers

The attributes were held in text format and therefore a pre-processing step to create a binary vector was required. The following list details how and why the different attributes were split:

1. **Driver name**: Not included for reasons of privacy.
2. **Sex**: Two attributes M/F.
3. **Job description**: Four attributes Executive, Senior Manager, Manager or Operations.

4. **Category**: Which operation ran the vehicle, three attributes Royal, Sun Alliance or Pool.

5. **Lease agreement**: Who was the lease agreement with, three attributes Lex, Hertz or Ford.

6. **Cost centre**: Which cost centre was the vehicle charged to, thirty attributes such as, Group Accounting-London, Planning & Research, Sponsorship, Accounting Internal Audit etc.

7. **Make/Model**: Thirty attributes Saloon 2 door petrol, Saloon 2 door diesel, Estate 2 door petrol etc.

8. **Registration date**: Thirty six attributes each month between January 1996 and December 1998.

9. **Latest mileage**: Twenty attributes, from 0 to 40,000 in increments of 2000.

10. **Last service**: Thirty six attributes each month between January 1996 and December 1998.

11. **Fuel card**: Two attributes Y/N.

12. **Average mpg**: Five attributes, 0 to 50 in increments of 10.

13. **Car telephone**: Two attributes Y/N.

14. **Average telephone bill**: Six attributes, 0 to 300 (yearly) in increments of 25.

15. **Licence issue date**: Six attributes, last thirty years in increments of 5.

16. **Accident**: Six attributes, last thirty years in increments of 5.

17. **Previous vehicle**; Two attributes Y/N.

18. **Other drivers**: Two attributes Y/N.

The total number of attributes was 195 with 17 binary 1's in each record.

The print stock database consisted of the following attributes:

1. Stock code.
2. Available stock.
3. Minimum level.
4. Maximum level.
5. Minimum supplier order quantity.
6. Maximum customer order quantity.
7. On supplier order.
8. On customer order.
9. Unit of sale.
10. Unit of purchase.
11. Pack size.
12. List price.
13. Average cost.
14. Latest cost.
15. Old list price.
16. Weight.
17. Date of last sale.
18. Date of last purchase.
19. Sales current month.
20. Sales last month
21. Sales last year.
22. Division/Department.
23. Supplier code.
24. VAT rate.

As with the fleet dataset the attributes were held in a mostly text format that required pre-processing to convert to a binary vector. The following list details how and why the different attributes were split:

1. **Stock code**: Two hundred attributes.
2. **Available stock**: Three attributes based on what was considered low, medium and high stock levels.
3. **Minimum level**: As above.
4. **Maximum level**: As above.

5. **Minimum supplier order quantity**: As above.
6. **Maximum customer order quantity**: As above.
7. **On supplier order**; As above.
8. **On customer order**: As above.
9. **Unit of sale**: Ten attributes detailing each, pads, kilo, book etc.
10. **Unit of purchase**: As above.
11. **List price**; Ten attributes 2,4,6,8,10,15,20,25,30,35.
12. **Average cost**: As above.
13. **Latest cost**: As above.
14. **Old list price**: As above.
15. **Weight**: Ten attributes 1-10 kilos.
16. **Date of last sale**: Twelve attributes one each month for past year.
17. **Date of last purchase**: As above.
18. **Sales current month**: Three attributes based on what was considered low, medium and high stock levels.
19. **Sales last month**: As above.
20. **Sales last year**: As above.
21. **Division/Department**: One hundred attributes, which department was stock ordered by Group Accounting-London, Planning & Research, Sponsorship, Accounting Internal Audit etc.
22. **Supplier code**: Thirty attributes.

23. **VAT rate**: Five attributes, zero, exempt, 17.5%, 15%, unused.

The total number of attributes was 459 with 24 binary 1's in each record.

Appendix B

The following tables detail the results from the Chapter 5 tests for building the T-tree.

Test Group A

A	Candidates at level 1
B	Supported at level 1
C	Execution time level 1
D	Candidates at level 2
E	Supported at level 2
F	Execution time level 2
G	Candidates at level 3
H	Supported at level 3
I	Execution time level 3
J	Total execution time

Key for the following three tables

Dataset description:

Dataset 1: Attributes 200, transaction records 100,000, record density 20%

sup't	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*
4.75%	1000	187	28	17391	-	62	-	-	-	90
4.50%	1000	187	28	17391	-	62	-	-	-	90
4.25%	1000	187	28	17391	6	68	9	-	-	96
4.00%	1000	187	28	17391	15	73	924	-	2	101
3.89%	1000	187	28	17391	2319	88	108880	-	739	855

Results for Test Group A, Dataset 1

Dataset description:

Dataset 2: Attributes 200, transaction records 100,000 (50,000 duplicates), record density 20%

sup't	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*
4.75%	1000	187	18	17391	-	29	-	-	-	47
4.50%	1000	187	28	17391	-	62	-	-	-	90
4.25%	1000	187	28	17391	6	34	9	-	-	52
4.00%	1000	187	28	17391	15	37	924	-	2	57
3.89%	1000	187	28	17391	2319	39	108880	-	349	406

Results for Test Group A, Dataset 2

Dataset description:

Dataset 3: Attributes 200, transaction records 100,000, record density 6%

sup't	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*
2.00%	1000	923	35	425503	51	799	-	-	-	834
1.50%	1000	927	35	429201	77	825	501	-	79	939
1.00%	1000	936	36	437580	151	845	1362	-	70	961
0.75%	1000	941	36	442270	204	896	2198	-	83	1015
0.50%	1000	944	36	445096	345	946	4562	-	99	1081

Results for Test Group A, Dataset 3

Test Group B

A	Candidates at level 1
B	Supported at level 1
C	Execution time level 1
D	Candidates at level 2
E	Supported at level 2
F	Execution time level 2
G	Candidates at level 3
H	Supported at level 3
I	Execution time level 3
J	Candidates at level 4
K	Supported at level 4
L	Execution time level 4
M	Total execution time

Key for the following four tables

Dataset description:

Dataset 1: Attributes 1000, transaction records 100,000, average 5 items per transaction.

sup't	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*
2.00%	1000	30	10	435	1	11	-	-	-	-	-	-	21
1.50%	1000	44	10	946	3	12	70	-	2	-	-	-	22
1.00%	1000	65	11	2080	10	15	237	1	2	-	-	-	28
0.75%	1000	92	12	4186	16	15	499	1	2	-	-	-	29
0.50%	1000	138	14	9453	30	18	1151	1	2	-	-	-	34

Results for Test Group B, Dataset 1

Dataset description:
Dataset 2: Attributes 1000, transaction records 100,000, average 20 items per transaction.

sup't	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*
2.00%	1000	277	30	38226	21	78	317	-	4	-	-	-	112
1.50%	1000	364	31	66066	42	147	1456	-	63	-	-	-	241
1.00%	1000	452	33	101926	116	159	13396	-	93	-	-	-	285
0.75%	1000	510	35	129795	418	218	57106	1	146	-	-	-	399
0.50%	1000	597	38	177906	1479	337	266459	39	1208	406	8	421	2004

Results for Test Group B, Dataset 2

Dataset description:
Dataset 3: Attributes 1000, transaction records 200,000, average 20 items per transaction.

sup't	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*
2.00%	1000	274	62	37401	57	160	1095	-	20	-	-	-	242
1.50%	1000	352	63	61776	97	194	2396	-	35	-	-	-	292
1.00%	1000	440	63	96580	211	227	13161	-	49	-	-	-	339
0.75%	1000	496	68	122769	487	308	54115	1	87	-	-	-	463
0.50%	1000	584	73	170236	1465	401	239404	28	2598	4470	2	654	3726

Results for Test Group B, Dataset 3

Dataset description:
Dataset 4: Attributes 1000, transaction records 100,000 (clustered), average 20 items per transaction.

sup't	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*
2.00%	1000	50	9	1225	5	8	47	-	2	-	-	-	17
1.50%	1000	81	15	3240	6	21	66	-	4	-	-	-	40
1.00%	1000	166	19	13695	25	13	142	-	12	-	-	-	56
0.75%	1000	240	21	28680	28	26	361	-	14	-	-	-	61
0.50%	1000	364	33	66066	70	78	1745	1	17	-	-	-	130

Results for Test Group B, Dataset 4

Test Group B

A	Candidates at level 1
B	Supported at level 1
C	Execution time level 1
D	Candidates at level 2
E	Supported at level 2
F	Execution time level 2
G	Candidates at level 3
H	Supported at level 3
I	Execution time level 3
J	Candidates at level 4
K	Supported at level 4
L	Execution time level 4
M	Candidates at level 5
N	Supported at level 5
O	Execution time level 5
P	Total execution time

Key for the following three tables

Dataset description:
Print Dataset : Attributes 459, transaction records 6,800, average 24 items per transaction.

sup't	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*O*	*P*
3.50%	459	137	7	9316	1132	3	26579	2726	95	19629	-	207	-	-	-	316
3.00%	459	158	7	12403	2686	3	64513	3238	347	22782	-	244	-	-	-	607
2.50%	459	159	7	12561	3342	3	88085	3491	462	27063	-	288	-	-	-	765
2.00%	459	159	7	1261	3549	3	99277	5006	530	49223	-	533	-	-	-	1078
1.50%	459	159	7	1261	3832	3	119816	5321	619	54372	497	582	2243	-	23	1078

Results for Test Group C, Print Dataset

Dataset description:
Fleet Dataset : Attributes 195, transaction records 9,000, average 17 items per transaction.

sup't	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*O*	*P*
3.50%	195	79	2	30816	651	2	14972	1792	62	34306	575	411	2626	-	30	513
3.00%	195	119	2	7021	721	2	30492	2030	81	59263	743	702	3547	18	40	831
2.50%	195	194	2	18721	924	2	80596	3198	124	110669	1443	1324	21798	35	257	1715
2.00%	195	195	3	18915	1040	2	91627	3942	133	114521	3370	1369	104718	360	1259	2272
1.50%	195	195	3	18915	1821	3	22573	5527	173	257589	6050	3096	141740	899	1687	4668

Results for Test Group C, Fleet Dataset

Dataset description:
Fleet Dataset : Attributes 191, transaction records 9,000, average 15 items per transaction.

sup't	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*O*	*P*
3.50%	190	66	2	2145	471	1	6483	620	34	4756	48	58	24	-	1	90
3.00%	191	116	2	6670	540	1	18302	782	50	10096	70	119	48	-	1	179
2.50%	191	189	2	17766	696	2	52371	1447	82	20129	203	242	1313	-	4	337
2.00%	191	190	3	17955	762	2	58073	1903	82	22093	517	257	702	-	8	357
1.50%	191	190	3	17955	1287	2	74974	2913	99	86596	1278	1044	2862	18	34	1187

Results for Test Group C, Fleet191 Dataset

Appendix C

Rules generated from the Fleet Dataset.
****
At 0.55 confidence no rules were generated
for the following;
A => BC
A => BCD
A => BCDE
AB => CD
AB => CDE
ABC => DE
ABCD => E
****
Number of rules = 54 at 0.5500 confidence
****
Rule is Group accounting - London => Other drivers No :- 169/306 = 0.5522
Rule is Group accounting - Liverpool => Phone No :- 163/295 = 0.5525
Rule is Planning and Research => Prev Veh No :- 175/311 = 0.5627
Rule is Group Treasury Corp Finance => Fuel Card No :- 168/303 = 0.5581
Rule is Group Treasury Corp Finance => Other drivers Yes :- 170/301 = 0.5647
Rule is Sponsorship => Prev Veh No :- 167/299 = 0.5585
Rule is RSA Challenge => Other drivers No :- 173/311 = 0.5562
Rule is Group Executive => Fuel Card No :- 182/324 = 0.5617
Rule is Group Legal(life) => Prev Veh Yes :- 169/300 = 0.5633
Rule is Group Strategy => Phone No :- 155/275 = 0.5636
Rule is Life Cornhill => Other drivers Yes :- 154/277 = 0.5559
Rule is Saloon 2Dr Petrol => Prev Veh No :- 166/297 = 0.5589
Rule is Saloon 2Dr Diesel => Fuel Card No :- 175/314 = 0.5573
Rule is Hatch 3Dr Petrol => Phone No :- 174/314 = 0.5541
Rule is Hatch 5Dr Petrol => Prev Veh No :- 180/318 = 0.5660
Rule is Hatch 5Dr Petrol => Other drivers Yes :- 180/318 = 0.5660
Rule is Coupe 3Dr => Phone No :- 178/301 = 0.5647
Rule is Coupe 3Dr auto => Prev Veh No :- 171/302 = 0.5662
Rule is Coupe 3Dr auto => Fuel Card No :- 173/302 = 0.5728
Rule is Commercial Petrol => Other drivers No :- 170/299 = 0.5685
Rule is Commercial Diesel => Fuel Card Yes :- 160/283 = 0.5653
Rule is Commercial Heavy => Phone Yes :- 145/259 = 0.5598
Rule is Apr 96 => Fuel Card Yes :- 152/263 = 0.5779
Rule is Apr 96 => Prev Veh No :- 146/263 = 0.5551

Rule is Jun 96 => Fuel Card No :- 140/250 = 0.5600
Rule is Sep 96 => Other drivers No :- 138/246 = 0.5609
Rule is Jan 97 => Phone Yes :- 140/250 = 0.5600
Rule is Jan 97 => Prev Veh No :- 140/250 = 0.5600
Rule is May 97 => Phone Yes :- 152/273 = 0.5567
Rule is Oct 97 => Prev Veh No :- 135/253 = 0.5744
Rule is Nov 97 => Phone No :- 140/248 = 0.5645
Rule is Nov 97 => Other drivers Yes :- 138/248 = 0.5564
Rule is Mar 98 => Fuel Card No :- 138/246 = 0.5609
Rule is May 98 => Prev Veh Yes :- 141/253 = 0.5573
Rule is Jul 98 => Other drivers No :- 139/243 = 0.5720
Rule is Aug 98 => Prev Veh No :- 148/264 = 0.5606
Rule is Sep 98 => Other drivers Yes :- 138/248 = 0.5564
Rule is Oct 98 => Prev Veh No :- 143/245 = 0.5836
Rule is 30,000 miles => Prev Veh Yes :- 266/476 = 0.5588
Rule is 38,000 miles => Prev Veh No :- 243/436 = 0.5573
Rule is 40,000 miles => Fuel Card No :- 248/448 = 0.5535
Rule is Serv Jan 96 => Phone No :- 143/257 = 0.5564
Rule is Serv Jan 96 => Prev Veh Yes :- 142/257 = 0.5525
Rule is Serv May 96 => Other drivers Yes :- 138/243 = 0.5679
Rule is Serv Sept 96 => Other drivers Yes :- 140/227 = 0.6167
Rule is Serv Oct 96 => Prev Veh No :- 160/280 = 0.5714
Rule is Serv Oct 96 => Other drivers Yes :- 156/280 = 0.5571
Rule is Serv Nov 96 => Prev Veh Yes :- 145/261 = 0.5555
Rule is Serv Nov 97 => Fuel Card Yes :- 135/231 = 0.5844
Rule is Serv Apr 98 => Prev Veh No :- 155/276 = 0.5615
Rule is Serv Aug 98 => Other drivers Yes :- 141/256 = 0.5507
Rule is Fuel Card Yes => Phone Yes :- 140/243 = 0.5761

****
Number of rules = 58 at 0.5500 confidence
****
Rule is Executive Pool => Fuel Card No :- 404/728 = 0.5549
Rule is Female 12,000 miles => Other drivers Yes :- 138/240 = 0.5750
Rule is Male 30,000 miles => Fuel Card Yes :- 139/248 = 0.5604
Rule is Male 30,000 miles => Prev Veh Yes :- 142/248 = 0.5725
Rule is Male 38,000 miles => Prev Veh No :- 136/225 = 0.6044
Rule is 26,000 miles Fuel Card Yes => Phone Yes :- 142/236 = 0.6016
Rule is 28,000 miles Fuel Card Yes => Prev Veh No :- 140/242 = 0.5785
Rule is 30,000 miles Fuel Card Yes => Prev Veh Yes :- 138/250 = 0.5520
Rule is Operations 20 mpg => Prev Veh No :- 260/462 = 0.5627
Rule is Hertz 20 mpg => Prev Veh No :- 332/600 = 0.5533
Rule is Manager 30 mpg => Prev Veh No :- 260/470 = 0.5531
Rule is Operations 50 mpg => Prev Veh No :- 260/462 = 0.5627
Rule is 26,000 miles Phone Yes => Other drivers No :- 146/261 = 0.5593

Rule is Executive 50 Pounds => Prev Veh No :- 213/387 = 0.5503
Rule is Operations 100 Pounds => Other drivers Yes :- 203/367 = 0.5531
Rule is 40 mpg 100 Pounds => Prev Veh Yes :- 160/282 = 0.5673
Rule is Pool 200 Pounds => Prev Veh No :- 284/512 = 0.5546
Rule is 20 mpg 200 Pounds => Other drivers Yes :- 170/306 = 0.5555
Rule is Executive 250 Pounds => Other drivers Yes :- 207/375 = 0.5520
Rule is Hertz 300 Pounds => Prev Veh No :- 290/521 = 0.5566
Rule is 10 mpg 300 Pounds => Other drivers Yes :- 168/304 = 0.5526
Rule is Operations 0-5 yrs => Prev Veh No :- 214/380 = 0.5631
Rule is 20 mpg 0-5 yrs => Prev Veh No :- 170/298 = 0.5704
Rule is 50 mpg 0-5 yrs => Other drivers No :- 166/300 = 0.5533
Rule is 150 Pounds 0-5 yrs => Other drivers No :- 141/254 = 0.5551
Rule is 200 Pounds 0-5 yrs => Prev Veh No :- 152/256 = 0.5937
Rule is Operations 6-10 yrs => Prev Veh No :- 207/370 = 0.5594
Rule is Lex 6-10 yrs => Other drivers Yes :- 291/517 = 0.5628
Rule is 20 mpg 6-10 yrs => Other drivers Yes :- 170/299 = 0.5685
Rule is 300 Pounds 6-10 yrs => Other drivers Yes :- 146/254 = 0.5748
Rule is 50 Pounds 11-15 yrs => Other drivers No :- 137/235 = 0.5829
Rule is 20 mpg 16-20 yrs => Prev Veh No :- 159/288 = 0.5520
Rule is Female 25-30 yrs => Other drivers Yes :- 417/752 = 0.5545
Rule is Royal 25-30 yrs => Other drivers Yes :- 285/513 = 0.5555
Rule is Fuel Card Yes 25-30 yrs => Other drivers Yes :- 411/733 = 0.5607
Rule is 30 mpg 25-30 yrs => Other drivers Yes :- 151/272 = 0.5551
Rule is 50 Pounds 25-30 acc yrs => Other drivers No :- 136/236 = 0.5762
Rule is 20 mpg 21-25 acc yrs => Prev Veh No :- 168/295 = 0.5694
Rule is 40 mpg 21-25 acc yrs => Other drivers Yes :- 168/279 = 0.6021
Rule is 200 Pounds 21-25 acc yrs => Other drivers Yes :- 146/259 = 0.5637
Rule is 0-5 yrs 16-20 acc yrs => Prev Veh No :- 138/244 = 0.5655
Rule is 11-15 yrs 16-20 acc yrs => Other drivers Yes :- 140/243 = 0.5761
Rule is 16-20 yrs 16-20 acc yrs => Other drivers No :- 137/245 = 0.5591
Rule is 21-25 yrs 16-20 acc yrs => Prev Veh Yes :- 138/250 = 0.5520
Rule is 10 mpg 11-15 acc yrs => Other drivers Yes :- 165/296 = 0.5574
Rule is 20 mpg 11-15 acc yrs => Prev Veh No :- 162/292 = 0.5547
Rule is 300 Pounds 11-15 acc yrs => Prev Veh No :- 140/251 = 0.5577
Rule is 0-5 yrs 11-15 acc yrs => Other drivers No :- 146/252 = 0.5793
Rule is 6-10 yrs 11-15 acc yrs => Other drivers Yes :- 144/260 = 0.5538
Rule is 100 Pounds 6-10 acc yrs => Other drivers Yes :- 144/251 = 0.5737
Rule is 250 Pounds 6-10 acc yrs => Prev Veh Yes :- 145/248 = 0.5846
Rule is 300 Pounds 6-10 acc yrs => Prev Veh No :- 144/257 = 0.5631
Rule is 25-30 yrs 6-10 acc yrs => Other drivers Yes :- 137/236 = 0.5805
Rule is 40 mpg 0-5 acc yrs => Prev Veh Yes :- 161/292 = 0.5513
Rule is 150 Pounds 0-5 acc yrs => Prev Veh Yes :- 152/266 = 0.5714
Rule is 250 Pounds 0-5 acc yrs => Other drivers Yes :- 150/261 = 0.5747
Rule is 6-10 yrs 0-5 acc yrs => Other drivers Yes :- 136/246 = 0.5528

****
Number of rules = 58 at 0.5500 confidence
****
Rule is Male Pool Lex => Phone No :- 295/518 = 0.5694
Rule is Male Pool Fuel Card No => Phone No :- 425/757 = 0.5614
Rule is Male Lex Fuel Card No => Phone No :- 427/771 = 0.5538
Rule is Pool Lex Fuel Card No => Phone No :- 288/522 = 0.5517
Rule is Female Executive Phone No => Other drivers Yes :- 323/575 = 0.5617
Rule is Male Operations Phone No => Prev Veh No :- 333/585 = 0.5692
Rule is Operations Fuel Card No Phone No => Prev Veh No :- 318/562 = 0.5658
Rule is Operations Phone Yes Prev Veh No => Other drivers Yes :- 305/553 = 0.5515
Rule is Executive Phone No Prev Veh No => Other drivers Yes :- 316/574 = 0.5505

****

Number of Nodes in P-tree = 11654

Support Level = 280 (3%)

Supported at level 1 = 119
Execution time level 1 = 4.00 seconds

Candidates generated for level 2 = 7021
Supported at level 2 = 721
Execution time level 2 = 2.00 seconds

Candidates generated for level 3 = 30492
Supported at level 3 = 2030
Execution time level 3 = 92.00 seconds

Candidates generated for level 4 = 59263
Supported at level 4 = 743
Execution time level 4 = 747.00 seconds

Candidates generated for level 5 = 3547
Supported at level 5 = 18
Execution time level 5 = 47.00 seconds

The following are some of the singles and pairs of
supported sets from the Fleet Dataset

0 Support for this itemset = 4518
1 Support for this itemset = 4482
2 Support for this itemset = 2248
3 Support for this itemset = 2262
4 Support for this itemset = 2224
5 Support for this itemset = 2266
6 Support for this itemset = 2959
7 Support for this itemset = 3079
8 Support for this itemset = 2962
9 Support for this itemset = 3037
10 Support for this itemset = 3072
11 Support for this itemset = 2891
12 Support for this itemset = 306
13 Support for this itemset = 295
14 Support for this itemset = 296
15 Support for this itemset = 322
16 Support for this itemset = 320
17 Support for this itemset = 299
18 Support for this itemset = 311
19 Support for this itemset = 316
20 Support for this itemset = 301
21 Support for this itemset = 315
22 Support for this itemset = 299
23 Support for this itemset = 287
24 Support for this itemset = 311
25 Support for this itemset = 318
26 Support for this itemset = 324
27 Support for this itemset = 288
28 Support for this itemset = 287
29 Support for this itemset = 287
30 Support for this itemset = 300
31 Support for this itemset = 326
32 Support for this itemset = 293
33 Support for this itemset = 281
34 Support for this itemset = 291
35 Support for this itemset = 319
36 Support for this itemset = 317
37 Support for this itemset = 305
38 Support for this itemset = 297
39 Support for this itemset = 287
40 Support for this itemset = 303
41 Support for this itemset = 303
42 Support for this itemset = 314
43 Support for this itemset = 342
44 Support for this itemset = 332
45 Support for this itemset = 303
46 Support for this itemset = 311
47 Support for this itemset = 291
48 Support for this itemset = 316
49 Support for this itemset = 318
50 Support for this itemset = 286
51 Support for this itemset = 284
52 Support for this itemset = 310
53 Support for this itemset = 301
54 Support for this itemset = 291
55 Support for this itemset = 289
56 Support for this itemset = 302
57 Support for this itemset = 283
58 Support for this itemset = 299
59 Support for this itemset = 316
60 Support for this itemset = 281
61 Support for this itemset = 283
62 Support for this itemset = 341
63 Support for this itemset = 325
64 Support for this itemset = 283
65 Support for this itemset = 298
66 Support for this itemset = 281
67 Support for this itemset = 443
68 Support for this itemset = 427
69 Support for this itemset = 448
70 Support for this itemset = 492
71 Support for this itemset = 438
72 Support for this itemset = 478
73 Support for this itemset = 432
74 Support for this itemset = 449
75 Support for this itemset = 423
76 Support for this itemset = 429
77 Support for this itemset = 443
78 Support for this itemset = 448
79 Support for this itemset = 482
80 Support for this itemset = 478
81 Support for this itemset = 476
82 Support for this itemset = 449
83 Support for this itemset = 439
84 Support for this itemset = 442
85 Support for this itemset = 436
108 Support for this itemset = 443
109 Support for this itemset = 427
110 Support for this itemset = 448
111 Support for this itemset = 492
112 Support for this itemset = 438
113 Support for this itemset = 478

114 Support for this itemset = 432
115 Support for this itemset = 449
116 Support for this itemset = 423
117 Support for this itemset = 429
118 Support for this itemset = 443
119 Support for this itemset = 448
120 Support for this itemset = 482
121 Support for this itemset = 478
122 Support for this itemset = 476
123 Support for this itemset = 449
124 Support for this itemset = 439
125 Support for this itemset = 442
126 Support for this itemset = 436
127 Support for this itemset = 448
137 Support for this itemset = 280
164 Support for this itemset = 4483
165 Support for this itemset = 4517
166 Support for this itemset = 1798
167 Support for this itemset = 1787
168 Support for this itemset = 1834
169 Support for this itemset = 1765
170 Support for this itemset = 1816
171 Support for this itemset = 4462
172 Support for this itemset = 4538
173 Support for this itemset = 1502
174 Support for this itemset = 1463
175 Support for this itemset = 1502
176 Support for this itemset = 1529
177 Support for this itemset = 1480
178 Support for this itemset = 1524
179 Support for this itemset = 1505
180 Support for this itemset = 1479
181 Support for this itemset = 1482
182 Support for this itemset = 1489
183 Support for this itemset = 1512
184 Support for this itemset = 1533
185 Support for this itemset = 1529
186 Support for this itemset = 1487
187 Support for this itemset = 1507
188 Support for this itemset = 1464
189 Support for this itemset = 1494
190 Support for this itemset = 1528
191 Support for this itemset = 4440
192 Support for this itemset = 4560
193 Support for this itemset = 4558
194 Support for this itemset = 4442

The sets at level 2 are...

0 2 Support for this itemset = 1150
1 2 Support for this itemset = 1098
0 3 Support for this itemset = 1122
1 3 Support for this itemset = 1140
0 4 Support for this itemset = 1125
1 4 Support for this itemset = 1099
0 5 Support for this itemset = 1121
1 5 Support for this itemset = 1145
0 6 Support for this itemset = 1522
1 6 Support for this itemset = 1437
2 6 Support for this itemset = 744
3 6 Support for this itemset = 746
4 6 Support for this itemset = 705
5 6 Support for this itemset = 764
0 7 Support for this itemset = 1531
1 7 Support for this itemset = 1548
2 7 Support for this itemset = 760

Rules generated from the Print Dataset.

At 0.3800 confidence no rules were generated for the following;

A => BC

****
Number of rules = 8 at 0.3800 confidence
****

Rule is List price 8 => Sales last yr: Low :- 264/689 = 0.3831
Rule is Weight 2kg => Sales last yr: Low :- 248/652 = 0.3803
Rule is Weight 5kg => Sales last mth: high :- 247/647 = 0.3817
Rule is Last sale Jan => Sales last mth: med :- 228/582 = 0.3917
Rule is Last sale Apr => Sales last yr: Low :- 244/577 = 0.3882
Rule is Last sale Dec => Sales last yr: Low :- 213/531 = 0.4011
Rule is Last purch Sept => Sales last mth: high :- 208/546 = 0.3809
Rule is Last purch Dec => Sales current mth: high :- 215/560 = 0.3839

****
Number of rules = 17 at 0.3800 confidence
****
Rule is Avail stock low Min lev med => Sales last yr: low :- 288/784 = 0.3850
Rule is Min lev low Max lev low => Cust ord med :- 267/697 = 0.3803
Rule is Avail stock med Max lev high => On sup ord med :- 274/707 = 0.3875
Rule is Avail stock high Min sup ord low => On sup ord med :- 290/760 = 0.3815
Rule is Avail stock med Min sup ord med => Sales crnt mth low :- 276/716 = 0.3854
Rule is Avail stock low On sup ord low => Sales crnt mth high :- 297/761 = 0.3902
Rule is Min lev med On sup ord med => Sales crnt mth low :- 297/761 = 0.3902
Rule is Avail stock high On sup ord high => Sales last mth med :- 283/736 = 0.3845
Rule is Min lev low On sup ord high => On cust ord med :- 287/743 = 0.3862
Rule is Max lev low On sup ord high => On cust ord med :- 294/763 = 0.3853
Rule is Max lev high On sup ord high => Sales crnt mth low :- 301/791 = 0.3805
Rule is Avail stock low On cust ord high => Sales last yr: low :- 274/711 = 0.3853
Rule is Max lev low On cust ord high => Sales crnt mth low :- 274/708 = 0.3870
Rule is Avail stock med Sales crnt mth low => Sales last mth high :- 292/767 = 0.3807
Rule is Max lev med Sales crnt mth med=> Sales last yr: low :- 293/761 = 0.3850
Rule is On cust ord high Sales crnt mth med => Sales last yr: low :- 272/713 = 0.3814
Rule is Avail stock low Sales crnt mth high => Sales last yr: low :- 296/760 = 0.3894
****