MINING RULES
IN SINGLE-TABLE AND MULTIPLE-TABLE DATABASES

A Dissertation Presented

by

Laurentiu B. Cristofor

Submitted to the Office of Graduate Studies, University of Massachusetts Boston, in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

June 2002

Computer Science Program

MINING RULES

IN SINGLE-TABLE AND MULTIPLE-TABLE DATABASES

A Dissertation Presented

by

Laurentiu B. Cristofor

Approved as to style and content by:

Dan A. Simovici, Professor
Chairperson of Committee

Richard L. Tenney, Professor
Member

William R. Campbell, Associate Professor
Member

Ivan Stojmenovic, Professor
Member

Dan A. Simovici, Program Director
Computer Science Program

Peter Fejer, Chairperson
Computer Science Department

ABSTRACT

MINING RULES
IN SINGLE-TABLE AND MULTIPLE-TABLE DATABASES

June 2002

Laurentiu B. Cristofor, B.S., Politehnica University
M.S., Politehnica University
M.S., University of Massachusetts Boston
Ph.D., University of Massachusetts Boston

Directed by Professor Dan A. Simovici

Data mining represents the process of extracting interesting and previously unknown knowledge from data. In this thesis we address the important data mining problem of discovering association rules in single-table and multiple-table databases, and we also introduce a generalization of the database concept of functional dependency: the purity dependencies, which can be viewed as a type of rules that are informationally more significant than association rules. An association rule expresses the dependence of a set of attribute-value pairs, also called *items*, upon another set of items (*itemset*). The mining of association rules is performed in two stages: the discovery of frequent sets of items from the data and the generation of association rules from the frequent itemsets. The first stage is computationally intensive and the second stage has the drawback of possibly generating thousands of rules, thus making the analysis of the results hard to perform by a human analyst. Our contributions relate to both stages of this process. We analyze stage one and introduce two new algorithms among which algorithm *Closure* improves upon the performance of the classic algorithm *Apriori*. We define the concept of an informative cover of the set of association rules and present algorithm *CoverRules* for computing such a cover, which is in practice one or more orders of magnitude smaller than the set of association rules. Next, we investigate how the mining of association rules should be performed when the data reside in several tables organized in a star schema. We show how to modify the *Apriori* algorithm to compute the support of itemsets by analyzing the star schema tables or their outer join. Whereas association rules express the dependency between itemsets, purity dependencies express the dependency between sets of attributes. We introduce the concept of purity dependency as a generalization of the concept of functional dependency and we show that these purity dependencies satisfy properties that are similar to the Armstrong rules for functional dependencies. As an application of our theory we present an algorithm for discovering purity dependencies that can be used to predict the value of a target attribute.

To my parents.

Acknowledgments

The work presented in this thesis has been done in the Department of Computer Science (former Department of Mathematics and Computer Science) of the University of Massachusetts Boston. My thanks go to my advisor, Dan Simovici, for teaching me to express problems formally and for his helpful suggestions and discussions; to Richard Tenney, William Campbell, and Elizabeth O'Neil for their feedback and suggestions regarding my thesis; and to the professors at UMass/Boston with whom I had the pleasure to interact: Ethan Bolker, Richard Eckhouse, Peter Fejer, Colin Godfrey, George Lukas, Joan Lukas, John Lutts, Kenneth Newman, Carl Offner, Patrick O'Neil. I also want to thank my wife, Dana Cristofor, for her support and for being my collaborator in the work on which Chapter 2 and Chapter 5 are based.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

1.1 Data Mining

Data Mining (DM), also known as Knowledge Discovery in Databases (KDD), represents the process of discovering interesting and previously unknown information from data. There is no restriction to the type of data that can be analyzed by data mining. We can analyze data contained in a relational database, a data warehouse, a web server log, or a simple text file. Data mining algorithms and technologies are researched and implemented for any type of data worth analyzing. One of the first applications of data mining consisted of the analysis of the transaction data stored by supermarkets in view of improving the way products are arranged on shelves [AIS93b]. Other applications of data mining consist of:

- the analysis of past credit behavior, for determining the eligibility of clients for new credit;

- the analysis of user response to mail advertising, for eliminating the costs of sending such ads to persons that do not respond;

- the analysis of user preferences, for providing users with recommendations of other products or services;

- the analysis of web server logs, for determining user browsing patterns and reorganizing the design of web sites to make their navigation easier;
- the building of applications for automatic analysis and classification of images gathered by space probes.

The interest in data mining comes from the fact that the information obtained through this process can provide critical support for decision making and, in general, can help in making sense of large amounts of data. With the current availability of massive data storage solutions, many organizations have collected gigabytes, and in some cases terrabytes of data, which are impossible to analyze without the help of proper techniques. It has become increasingly apparent that, while data are readily available, there is a lack of techniques and tools that can “mine” and retrieve the interesting information. The difficulties related to this task are not only technical, but also of a philosophical nature, because they require definitions for concepts such as information and interestingness of information. In the past ten years, data mining has grown from being a rather exotic topic to being an important research subject in the database community, with several international conferences (KDD, PKDD, PAKDD, EGC) being exclusively dedicated to it.

Data mining is an interdisciplinary field, as it requires knowledge of Databases, Machine Learning, and Statistics. Data mining has set itself apart from machine learning and statistics by providing results that are easy to interpret and by allowing the processing of large volumes of data. This has resulted both in a review of the techniques developed for machine learning and statistics — seeking to improve their scalability and ease of use — and in an introduction of new techniques, algorithms, and even problems.

In general, the process of data mining consists of the following stages:

1. selection of data to analyze — the stage in which we determine what data should be analyzed;
2. extraction of data — the stage in which we isolate the selected data;
3. cleaning of data — the stage in which we deal with data errors and/or omissions;
4. conversion of data — the stage in which we convert the data to a format suitable for the next stage;
5. mining of information — the stage in which we execute a data mining algorithm;
6. information visualization — the stage in which the results of the data mining algorithm are processed and presented in a form that allows their easy interpretation.

Stages 2–4 are also commonly known as data preprocessing. Some of these stages are usually performed several times by varying some parameters or strategies. The focus of this thesis is on data mining algorithms that can be applied in stage 5.

In [AIS93a], the authors have set apart three types of data mining problems as being characteristic for the field: classification, discovery of associations, and discovery of sequences. We briefly present these problems in the next subsections, after which, in Sections 1.2 and 1.3, we introduce the problems that will be analyzed in this thesis.

1.1.1 Classification

In the classification problem we are looking for a set of rules that can be used to partition the data into disjoint sets. For example, in the case of a mail advertising company, we are interested in determining rules that can allow us to decide whether a person on the mailing list is likely to respond to the ad or not. Or, in the case of credit approval, we are interested in rules that allow us to determine the credit eligibility of customers.

To determine the classification rules we use previously classified data. If this is not available, we can use the services of a domain expert for classifying a subset of the data. This already classified data is called **training set** and is analyzed to determine the profile of each class, and to extract the rules that allow us to classify an instance of the data.

An instance of the classification problem has been studied in machine learning under the name of decision tree learning ([Qui93], [Mit97]). This name is due to the method of representing the classification rules in the form of a decision tree. A decision tree is a tree in which the leaves of the tree are labeled with a class, each internal node is labeled with an attribute, and each branch is labeled with an attribute value. To classify an instance of data, we start at the root of the decision tree. We descend recursively through the tree by analyzing the attribute that labels the current node and following the branch labeled with the attribute value for our instance. The classification of the instance data is determined upon reaching a leaf node, by reading its label.

1.1.2 Associations

This problem has been introduced by Agrawal, Imielinski, and Swami in [AIS93a] and [AIS93b], and is one of the problems that we analyze in this thesis. The associations, or association rules, were initially introduced in the context of analyzing supermarket transactions for discovering the buying patterns of customers. A supermarket transaction consists of a list of items purchased by a customer, and the supermarket database consists of a list of such transactions. An association rule has the form $X \to Y$, where X and Y represent sets of items and are called **antecedent** and **consequent**, respectively. Such a rule indicates that customers who have bought the items in X tend to also purchase the items in Y. To assess the importance and interestingness of the association, two measures are used. The **support** of the rule indicates the number of transactions that confirm the rule — the number of transactions in which both X and Y are present. The **confidence** of the rule represents the ratio between the number of transactions that contain $X \cup Y$ and the number of transactions that contain X. As an example, consider the rule *cheese* $\to$ *tomatoes* with support 30% and confidence 40%. The rule indicates that 30 percent of the supermarket transactions involve the buying of cheese and tomatoes, and that 40 percent of the transactions that involve the buying of cheese also involve the buying of tomatoes. The association rules problem consists of discovering all association rules with support and confidence larger than some user specified thresholds. The association rules problem is presented in detail in Section 1.2.

An important thing to note is that not all association rules actually indicate a causal relationship between their antecedent and consequent. The confidence of a rule $X \to Y$ gives us only the estimated conditional probability $P(Y|X)$.

To determine the degree of correlation between X and Y, we need to use other measures than the confidence (see [BA99] for a discussion of other measures).

1.1.3 Sequences

If we add the time factor to the associations problem, then we obtain the problem of discovering sequences. In this problem, data has an attached timestamp, as in stock market data. The rules that involve time are now called sequences. An example of such a sequence is: *when the stock of company A goes up and does not fall for one week, the stock of company B will also go up at the end of the week.* This problem has been analyzed in [AS95b], [AS95c], and [AS96b].

1.2 Association rules

The association rules[1] problem has been introduced in [AIS93b] in the context of analyzing supermarket data. We begin our discussion of this problem with an analysis of possible organizations of data.

1.2.1 Database considerations

The supermarket data analyzed in [AIS93b] has a simple formal model. Let $\mathcal{I} = \{i_1, \ldots, i_n\}$ be a set whose elements are called **items** and let $\mathcal{T} = \{t_1, \ldots, t_m\}$ be a multiset whose elements are called **transactions**, where each transaction t_i represents a subset of $\mathcal{I}$. The multiset $\mathcal{T}$ can be stored in a special table with binary attributes $\mathcal{I}$ and an additional transaction identifier attribute *tid*. Such a table is called a **binary table** or **binary database**.

[1] Because the term association rule is a central concept in our thesis, we will sometime use the acronym AR to refer to it.

Table 1.1 shows the contents of a binary table for $\mathcal{I} = \{i_1, i_2, i_3\}$ and $\mathcal{T} = \{\{i_1\}, \{i_2\}, \{i_1, i_2, i_3\}, \{i_1, i_2, i_3\}, \{i_1, i_2\}\}$.

tid	i_1	i_2	i_3
1	1	0	0
2	1	0	0
3	1	1	1
4	1	1	1
5	1	1	0

Table 1.1: An example of a binary table

In a binary table, the presence of an item in a transaction is indicated by a value of 1 for the attribute corresponding to the item, and its absence is indicated by a 0. Thus, the cardinality of the domain of the attributes corresponding to items is two. In most tables used in practice it is infrequent to have only binary attributes. Attributes that take values in a limited domain are called **nominal** or **categorical** attributes, and attributes whose domain is large are called **numerical** attributes. In the case of a nominal attribute, we will extend the use of the term **item** to designate a particular attribute–value pair (A, a), denoting $A = a$ (e.g. *color=red*). Numerical attributes are handled by transforming them to nominal attributes through the process of dividing their domain into intervals and assigning identifiers to these intervals. For example, for an attribute *age* we could assign the identifier *teenager* for the age interval 10–19, resulting in item *age=teenager*. This transformation of numerical attributes into nominal attributes is called **discretization**.

Contemplating Table 1.1, we can also consider the problem of data density. Intuitively, we consider one binary database to be denser than another when its attributes take the value 1 more often. We say that a database is **sparse** when its rows contain a relatively small number of 1's, otherwise we call the database **dense**. These notions are informal, however, as there is no formal measure for quantifying the density of a database.

In the case of mining association rules, for reasons of simplicity, we will usually assume that databases are binary. All algorithms that work on binary databases, however, can easily be modified to work in the case of non binary databases whose numerical attributes have been discretized.

1.2.2 Problem definition

We begin by defining the central concepts of the AR mining problem. For historical reasons, the terminology that was introduced in [AIS93b] in the context of analyzing supermarket data has been kept intact in all successive papers published on this subject, even when the papers discussed different applications.

Definition 1.1 *Let $\mathcal{I} = \{i_1, \ldots, i_n\}$ be a set whose elements are called* ***items*** *and let $\mathcal{T} = \{t_1, \ldots, t_m\}$ be a multiset whose elements are called* ***transactions****, where each transaction $t_i \subseteq \mathcal{I}$. A subset of $\mathcal{I}$ is called an* ***itemset****. An itemset of cardinality k is called a k-**itemset**. The* ***support*** *of an itemset I is denoted by $\mathsf{supp}(I)$ and defined as:*

$$\mathsf{supp}(I) = \frac{|\{t \in \mathcal{T} \mid t \supseteq I\}|}{|\mathcal{T}|}.$$

We define $\mathsf{supp}(\emptyset) = 1$.

An ***association rule*** *is an implication of the form $A \to C$, where A and C are two disjoint itemsets called* ***antecedent*** *and* ***consequent****, respectively. We refer to the antecedent and consequent of a rule r by using the notations $\mathsf{antc}(r)$*

and $\mathsf{cons}(r)$*. We refer to the items of rule* r *by* $\mathsf{items}(r) = \mathsf{antc}(r) \cup \mathsf{cons}(r)$*. The* ***support*** *of an association rule* r *is defined as:*

$$\mathsf{supp}(r) = \mathsf{supp}(\mathsf{items}(r)).$$

The ***confidence*** *of an association rule* r *is defined as:*

$$\mathsf{conf}(r) = \frac{\mathsf{supp}(\mathsf{items}(r))}{\mathsf{supp}(\mathsf{antc}(r))}.$$

□

Note that the support of an itemset or rule, and the confidence of a rule, take values in the interval [0, 1]. To simplify the notation, we will use $i_1 i_2 i_3$ to denote the itemset $\{i_1, i_2, i_3\}$. In most algorithms, itemsets are actually implemented as lists where the items appear in lexicographic order, which allows them to be processed easily and efficiently.

It is important to keep in mind that an itemset stands for a conjunction of clauses that require the presence of all of its items in a transaction.

Based on Definition 1.1, the association rules mining problem is defined as follows: given user specified minimum support and minimum confidence thresholds, denoted as $\mathsf{minsupp}$ and $\mathsf{minconf}$, discover all association rules from $\mathcal{T}$ having support and confidence greater or equal to $\mathsf{minsupp}$ and $\mathsf{minconf}$, respectively.

In the context of this problem, one more definition is necessary.

Definition 1.2 *Let* $\mathsf{minsupp}$ *be a support threshold. An itemset having support greater or equal to* $\mathsf{minsupp}$ *is said to be* ***frequent****. A frequent itemset that is maximal with respect to set inclusion is called* ***large***[2]*.* □

[2] *Some papers also use the term large to refer to frequent itemsets. We consider the term large to be more appropriate in designating a maximal frequent itemset.*

In [AIS93b], the AR problem was divided into two subproblems:

1. finding all frequent itemsets

2. generating all association rules starting from the frequent itemsets found

Finding all frequent itemsets is the more computationally intensive of the two subproblems, and research efforts have focused on finding more efficient algorithms. Some methods search for only a subset of all frequent itemsets that has the property of summarizing or of allowing to infer the information on all frequent itemsets. For example, this subset can represent the set of large itemsets [Bay98] or the set of closed[3] itemsets [PBT99b].

The rule generation stage is usually much more efficient than the first stage, but has the drawback of possibly producing a very large number of rules, more than a human analyst could handle. It is not unusual to obtain tens of thousands of rules. To address this problem, researchers have focused either on defining measures for the interestingness or degree of surprise of a rule, or on determining subsets of rules from which all other rules can be inferred using a set of inference rules. Our contribution following the latter approach is presented in Chapter 3.

Note that both these stages have exponential complexity in their worst case scenarios. In the case of finding the frequent itemsets, the complexity is in terms of the number of items. Indeed, given n items, we can form 2^n itemsets, all of which could be frequent. In the case of generating the association rules, the complexity is determined by the number of frequent itemsets and by their size, because for an m-itemset we can generate $2^m - 2$ association rules. This value was obtained by considering the number of possible antecedents of a rule and by excluding the

[3] A closed itemset I has the property that no item in $\mathcal{I} - I$ appears in all transactions containing I. The concept of closed itemset is discussed in Chapter 2

case of the empty set and that of the maximal set, which would imply an empty antecedent and, respectively, an empty consequent. Although both subproblems have exponential complexity, in practice the first one is more costly because for the computation of the support of an itemset we need to access the transactions of $\mathcal{T}$ and verify the inclusion of the itemset in each transaction.

1.2.2.1 Mining frequent itemsets

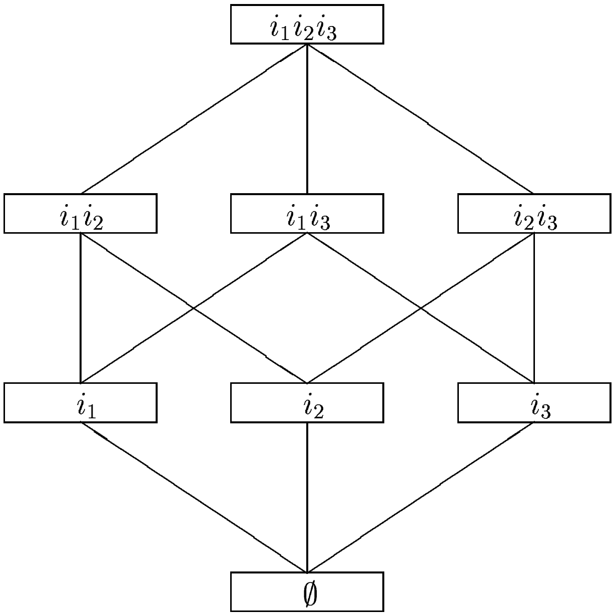


Figure 1.1: Lattice of itemsets for $\mathcal{I} = \{i_1, i_2, i_3\}$

For a given set $\mathcal{I}$, the potential frequent itemsets form a lattice, which we call the **lattice of itemsets**. Figure 1.1 illustrates such a lattice of itemsets in the case when $\mathcal{I} = \{i_1, i_2, i_3\}$. To discover frequent itemsets, algorithms have to explore this lattice. Based on how the traversal is made, algorithms for mining frequent itemsets can be classified as:

1. **breadth-first** algorithms, which explore the lattice level by level.

2. **depth-first** algorithms, which explore the lattice by moving from a node to a next level node if possible, or otherwise to the next node at the current level.

Based on the point from which they start the lattice traversal, algorithms can also be categorized as:

1. **bottom-up** algorithms, which start their traversal at the bottom of the lattice.

2. **top-down** algorithms, which start their traversal at the top of the lattice.

A trivial algorithm for mining frequent itemsets would just compute the support of all itemsets from the lattice. The performance of this algorithm would then be $\Theta(2^{|\mathcal{I}|})$. Practical algorithms attempt to minimize the number of nodes examined in their traversal of the lattice of itemsets. Ideally, an algorithm would only examine those itemsets that are actually frequent, so as to compute their support. Because it is impossible to know in advance which itemsets are frequent, algorithms will use various techniques to minimize the number of itemsets that they examine, and to ensure that no frequent itemsets are overlooked.

1.2.2.2 Mining association rules

After the frequent itemsets are computed, we have to generate the association rules. A very simple algorithm could work as follows: for each frequent itemset I, and for each of its non-empty subsets I_a, we generate the rule $I_a \rightarrow I - I_a$ if its confidence is equal to or greater than $\mathsf{minconf}$. This algorithm would generate all possible association rules and an improved version of it is presented in the following section.

1.2.3 The *Apriori* algorithm

In [AS94b], Agrawal and Srikant introduced the classic algorithm *Apriori*. The technical report version of this paper (see [AS94a]) contains more detailed information about the algorithm implementation. The algorithm has also been presented in [AMS96]. In this section we describe *Apriori* and the ideas behind it.

Apriori mines the frequent itemsets in a bottom-up, breadth-first fashion, and its pseudocode is given by Algorithm 1.1. The algorithm works iteratively. At iteration k, *Apriori* starts with a collection of possible k-frequent itemsets called **candidate** itemsets, scans the data to determine which candidates are frequent, and then generates candidates for the next iteration by applying the procedure *apriori-gen* to the set of frequent k-itemsets. The algorithm starts initially with a collection of candidate itemsets consisting of all 1-itemsets.

Algorithm 1.1 ***(Apriori)***

Input: *the transactions $\mathcal{T}$ containing items $\mathcal{I}$, and* $\mathsf{minsupp}$.

Output: *the list of frequent itemsets.*

Uses: *list of candidates $\mathcal{C}$, list of frequent itemsets $\mathcal{F}$, list of frequent k-itemsets $\mathcal{K}$.*

```
1 Initialize $\mathcal{C}$ to all 1-itemsets.

2 Scan $\mathcal{T}$ and count the number of occurrences for each candidate itemset.
```

3 Set the support of each candidate itemset to the ratio of its number of occurrences and $|\mathcal{T}|$. Discard the candidates that have support less than minsupp *and copy the frequent candidates to $\mathcal{F}$ and $\mathcal{K}$. Clear $\mathcal{C}$.*

4 If this is step $|\mathcal{I}|$, then go to step 7.

5 Add to $\mathcal{C}$ the itemsets generated by the procedure apriori-gen when applied to $\mathcal{K}$. Clear $\mathcal{K}$.

6 If $\mathcal{C}$ is not empty, then go to step 2.

7 Return $\mathcal{F}$.

■

To minimize the effort needed to determine which candidates are frequent, the candidate itemsets are stored in a special structure called **hash-tree**. During the scan of the data, when examining each transaction, instead of verifying for each candidate whether it is contained in that transaction, we use the hash-tree to perform this test for only a subset of all candidates.

The hash-tree is a special tree containing two types of nodes: hash-nodes that are used for the interior nodes, and list-nodes that are used for the leaf nodes. A hash-node consists of a hashtable of (item, node) pairs. A list-node consists of a list of itemsets. An empty hash-tree consists of one empty list-node. When we add itemsets[4] to the hash-tree, we start from the root and descend through the tree until we reach a list-node. At level k (assume that the root's level is 1), we examine the type of the current node. If we have a hash-node, we retrieve the hash entry corresponding to the k-th item of our itemset. If there is no such entry,

[4] An itemset is implemented as a list in which the items appear in lexicographic order.

we make one by creating a new list-node and adding entry (k-th item, list-node) to the hash-node, otherwise we just descend one level to the node retrieved from the entry. If we have a list-node which is not full, then we add the itemset to this node, otherwise we replace the list-node with a hash-node and add all the itemsets of the list-node, and our itemset, to this new hash-node.

Once the hash-tree has been filled with a set of candidate itemsets, we can use it to retrieve those itemsets that could be included in a transaction $t = \{i_1, \dots, i_m\}$. For this, we descend from the root of the hash-tree until we reach a number of list-nodes. The itemsets contained in these list-nodes are potentially included in t. For the descent, to insure that we retrieve all possible subsets of t, we perform the following procedure: at level 1, we hash on all possible items $i_k \in t$ for $k = 1 \dots m$, then at each next level, if we got there by hashing on i_j, we will continue to hash on all items $i_k \in t$ with $k > j$. This method traces the paths that we would have followed during the addition to the hash-tree of all subsets of t, which guarantees that we will not miss any such subset.

The *apriori-gen* procedure uses a couple of techniques to minimize the number of non-frequent candidates generated. The design of the *Apriori* algorithm is based on several observations resulting from the following property:

Theorem 1.1 *Let $I_1 \subset I_2$ be two itemsets. Then we have $\mathsf{supp}(I_1) \geq \mathsf{supp}(I_2)$.*

Proof. All transactions containing I_2 also contain I_1 and there may exist transactions that contain I_1 but do not contain I_2, hence $\mathsf{supp}(I_1) \geq \mathsf{supp}(I_2)$. ∎

Theorem 1.1 can also be rephrased in the following way: if an itemset is frequent, then all of its subsets must be frequent, and if an itemset is infrequent, then all of its supersets must also be infrequent. This means that if, for example, we have a frequent itemset $I = i_1 i_2 \dots i_k$, then all of its subsets are frequent and, in particular, subsets $i_1 i_2 \ldots i_{k-2} i_{k-1}$ and $i_1 i_2 \ldots i_{k-2} i_k$ are frequent and they are also successive $(k-1)$-itemsets (according to the lexicographic order), because they have the same first $k-2$ items. To limit the number of candidates generated, *apriori-gen* uses this property by only generating a new candidate from frequent k-itemsets that have the same first $k-2$ items. This method is also efficient because, if we keep the itemsets in lexicographic order, which only requires careful planning of the algorithm, then we need to verify only a small number of pairs of itemsets.

To further limit the creation of non-frequent candidate itemsets, the *apriori-gen* method verifies for each of the candidate k-itemsets generated, whether all of their subsets of cardinality $k-1$ are frequent. Candidates that fail this test are discarded.

Through these two techniques, the *apriori-gen* procedure ensures that we consider only candidate itemsets that do not contradict the current knowledge that we have about the frequent itemsets (the *a priori* knowledge). The pseudocode of the *apriori-gen* procedure is given in Algorithm 1.2.

Algorithm 1.2 ***(apriori-gen)***

```
Input: the list of frequent $k$-itemsets $\mathcal{F}$.

Output: the list of candidate $(k+1)$-itemsets.

Uses: the list of candidate $(k+1)$-itemsets $\mathcal{C}$.

1 Scan $\mathcal{F}$ and for each two itemsets $I_1$, $I_2$ that have the same first $k-2$ items
  do:

    1.1 Generate candidate $I_c = I_1 \cup I_2$.
    1.2 If all subsets of $I_c$ having cardinality $k$ can be found in $\mathcal{F}$, then add $I_c$ to $\mathcal{C}$.
2 Return $\mathcal{C}$.
```

▮

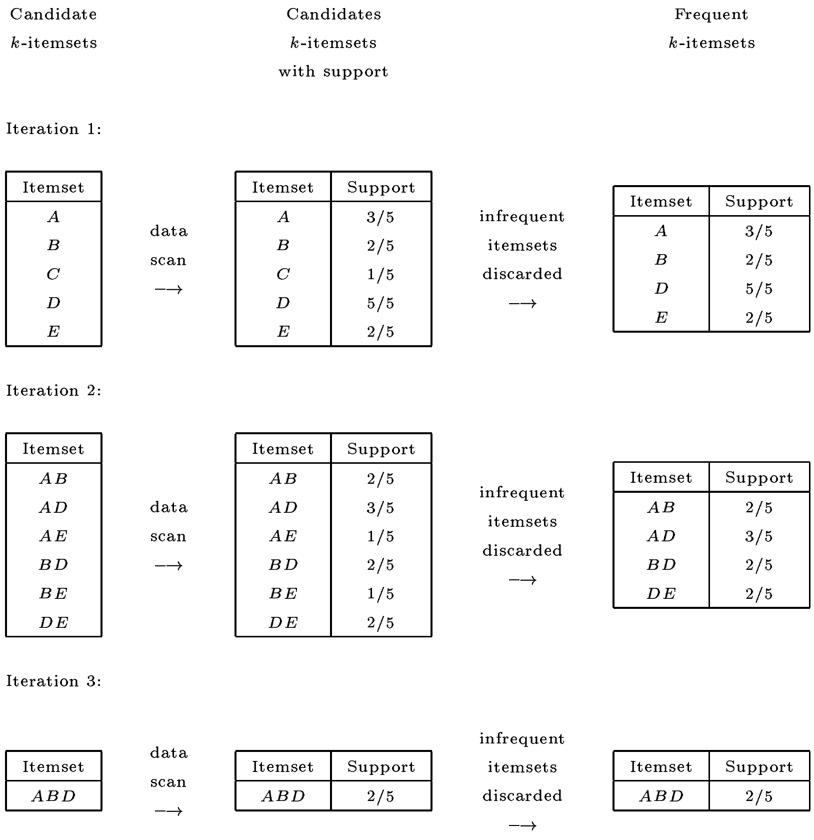
To illustrate the steps of the *Apriori* algorithm we use the binary database presented in Table 1.2 that has 5 items, $\mathcal{I} = \{A, B, C, D, E\}$. We will refer to this database in future examples by using the name $\mathcal{D}$.

tid	A	B	C	D	E
1	1	1	0	1	1
2	0	0	1	1	0
3	1	1	0	1	0
4	0	0	0	1	1
5	1	0	0	1	0

Table 1.2: The binary database $\mathcal{D}$

Figure 1.2 shows how *Apriori* discovers the frequent itemsets in table $\mathcal{D}$ when the minimum support value is set as $\frac{2}{5}$. After each iteration k, the procedure *apriori-gen* generates the candidate $(k+1)$-itemsets out of the frequent k-itemsets. *Apriori* performs three database scans to compute all frequent itemsets. After the third pass over the database, the procedure *apriori-gen* cannot generate any new candidates (because there is only one frequent itemset: $ABCD$), so the algorithm stops.

On data with characteristics similar to supermarket data, that is, when the number of items appearing in a transaction is small, the *Apriori* algorithm runs



Candidate k-itemsets

Candidates k-itemsets with support

Frequent k-itemsets

Iteration 1:

Itemset
A
B
C
D
E

data scan ⟶

Itemset	Support
A	3/5
B	2/5
C	1/5
D	5/5
E	2/5

infrequent itemsets discarded ⟶

Itemset	Support
A	3/5
B	2/5
D	5/5
E	2/5

Iteration 2:

Itemset
AB
AD
AE
BD
BE
DE

data scan ⟶

Itemset	Support
AB	2/5
AD	3/5
AE	1/5
BD	2/5
BE	1/5
DE	2/5

infrequent itemsets discarded ⟶

Itemset	Support
AB	2/5
AD	3/5
BD	2/5
DE	2/5

Iteration 3:

Itemset
ABD

data scan ⟶

Itemset	Support
ABD	2/5

infrequent itemsets discarded ⟶

Itemset	Support
ABD	2/5

Figure 1.2: Execution of *Apriori* on table $\mathcal{D}$ for $\mathsf{minsupp} = 2/5$

efficiently. On data containing longer transactions, however, as in the case of census data, *Apriori* performs less well [Bay98] because of the necessity of performing a number of database passes equal to the size of the largest frequent itemset. In Chapter 2 we introduce an algorithm that reduces by half the number of passes necessary to discover the frequent itemsets.

The generation of association rules from the frequent itemsets is performed by algorithm *Apriori* using the procedure *ap-genrules*. The procedure *ap-genrules* improves on the simple procedure mentioned in Section 1.2.2.2 by making careful use of Theorem 1.1. Consider that we have the frequent itemset I and that we check whether the rule that uses $I_a \subset I$ as an antecedent has minimum confidence. The confidence of this rule is $\frac{\mathsf{supp}(I)}{\mathsf{supp}(I_a)}$. If the rule does not have minimum confidence, then for all $I'_a \subset I_a$, the rules $I'_a \to I - I'_a$ will also lack minimum confidence because $\mathsf{supp}(I'_a) \geq \mathsf{supp}(I_a)$. We can build on this observation by remarking that we can also say that if I is an itemset and the rule $I - I_c \to I_c$ with $I_c \subset I$ has minimum confidence, then all rules of the form $I - I'_c \to I'_c$ with $I'_c \subset I_c$ will also have minimum confidence. This happens because we have $I - I_c \subset I - I'_c$, so $\mathsf{supp}(I - I_c) \geq \mathsf{supp}(I - I'_c)$ and we get:

$$\mathsf{conf}(I - I'_c \to I'_c) = \frac{\mathsf{supp}(I)}{\mathsf{supp}(I - I'_c)} \geq \frac{\mathsf{supp}(I)}{\mathsf{supp}(I - I_c)} = \mathsf{conf}(I - I_c \to I_c).$$

Based on this observation, *ap-genrules* works in *Apriori*-like fashion by starting with the set of all 1-itemset possible consequents and then by generating new candidate consequents using the procedure *apriori-gen*. Algorithm 1.3 presents the pseudocode of *ap-genrules*.

Algorithm 1.3 ***(ap-genrules)***

Input: *itemset I and set of candidate k-itemset consequents $\mathcal{K}$*

Output: *the list of rules involving the items of I and having the size of the consequent larger than or equal to k*

Uses: *list of association rules $\mathcal{R}$, list of candidate $(k+1)$-itemset consequents $\mathcal{C}$*

```
1 If $|I| \leq k$, then return $\mathcal{R}$.
2 For each itemset $I_c$ from $\mathcal{K}$, if $\mathsf{conf}(I - I_c \to I_c) \geq \mathsf{minconf}$, then add $I - I_c \to I_c$ to $\mathcal{R}$, else remove $I_c$ from $\mathcal{K}$.
3 Add to $\mathcal{C}$ the itemsets generated by the procedure apriori-gen when applied to $\mathcal{K}$.
```

```
4 Recursively call ap-genrules for parameters $I$ and $\mathcal{C}$ and add the returned rules to $\mathcal{R}$.
5 Return $\mathcal{R}$.
```

■

The pseudocode of the algorithm that uses *ap-genrules* to generate all association rules is shown by Algorithm 1.4[5].

Algorithm 1.4 *(generation of all association rules)*

Input: *set of frequent itemsets* $\mathcal{F}$

Output: *the list of all association rules*

Uses: *list of rules* $\mathcal{R}$*, list of candidate* 1*-itemset consequents* $\mathcal{C}$

```
1 For each frequent itemset $I$ from $\mathcal{F}$, do:
    1.1 Fill $\mathcal{C}$ with all 1-itemsets included in $I$.
    1.2 Call ap-genrules for parameters $I$ and $\mathcal{C}$ and add the returned rules to $\mathcal{R}$.
2 Return $\mathcal{R}$.
```

■

The procedure *ap-genrules* is efficient. The only drawback existing with this part of the algorithm is the possible generation of too many rules, which is not in fact the algorithm's fault. The problem is that the confidence of a rule is not a

[5] The rule generation algorithm presented in [AS94a] is incomplete because it does not generate the rules with consequents of size 1. The algorithms that we present here — Algorithm 1.3 and Algorithm 1.4 — correct this problem.

very good measure of the rule's value, which leads to the difficulty of finding the interesting rules among the many others that are of little interest. For example, if we consider the database $\mathcal{D}$, item D appears in ever transaction, so every rule that has D as a consequent will have minimum confidence, and in fact will have confidence 1! We will discuss this topic in more detail in Chapter 3, where we will present an algorithm for extracting a subset of all association rules, called cover, from which all association rules can be inferred by knowing the support of the frequent itemsets.

1.2.4 Extensions of the association rules problem

The problem of mining association rules can be extended in several ways. One extension regards the generation of rules that include negations. Such rules would take into account not only the presence of an item in a transaction, but also its absence. Taxonomies, introduced in [AS95a], can be regarded as a way of transforming association rules from implication rules involving conjunctions of predicates to implication rules that involve conjunctions of disjunctions of predicates. Such rules could express a larger range of knowledge than simple associations can. Quantitative rules were introduced in [AS96a] by considering not only whether an item has been bought, but also in what quantity it has been bought.

1.3 Functional and purity dependencies

Association rules involve items, which in the case of non-binary databases represent attribute–value pairs, as shown in Section 1.2.1. On the other hand, functional dependencies represent exact implication rules involving attributes. Given three attributes X, Y, and Z , the functional dependency $XY \rightarrow Z$ tells us that the values of attributes X and Y determine the value of attribute Z. For example, in a database of US customer addresses, we have the functional dependency *zipcode* → *state* because the state can be inferred from the zipcode. Functional dependencies are a central topic in the theory of relational databases [ST95]. The following rules permit the inference of functional dependencies and are known as Armstrong's rules.

Definition 1.3 ***Armstrong inference rules***

Let U, V, W be sets of attributes. Then we have the rules of inference:

- $\dfrac{V \subseteq U}{U \to V}$ *(Inclusion)*
- $\dfrac{U \to V}{UW \to VW}$ *(Augmentation)*
- $\dfrac{U \to V,\, V \to W}{U \to W}$ *(Transitivity)*

□

Functional dependencies represent exact implications, but it is also interesting to find and be able to quantify approximate dependencies. In Chapter 5 we define such dependencies using the concept of generalized entropy and we name them purity dependencies. We show that purity dependencies satisfy properties similar to the Armstrong rules, so they can be regarded indeed as generalizations of functional dependencies.

1.4 Thesis organization

In the next chapters we will present our contributions to the problems of mining association rules and generalizing functional dependencies. Each chapter will begin with a presentation of the problem and of current results, and then it will continue with the presentation of our contribution.

In Chapter 2 we examine the problem of mining frequent itemsets in a single-table database using the concept of Galois connections, and we introduce two algorithms, among which *Closure* offers a performance improvement over *Apriori*. We continue in Chapter 3 by addressing the problem of generating association rules. We introduce a new rule of inference for association rules, and based on it we define a cover for the set of association rules. We present algorithm *CoverRules* for computing such a cover, and our experimental results show that the size of the cover is usually one or more orders of magnitude smaller than the total number of association rules, and that covers can be computed efficiently with little or no overhead compared to the classic *Apriori* method. The problem of mining association rules in a multiple-table database is addressed in Chapter 4, where we present two algorithms for mining association rules from a database organized in a star schema. We conclude that chapter with a discussion of the inherent difficulty of mining complex database schemas. In Chapter 5 we present the concept of purity dependency, which is defined using the notion of generalized entropy. Purity dependencies can be regarded as generalized or approximate functional dependencies, and they satisfy properties similar to the Armstrong rules [ST95] that are satisfied by functional dependencies. The final chapter concludes our thesis with an overview of our results and suggestions for possible future research directions.

CHAPTER 2

Mining frequent itemsets in single-table databases

The introduction of the association rules problem in [AIS93b] and [AS94b] has stimulated research for new solutions. The *Apriori* algorithm had been designed primarily for the mining of association rules from sparse databases, which, for the purpose of our discussion, we define as being databases where the number and size of the large itemsets is relatively small. [Bay98] pointed out that *Apriori* does not perform well on databases that are dense, as is the case for census data. In [BMU97], the authors mentioned that when they applied their algorithm (*DIC* — an optimization of *Apriori* that we will discuss later) to census data, mining was feasible only for high values of minsupp, even after the authors removed the items that appeared in more than 80% of the transactions. *Apriori* does not handle dense databases well; this happens because, when the size of the largest frequent itemset is N, *Apriori* will require N passes over the data to discover that itemset. For dense databases, this N can be relatively large, leading to increased I/O costs and thus, to a degradation of *Apriori*'s performance. A large value of N also implies a large number of frequent itemsets, because all 2^N subsets of a frequent N-itemset are frequent too. Any algorithm would have to deal with this exponential complexity, but *Apriori* will also have to deal with a further complexity increase due to the generation of non frequent candidate itemsets. To deal with these problems, a significant number of methods have been proposed. We briefly discuss some of the most important results.

The *Partition* algorithm has been introduced in [SON95] and its improvement consists of requiring only 2 passes over the data. To achieve this, the *Partition* algorithm partitions the database horizontally into blocks that can be loaded into main memory. In the first pass over the database, the algorithm loads each block in memory, where it is mined using an *Apriori*-like method, resulting in a number of itemsets that are locally frequent in that block. After this step, *Partition* has gathered a collection of itemsets that are frequent in at least one block. The support of these itemsets with respect to the entire database is computed in a second scan, which discovers all frequent itemsets. *Partition* makes use of the simple observation that an itemset that is frequent must be frequent in at least one block. The drawback of *Partition* is that for large databases, the number of frequent itemsets obtained from each block may be large and the locally frequent itemsets may not have significant overlap. This can lead to problems in both storing all locally frequent itemsets and in computing their support during the second scan of the data. Experimental results show that *Partition* performs better than *Apriori* for small values of minsupp, but that *Apriori* could still outperform *Partition* for larger values of minsupp.

[Mue95] investigated sequential and parallel variants of *Apriori* that used a prefix tree structure instead of the hash-tree mentioned in [AS95a]. In [GKM97], the AR problem is analyzed in relation to the hypergraph transversal problem, and upper bounds on the complexity of two algorithms are provided. The idea of using a vertical organization of the data, and algorithms that perform a bottom-up or top-down search in the lattice of itemsets, were investigated in [ZPO97a] and [ZPO97b].

The *DIC* (Dynamic Itemset Counting) algorithm was analyzed in [BMU97]. *DIC* also tries to reduce the required number of passes over the data. To this purpose, *DIC* compresses into one database scan the actions that *Apriori* took during different scans of the data. More explicitly, after each scan of M ($\mathsf{minsupp} \cdot |\mathcal{T}| < M < |\mathcal{T}|$) rows of the database, *DIC* examines the candidate itemsets to see if any of them has been counted at least $\mathsf{minsupp} \cdot |\mathcal{T}|$ times. If such itemsets are found, *DIC* uses them to try to generate new candidate itemsets for which to compute the support in parallel with the current candidates. This leads to a reduction in the number of scans necessary, because the algorithm starts counting the support of new candidate itemsets as soon as possible, that is, after each block of M rows are scanned, whereas *Apriori* starts counting the support of new candidates only when it starts a new database scan. The choice of M depends on the database characteristics and influences the performance of the algorithm. The experimental results presented in [BMU97] showed that *DIC* performs indeed faster than *Apriori*.

A theoretical analysis of the AR problem using the theory of formal concept analysis is presented in [ZO98]. [Bay98] deals with the problem of mining dense databases by proposing algorithm *Max-Miner*, which only searches for the large frequent itemsets. The set of large frequent itemsets represents a concise description of all frequent itemsets and is thus useful when the mining of all frequent itemsets becomes unfeasible. Experiments showed that *Max-Miner* runs faster than *Apriori* and can find the large itemsets for values of $\mathsf{minsupp}$ for which *Apriori* runs out of memory [Bay98].

Independently of the research that we present in this chapter, and with different results, the application of Galois connections to the AR problem has been investigated in [PBT99b] and [PBT99c], resulting in the introduction of two new algorithms: *A-Close* and *Close*. These algorithms improve upon *Apriori* by searching for a subset of the set of frequent itemsets represented by the set of frequent closed[1] itemsets. For many databases, the number of frequent closed itemsets is considerably smaller than the number of frequent itemsets. Thus, searching only for closed itemsets usually allows *A-Close* and *Close* to perform fewer database passes and to compute the support for a smaller number of itemsets.

The results that we present in this chapter were obtained in the fall of 1999 and published in [CCS00]. A number of other algorithms have appeared since then, and we briefly mention them next. *ChARM* ([ZH99], [ZH02]) also uses the concept of closed itemsets and claims improved performance over *A-Close* and *Close*. *FPgrowth* [HPY00] uses two scans over the data to compress them in main memory using a structure called an *FP-tree*. *FPgrowth* then recursively mines this structure without having to generate candidate itemsets. Both *Tree Projection* ([AAP00], [AAP01]) and *Partial-Support Tree* ([GCL00], [CGL01]) make use of prefix tree structures to compute the support of itemsets using different methods.

2.1 Galois connections

Galois connections are algebraic constructions that play an important role in lattice theory (see [Bir73] and [GHK80]), in universal algebras, and in computer science. We demonstrate their usefulness as an algebraic and conceptual tool in the analysis of the properties of itemsets. We begin by defining the concepts that we will work with in the remainder of the chapter.

[1] The concept of a closed itemset will be defined later in this chapter.

Definition 2.1 *Let $(P, \leq)$ and $(Q, \leq)$ be two partially ordered sets. A* ***Galois connection*** *between P and Q is a pair of mappings (Φ, Ψ) such that $\Phi : P \longrightarrow Q$, $\Psi : Q \longrightarrow P$ and:*

$$x \leq x' \text{ implies } \Phi(x) \geq \Phi(x'),$$
$$y \leq y' \text{ implies } \Psi(y) \geq \Psi(y'),$$
$$x \leq \Psi(\Phi(x)) \text{ and } y \leq \Phi(\Psi(y)),$$

for $x, x' \in P$ and $y, y' \in Q$. □

It is easy to verify (see [Bir73]) that $\Phi(\Psi(\Phi(x))) = \Phi(x)$ and $\Psi(\Phi(\Psi(y))) = \Psi(y)$ for $x \in P$ and $y \in Q$.

Definition 2.2 *Let $(P, \leq)$ be a complete lattice, that is, a poset such that for any $K \subseteq P$ there exist both $\sup K$ and $\inf K$. A* ***closure*** *is a mapping $\mathsf{cl} : P \longrightarrow P$ such that the following conditions are satisfied:*

1. *$x \leq \mathsf{cl}(x)$,*
2. *$\mathsf{cl}(x) = \mathsf{cl}(\mathsf{cl}(x))$ and*
3. *if $x \leq x'$, then $\mathsf{cl}(x) \leq \mathsf{cl}(x')$,*

for all $x, x' \in P$. An element $x \in P$ is **cl-*closed*** *if $\mathsf{cl}(x) = x$. The set of cl-closed elements will be denoted by C_{cl}.* □

For any Galois connection $\mathsf{c} = (\Phi, \Psi)$ between the complete lattices $(P, \leq)$ and $(Q, \leq)$, the mapping $\mathsf{cl}_{\mathsf{c}} = \Psi\Phi$ is a closure on P, and the mapping $\mathsf{cl}'_{\mathsf{c}} = \Phi\Psi$ is a closure on Q. It is easy to see that, in this case, the mapping $\beta_{\mathsf{c}} = \Phi \rceil C_{\mathsf{cl}_{\mathsf{c}}}$ is a bijection between $C_{\mathsf{cl}_{\mathsf{c}}}$ and $C_{\mathsf{cl}'_{\mathsf{c}}}$.

A standard method for generating Galois connections is through the notion of **polarity**.

Definition 2.3 *Let X, Y be two sets and let $R \subseteq X \times Y$ be a relation. Define the mappings $\Phi : \mathcal{P}(X) \longrightarrow \mathcal{P}(Y)$ and $\Psi : \mathcal{P}(Y) \longrightarrow \mathcal{P}(X)$ by*

$$\Phi(K) = \{y \mid y \in Y, (x, y) \in R \textit{ for all } x \in K\},$$

for $K \subseteq X$ and

$$\Psi(H) = \{x \mid x \in X, (x, y) \in R \textit{ for all } y \in H\},$$

for $H \subseteq Y$. The pair $\mathsf{c} = (\Phi, \Psi)$ introduced above is a Galois connection and is usually referred to as ***the polarity on** X **and** Y **determined by the relation*** *R.* □

The following basic properties hold for any polarity $\mathsf{c} = (\Phi, \Psi)$, $K_0, K_1 \subseteq X$, and $H_0, H_1 \subseteq Y$:

1. $\Phi(K_0 \cup K_1) = \Phi(K_0) \cap \Phi(K_1)$
2. $\Phi(K_0 \cap K_1) \supseteq \Phi(K_0) \cup \Phi(K_1)$
3. $\Psi(H_0 \cup H_1) = \Psi(H_0) \cap \Psi(H_1)$
4. $\Psi(H_0 \cap H_1) \supseteq \Psi(H_0) \cup \Psi(H_1)$

We are now ready to define a set of measures and prove a number of interesting properties about them[2].

[2] The relevance of the next results to the topic of mining association rules will become apparent if the sets X and Y are considered to be the set of items $\mathcal{I}$ and the set of transactions $\mathcal{T}$, respectively. The results that we prove here apply to any polarity, so we use the generic notation X and Y instead of using the more specific context of $\mathcal{I}$ and $\mathcal{T}$.

Definition 2.4 *Let $\mathsf{c} = (\Phi, \Psi)$ be a polarity on the sets X and Y.*

*The **semidistances** generated by c are the mappings $d_{\mathsf{c}} : \mathcal{P}(X) \times \mathcal{P}(X) \longrightarrow \mathbb{N}$ and $d'_{\mathsf{c}} : \mathcal{P}(Y) \times \mathcal{P}(Y) \longrightarrow \mathbb{N}$ defined by $d_{\mathsf{c}}(U_0, U_1) = |\Phi(U_0) \oplus \Phi(U_1)|$ for $U_0, U_1 \in \mathcal{P}(X)$, where $\oplus$ is the symmetric difference operation, and $d'_{\mathsf{c}}(V_0, V_1) = |\Psi(V_0) \oplus \Psi(V_1)|$ for $V_0, V_1 \in Y$.*

*The **proximities** generated by c are the mappings $p_{\mathsf{c}} : \mathcal{P}(X) \times \mathcal{P}(X) \longrightarrow \mathbb{N}$ and $p'_{\mathsf{c}} : \mathcal{P}(Y) \times \mathcal{P}(Y) \longrightarrow \mathbb{N}$ defined by $p_{\mathsf{c}}(U_0, U_1) = |\Phi(U_0) \cap \Phi(U_1)|$ for $U_0, U_1 \in \mathcal{P}(X)$, and $p'_{\mathsf{c}}(V_0, V_1) = |\Psi(V_0) \cap \Psi(V_1)|$ for $V_0, V_1 \in Y$.*

*The **weight functions** generated by c are the mappings $w_{\mathsf{c}} : \mathcal{P}(X) \longrightarrow \mathbb{N}$ and $w'_{\mathsf{c}} : \mathcal{P}(Y) \longrightarrow \mathbb{N}$ given by $w_{\mathsf{c}}(U) = |\Phi(U)|$ and $w_{\mathsf{c}}(V) = |\Psi(V)|$ for every $U \in \mathcal{P}(X)$ and $V \in \mathcal{P}(Y)$.* $\square$

If the Galois connection c is clear from context, then the subscript c may be omitted. Also, if a set consists of only one element ℓ, we may write simply ℓ instead of $\{\ell\}$.

Proposition 2.1 *Let c be a polarity on the sets X and Y. We have*

1. $d_{\mathsf{c}}(U_0, U_0 \cup U_1) + d_{\mathsf{c}}(U_1, U_0 \cup U_1) = d_{\mathsf{c}}(U_0, U_1)$,

2. $p_{\mathsf{c}}(U_0, U_0 \cup U_1) = p_{\mathsf{c}}(U_1, U_0 \cup U_1) = p_{\mathsf{c}}(U_0, U_1)$,

3. $d_{\mathsf{c}}(U_0, \mathsf{cl}_{\mathsf{c}}(U_0)) = 0$,

for every $U_0, U_1 \in \mathcal{P}(X)$.

Proof. The definition of d implies that

$$\begin{aligned}
d_{\mathsf{c}}(U_0, U_0 \cup U_1) &= |\Phi(U_0) \oplus \Phi(U_0 \cup U_1)| \\
&= |\Phi(U_0) \oplus (\Phi(U_0) \cap \Phi(U_1))| \\
&= |\Phi(U_0) - \Phi(U_1)| \\
d_{\mathsf{c}}(U_1, U_0 \cup U_1) &= |\Phi(U_1) \oplus \Phi(U_0 \cup U_1)| \\
&= |\Phi(U_1) \oplus (\Phi(U_0) \cap \Phi(U_1))| \\
&= |\Phi(U_1) - \Phi(U_0)|,
\end{aligned}$$

which imply

$$\begin{aligned}
d_{\mathsf{c}}(U_0, U_1) &= |\Phi(U_0) \oplus \Phi(U_1)| \\
&= |\Phi(U_0) - \Phi(U_1)| + |\Phi(U_1) - \Phi(U_0)| \\
&= d_{\mathsf{c}}(U_0, U_0 \cup U_1) + d_{\mathsf{c}}(U_1, U_0 \cup U_1).
\end{aligned}$$

For the second part of the proposition we note that

$$\begin{aligned}
p_{\mathsf{c}}(U_0, U_0 \cup U_1) &= |\Phi(U_0) \cap \Phi(U_0 \cup U_1)| \\
&= |\Phi(U_0) \cap (\Phi(U_0) \cap \Phi(U_1))| \\
&= |\Phi(U_0) \cap \Phi(U_1)| \\
&= p_{\mathsf{c}}(U_0, U_1)
\end{aligned}$$

for every $U_0, U_1 \in \mathcal{P}(X)$. The proof of $p_{\mathsf{c}}(U_1, U_0 \cup U_1) = p_{\mathsf{c}}(U_0, U_1)$ is entirely similar.

Finally, note that $d_{\mathsf{c}}(U_0, \mathsf{cl}_{\mathsf{c}}(U_0)) = |\Phi(U_0) \oplus \Phi(\Psi(\Phi(U_0)))| = |\Phi(U_0) \oplus \Phi(U_0)| = 0$. ∎

Proposition 2.2 *The weight function w_{c} generated by a polarity $\mathsf{c} = (\Phi, \Psi)$ on X and Y has the following properties:*

1. $\max\{w_{\mathsf{c}}(U_0), w_{\mathsf{c}}(U_1)\} \leq d_{\mathsf{c}}(U_0, U_1) + p_{\mathsf{c}}(U_0, U_1)$,

2. $p_{\mathsf{c}}(U_0, U_1) = w_{\mathsf{c}}(U_0 \cup U_1) \leq \min\{w_{\mathsf{c}}(U_0), w_{\mathsf{c}}(U_1)\}$,

3. $d_{\mathsf{c}}(U_0, U_1) + p_{\mathsf{c}}(U_0, U_1) \leq w_{\mathsf{c}}(U_0 \cap U_1)$,

4. $w_{\mathsf{c}}(U_0) + w_{\mathsf{c}}(U_1) = d_{\mathsf{c}}(U_0, U_1) + 2p_{\mathsf{c}}(U_0, U_1)$,

5. $w_{\mathsf{c}}(U_0) = w_{\mathsf{c}}(\mathsf{cl}_{\mathsf{c}}(U_0))$,

for every $U_0, U_1 \in \mathcal{P}(X)$.

Proof. These properties result immediately from Definition 2.4 and basic properties of set operations.

As an example, we give a proof for the third part of the proposition. Note that:

$$\begin{aligned} d_{\mathsf{c}}(U_0, U_1) + p_{\mathsf{c}}(U_0, U_1) &= |\Phi(U_0) \oplus \Phi(U_1)| + |\Phi(U_0) \cap \Phi(U_1)| \\ &= |\Phi(U_0) \cup \Phi(U_1)| \\ &\leq |\Phi(U_0 \cap U_1)| \\ &= w_{\mathsf{c}}(U_0 \cap U_1). \end{aligned}$$

∎

Proposition 2.3 *The proximity function p_{c} generated by a polarity $\mathsf{c} = (\Phi, \Psi)$ on X and Y has the following properties:*

1. $p_{\mathsf{c}}(U, U) = w_{\mathsf{c}}(U)$

2. $p_{\mathsf{c}}(U_0, U_1) = p_{\mathsf{c}}(U_1, U_0)$

3. $p_c(U_0, U_1) = p_c(\mathit{cl}_c(U_0), \mathit{cl}_c(U_1))$

4. $p_c(U_0, U_2) \geq p_c(U_0, U_1) + p_c(U_1, U_2) - w_c(U_1)$,

for every $U, U_0, U_1, U_2 \in \mathcal{P}(X)$.

Proof. The first three properties result directly from Definition 2.4.

We only provide a proof for the fourth property. By rewriting the fourth property from Proposition 2.2, we obtain:

$$p_c(U_0, U_1) = \frac{w_c(U_0) + w_c(U_1) - d_c(U_0, U_1)}{2}.$$

We can use this formula to rewrite the fourth property of Proposition 2.3 in terms of d_c and w_c:

$$\begin{aligned} \frac{w_c(U_0) + w_c(U_2) - d_c(U_0, U_2)}{2} \geq\ & \frac{w_c(U_0) + w_c(U_1) - d_c(U_0, U_1)}{2} \\ & + \frac{w_c(U_0) + w_c(U_1) - d_c(U_0, U_1)}{2} \\ & - w_c(U_1), \end{aligned}$$

which, after canceling the w_c terms, leads to:

$$d_c(U_0, U_2) \leq d_c(U_0, U_1) + d_c(U_1, U_2),$$

which holds because d_c is a semidistance and respects the triangle inequality. ∎

Proposition 2.4 *Let* $U_0, U_1 \subseteq X$ *and let* c *be a polarity on the sets* X *and* Y. *For every sets* U', U'' *such that* $U_0 \subseteq U' \subseteq \mathit{cl}_c(U_0)$ *and* $U_1 \subseteq U'' \subseteq \mathit{cl}'_c(U_1)$ *we have* $d_c(U', U'') = d_c(\mathit{cl}(U_0), \mathit{cl}(U_1))$.

Proof. Note that if $U_0 \subseteq U' \subseteq \mathsf{cl}_c(U_0)$, then $\Phi(U_0) \supseteq \Phi(U') \supseteq \Phi(\Psi(\Phi(U_0))) = \Phi(U_0)$, so $\Phi(U') = \Phi(U_0)$. This implies $d_c(U', V') = |\Phi(U') \oplus \Phi(V')| = |\Phi(U_0) \oplus \Phi(U_1)| = d_c(U_0, V_0)$. ∎

2.2 Frequent itemsets and closures of itemsets

Starting from the set of transactions $\mathcal{T}$ and the set of items $\mathcal{I}$, we define the relation $R \subseteq \mathcal{T} \times \mathcal{I}$ by $R = \{(t, i) \mid t \in \mathcal{T}, i \in \mathcal{I}, i \in t\}$. The mappings of the polarity determined by the relation R are denoted by $\mathsf{ti}_R : \mathcal{P}(\mathcal{T}) \longrightarrow \mathcal{P}(\mathcal{I})$ and $\mathsf{it}_R : \mathcal{P}(\mathcal{I}) \longrightarrow \mathcal{P}(\mathcal{T})$. If R is clear from context, the subscript will be omitted.

We redefine the notion of support based on the weight measure introduced in Definition 2.4.

Definition 2.5 *Let c be a polarity on the set of transactions $\mathcal{T}$ and the set of items $\mathcal{I}$. The support of an itemset I, $I \subseteq \mathcal{I}$, is the value $\mathsf{supp}_{\mathsf{c}}(I) = w_{\mathsf{c}}(I)/|\mathcal{T}|$.*

*An itemset I is **ϵ-frequent** if $\mathsf{supp}_{\mathsf{c}}(I) \geq \epsilon$, and is **$\epsilon$-maximal** if it is ϵ-frequent and there is no ϵ-frequent set L such that $I \subset L$. An itemset I is **closed** if $I = \mathsf{cl}_{\mathsf{c}}(I)$.* □

The weight of an itemset I is given by

$$w_{\mathsf{c}}(I) = |\mathsf{ti}(I)| = |t \in \mathcal{T} \mid i \in t \text{ for all } i \in \mathcal{I}|.$$

In other words, the weight of the itemset I equals the number of transactions that contain every item $i \in I$.

In view of Proposition 2.2, we have:

Theorem 2.5 *If c is a polarity on the set of transactions $\mathcal{T}$ and the set of items $\mathcal{I}$, then for every two itemsets $I_0, I_1 \subseteq I$ we have:*

1. $\max\{\mathsf{supp}_{\mathsf{c}}(I_0), \mathsf{supp}_{\mathsf{c}}(I_1)\} \leq (d_{\mathsf{c}}(I_0, I_1) + p_{\mathsf{c}}(I_0, I_1))/|\mathcal{T}|$;

2. $\mathsf{supp}_{\mathsf{c}}(I_0 \cup I_1) \leq \min\{\mathsf{supp}_{\mathsf{c}}(I_0), \mathsf{supp}_{\mathsf{c}}(I_1)\}$;

3. $d_c(I_0, I_1) + p_c(I_0, I_1) \leq \mathsf{supp}_c(I_0 \cap I_1) \cdot |\mathcal{T}|$*;*

4. $\mathsf{supp}_c(I_0) + \mathsf{supp}_c(I_1) = (d_c(I_0, I_1) + 2p_c(I_0, I_1)) \cdot |\mathcal{T}|$*;*

5. $\mathsf{supp}_c(I_0) = \mathsf{supp}_c(\mathsf{cl}_c(I_0))$*;*

6. $I_0 \subseteq I_1$ *implies* $\mathsf{supp}_c(I_1) \leq \mathsf{supp}_c(I_0)$*.*

Proof. The arguments for the first five parts result immediately from Definition 2.5 and the corresponding parts of Proposition 2.2. The last part results from part 2. ■

Thus, if I_1 is an ϵ-frequent set of items and $I_0 \subseteq I_1$, then I_0 is also ϵ-frequent (see also Theorem 1.1).

Corollary 2.6 *For every sets of items* I, J *such that* $I \subseteq J \subseteq \mathsf{cl}_c(I)$*, we have* $\mathsf{supp}_c(I) = \mathsf{supp}_c(J)$*.*

Definition 2.6 *Let* c *be a polarity on the set of transactions* $\mathcal{T}$ *and the set of items* $\mathcal{I}$*, and let* $I \subseteq \mathcal{I}$ *be an itemset.*

The ***family of*** (ℓ, ϵ)***-extended closures of*** I ***relative to*** c *is the collection of sets*

$$\mathsf{XCL}_c^{\ell,\epsilon}(I) = \{\mathsf{cl}_c(I) \cup J \mid |J| = \ell, J \cap \mathsf{cl}_c(I) = \emptyset, \text{ and } \mathsf{supp}(I \cup J) \geq \epsilon\}.$$

□

Note that

$$\mathsf{XCL}_{\mathsf{c}}^{0,\epsilon}(I) = \begin{cases} \emptyset & \text{if } \mathsf{supp}(I) < \epsilon, \\ \{\mathsf{cl}_{\mathsf{c}}(I)\}, & \text{otherwise.} \end{cases}$$

Further,

$$\mathsf{XCL}_{\mathsf{c}}^{1,\epsilon}(I) = \{\mathsf{cl}_{\mathsf{c}}(I) \cup \{i\} \mid i \in \mathcal{I},\ i \notin \mathsf{cl}_{\mathsf{c}}(I) \text{ and } \mathsf{supp}(I \cup \{i\}) \geq \epsilon\}.$$

We refer to any member of $\mathsf{XCL}_{\mathsf{c}}^{\ell,\epsilon}(I)$ as an (ℓ, ϵ)-**extended closure**. Unless stated otherwise, we always work with $(1, \epsilon)$-extended closures; so, we will just refer to them as **extended closures**.

To compute the family of extended closures for a set of itemsets K, we define the K-matrix M_K. Every row of this matrix corresponds to an itemset from K. We assume that the set of items is $\mathcal{I} = \{i_1, \ldots, i_n\}$ and that the itemsets of K are arranged in lexicographical order. The columns of the matrix M_K correspond to the items of $\mathcal{I}$. If I is an itemset from K and $i_p \in \mathcal{I}$ is some item, then $M_K(I, i_p)$, the element located in the I row and p-th column, will have the value:

$$M_K(I, i_p) = p_{\mathsf{c}}(I, \{i_p\}) = |\{t \in \mathcal{T} \mid I \cup \{i_p\} \subseteq t\}|.$$

Note that the largest value among the entries of the I-line of the matrix M_K will be found in the columns that correspond to the members of $\mathsf{cl}_{\mathsf{c}}(I)$, as shown by the following theorem.

Theorem 2.7 *Let $\mathsf{c} = (\mathsf{ti}_R, \mathsf{it}_R)$ be the polarity associated to the relation $R \subseteq \mathcal{T} \times \mathcal{I}$ and let I be a k-itemset. We have $i \in \mathsf{cl}_{\mathsf{c}}(I)$ if and only if $M_{\{I\}}(I, i) = \max\{M_{\{I\}}(I, j) \mid j \in \mathcal{I}\}$.*

Proof. Note that $\max\{M_{\{I\}}(I, j) \mid j \in \mathcal{I}\} = \mathsf{supp}_{\mathsf{c}}(I)$. For any $i \in \mathcal{I}$ such that $M_{\{I\}}(I, i) = \mathsf{supp}_{\mathsf{c}}(I)$ and $i \notin \mathcal{I}$ we have to prove that $i \in \mathsf{cl}_{\mathsf{c}}(I)$. This results from the definition of the value $M_{\{I\}}(I, i)$, which tells us that i appears in all transactions in which I appears, so i belongs to the closure of the itemset I. ∎

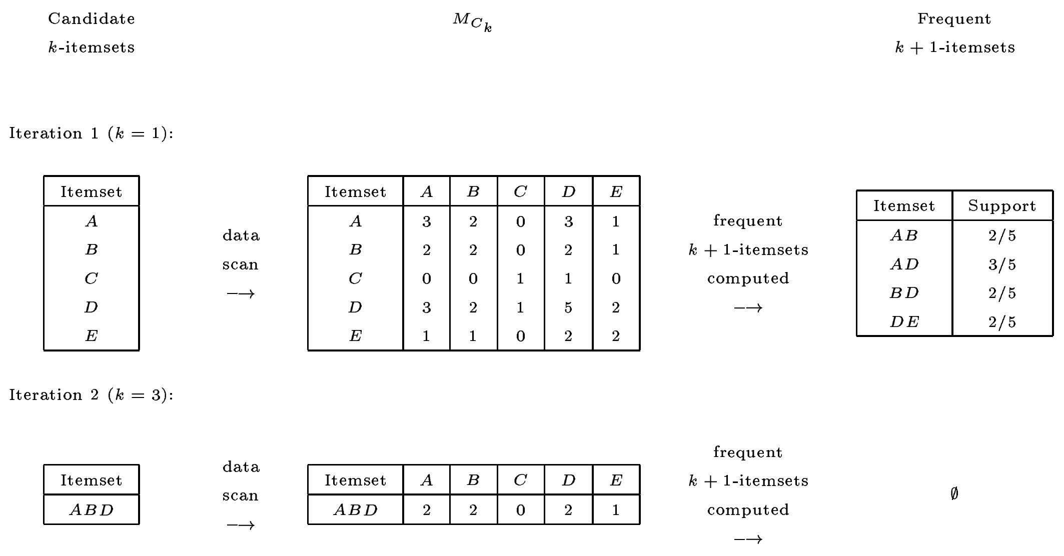
2.3 The *Closure* algorithm

The *Closure*[3] algorithm is a variation of the *Apriori* algorithm and performs fewer database scans than *Apriori*. *Closure* is based on the observation that, if we compute the matrix M_{C_k} for the set of candidate k-itemsets C_k, then we can use it to determine the frequent $k+1$-itemsets, skipping the $k+1$-st step of *Apriori*, and moving directly to the generation of candidate $k+2$-itemsets. Thus, *Closure* requires roughly $\frac{N}{2}$ scans of the database, where N is the size of the largest frequent itemset. Determining the frequent $k+1$-itemsets from M_{C_k} is easy, because for each candidate itemset, the matrix allows us to determine the weight of the itemset and also how often each item appears in the transactions of $\mathcal{T}$ that contain the itemset.

Figure 2.1 shows the execution of *Closure* on our sample database $\mathcal{D}$, introduced in Section 1.2.3. Note that M_{C_1} is symmetric and that it allows us to determine the support of all 1-itemsets and also the support for the 2-itemsets. Analyzing M_{C_1} results in the discovery of four frequent 2-itemsets, out of which, the procedure *apriori-gen* can generate only one candidate. Examination of M_{C_3} fails to discover any frequent 4-itemset, so the algorithm terminates after two data scans, compared to the three scans required by *Apriori*.

The pseudocode for *Closure* is given by Algorithm 2.1. The difference with respect to the pseudocode of *Apriori* appears in step 2, which computes the matrix M_{C_k}, and in step 3, where the matrix is analyzed and the frequent $k+1$-itemsets are determined. Step 3 performs the actions that required two separate database scans in the *Apriori* algorithm.

[3] The *Closure* algorithm that we present here is an improved version of the algorithm described in [CCS00]. The current version of the algorithm is both simpler and more efficient than the original and a public implementation is available in ARtool [Cri02]. ARtool is a free Java application for mining association rules that is distributed under the GNU General Public License.



Iteration 1 ($k = 1$):

Itemset
A
B
C
D
E

Itemset	A	B	C	D	E
A	3	2	0	3	1
B	2	2	0	2	1
C	0	0	1	1	0
D	3	2	1	5	2
E	1	1	0	2	2

Itemset	Support
AB	2/5
AD	3/5
BD	2/5
DE	2/5

Iteration 2 ($k = 3$):

Itemset
ABD

Itemset	A	B	C	D	E
ABD	2	2	0	2	1

Figure 2.1: Execution of *Closure* on table $\mathcal{D}$ for minsupp $= 2/5$

Algorithm 2.1 ***(Closure)***

Input: *the transactions $\mathcal{T}$ containing items $\mathcal{I}$, and minsupp.*

Output: *the list of frequent itemsets.*

Uses: *list of candidates $\mathcal{C}$, list of frequent itemsets $\mathcal{F}$, list of frequent k-itemsets $\mathcal{K}$.*

1 Initialize $\mathcal{C}$ to all 1-itemsets.

2 Scan $\mathcal{T}$ and compute the matrix $M_{\mathcal{C}}$.

3 Set the support of each candidate itemset to the ratio of its number of occurrences and $|\mathcal{T}|$. Discard the candidates that have support less than minsupp. Copy all frequent k-itemsets to $\mathcal{F}$. Determine all frequent $k+1$-itemsets using $M_{\mathcal{C}}$ and copy them to $\mathcal{F}$ and $\mathcal{K}$. Clear $\mathcal{C}$.

4 If $k + 1 = |\mathcal{I}|$, then go to step 7.

5 *Add to $\mathcal{C}$ the itemsets generated by the procedure apriori-gen when applied to $\mathcal{K}$. Clear $\mathcal{K}$.*

6 *If $\mathcal{C}$ is not empty, then go to step 2.*

7 *Return $\mathcal{F}$.*

■

Closure represents an improvement over *Apriori* because, during step 3, it directly generates the $k + 1$-itemsets without the cost of accessing the data or computing the supports for any infrequent itemsets. The only drawback is that, during step 2, the algorithm has to compute the matrix M_{C_k} which also uses additional space. The matrix M_{C_k} uses space $|C_k| \cdot |\mathcal{I}|$, but we could actually reduce the space required by having columns only for the items that appear in the itemsets of C_k. Overall, the advantages of *Closure* easily overcome the additional costs of computing and storing the matrix M_{C_k}.

2.4 The *MaxClosure* algorithm

The *MaxClosure* algorithm uses the concept of extended closure to greedily "jump" through the lattice of itemsets so that it can discover quickly the large itemsets. As mentioned in [Bay98], finding all frequent sets is an extremely expensive computation for some databases, so it makes sense to have available a fast algorithm that would discover only the large itemsets (which also tell us what the frequent itemsets are) without finding the support for all frequent candidates. Based on this knowledge, we could guide the mining process to make it efficient to extract only the rules we are interested in.

MaxClosure is a specialization of the *Closure* algorithm. It uses the matrix M for computing the extended closures of itemsets, and it makes use of the fact that if at any point a frequent itemset has no extended closures, then that itemset is a large itemset.

Theorem 2.8 *If for a frequent itemset I we have $\mathsf{XCL}_{\mathsf{c}}^{1,\epsilon}(I) = \emptyset$, then $\mathsf{cl}_{\mathsf{c}}(I)$ is a maximal frequent itemset.*

Proof. Suppose that $\mathsf{cl}_{\mathsf{c}}(I)$ is not a large itemset and that its family of extended closures is empty. Then, there exists a frequent itemset G such that $\mathsf{cl}_{\mathsf{c}}(I) \subset G$ and therefore, $G - \mathsf{cl}_{\mathsf{c}}(I) \neq \emptyset$. Let i be an item such that $i \in G$ and $i \notin \mathsf{cl}_{\mathsf{c}}(I)$. Because G is frequent, it follows that i is frequent and appears with $\mathsf{cl}_{\mathsf{c}}(I)$ in a fraction of transactions greater than ϵ. Then, because i does not belong to the closure of I, the family of extended closures of I is not empty and we have a contradiction. ▮

The pseudocode of *MaxClosure* is shown in Algorithm 2.2.

Algorithm 2.2 ***(MaxClosure)***

Input: *the transactions $\mathcal{T}$ containing items $\mathcal{I}$, and $\mathsf{minsupp}$.*

Output: *the list of large itemsets.*

Uses: *list of candidates $\mathcal{C}$, list of frequent itemsets $\mathcal{F}$, list of large itemsets $\mathcal{L}$.*

```
1 Initialize $\mathcal{C}$ to all 1-itemsets.

2 Scan $\mathcal{T}$ and compute the matrix $M_{\mathcal{C}}$.

3 For each frequent itemset from $\mathcal{C}$, compute its extended closures. If there are
  no extended closures, set the support of the itemset to the ratio of its number
  of occurrences and $|\mathcal{T}|$, and add the candidate to $\mathcal{L}$ if it is not already in $\mathcal{L}$;
  otherwise add all extended closures to $\mathcal{F}$ if they are not already in $\mathcal{F}$.
```

4 Clear $\mathcal{C}$.

4 Eliminate all itemsets from $\mathcal{F}$ that are included in an itemset from $\mathcal{L}$.*

5 If $\mathcal{F}$ is not empty, then copy $\mathcal{F}$ into $\mathcal{C}$, empty $\mathcal{F}$, and go to step 2.

6 Return $\mathcal{L}$.

■

We analyze the execution of *MaxClosure* on database $\mathcal{D}$. Initially, we start with the set of candidate itemsets $\mathcal{C} = \{A, B, C, D, E\}$. The matrix $M_{\mathcal{C}}$ for this step is presented in Table 2.1. Using $M_{\mathcal{C}}$, the closure of A is determined to be AD, and its extended closure is ABD, with support $\frac{2}{5}$. We add ABD to $\mathcal{F}$, and move to the next candidate. In the case of B, its closure is ABD, and there are no extended closures, so we add ABD to $\mathcal{L}$. C is not frequent. For D, the closure is D, and the extended closures are AD, BD, and DE, which we add to $\mathcal{F}$. Finally, for E the closure is DE, and there are no extended closures, so we add DE to $\mathcal{L}$. At this point we have $\mathcal{L} = \{ABD, DE\}$, and $\mathcal{F} = \{ABD, AD, BD, DE\}$. During Step 4* all itemsets from $\mathcal{F}$ are removed and $\mathcal{L}$ is returned.

Itemset	A	B	C	D	E
A	3	2	0	3	1
B	2	2	0	2	1
C	0	0	1	1	0
D	3	2	1	5	2
E	1	1	0	2	2

Table 2.1: Matrix $M_{\mathcal{C}}$ for step 1 of *MaxClosure*

We justify the correctness of Step 4* of Algorithm 2.2. Suppose that during some iteration of *MaxClosure* we examine itemset I from $\mathcal{C}$ and determine that one of its extended closures is $I' = \mathsf{cl_c}(I) \cup i$, for some item i. We also find in $\mathcal{L}$ a large itemset I'_l, such that $I' \subseteq I'_l$. If I' is large, then it must be equal to I'_l and we can safely eliminate it, so from now on we can assume that $I' \subset I'_l$. Note that the large itemset I'_l must have been added to $\mathcal{L}$ during this iteration of the algorithm, otherwise I would have been eliminated during the precedent iteration. Let I_l be the itemset from whose examination we added I'_l to $\mathcal{L}$. We must have $I'_l = \mathsf{cl_c}(I_l)$, otherwise I'_l would not have been added to $\mathcal{L}$ during this step. To summarize, we have $I' = \mathsf{cl_c}(I) \cup i \subset \mathsf{cl_c}(I_l) = I'_l$. It is incorrect to remove I' if one of its extended closures would lead to some large itemset I_x which cannot be obtained otherwise. Because $I' \subseteq I'_l$, it results that $\mathsf{cl_c}(I') \subseteq \mathsf{cl_c}(I'_l) = I'_l$, so I_x must contain some item $j \notin I'_l$. But then, $\mathsf{cl_c}(I) \cup j$ would also be an extended closure of I, so I_x can be discovered starting from it and we do not need I'.

In practice, the usefulness of Step 4* is limited because we rarely happen to find the large itemsets in the first steps of the algorithm, as it happened during the execution on $\mathcal{D}$.

The *MaxClosure* algorithm requires N scans of the database in the worst case, where N is the size of the largest frequent itemset. The reason is that, in the worst case, a candidate itemset can be extended with only one item during one scan of the database, so it will take N scans of the database to generate an N-itemset.

2.5 Experimental results

We implemented in C++ the algorithms *Apriori*, *Closure*, and *MaxClosure*. We used the hash-tree structure [AS94b] to efficiently test for itemset inclusions. The data was generated using the IBM Quest synthetic data generator [AS94a] and was stored in binary files. The implementation of algorithm *MaxClosure* omitted Step 4* because we noticed that it did not provide any improvement. On the contrary, the execution of Step 4* seemed to slow down the algorithm for low values of minsupp.

We started by generating two synthetic databases with characteristics similar to the databases `t5i2d100k`[4] and `t10i4d100k`, as they were described in [AS94a]. These databases have 1000 items, 100,000 transactions, and 2000 patterns. `t5i2d100k` has an average of 5 items per transaction and 2 items per pattern, and `t10i4d100k` has an average of 10 items per transaction and 4 items per pattern. Both these databases are sparse and, as can be observed from Table 2.2, *Closure* and *MaxClosure* clearly outperform *Apriori*. *MaxClosure* performs more database scans and, because the number of frequent itemsets is small, the savings obtained from the computation of the support cannot overcome the I/O costs. Thus, *MaxClosure* is a bit slower than *Closure*. As observed by other researchers [Mue95], when working with files the bottleneck is in the CPU computations, not in accessing the disk. The plots of these results are shown in Figure 2.2 and Figure 2.3.

Notice that for `t5i2d100k` and minsupp $\in \{1, 1.5, 2\}$, *MaxClosure* performs fewer database scans than *Apriori*.

For the next experiment, we generated three synthetic databases with 100 items, an average of 10 items per transaction, and 200 patterns with an average of 4 items per pattern. `t10i4d50k` has 50,000 rows, `t10i4100k` has 100,000, and `t10i4d1M` has 1,000,000 rows. These databases are denser than the ones from

[4] We follow the database naming convention from [AS94b] which does not include information about the number of items present in the database.

minsupp (%)	Large itemsets	Time(seconds)/Data scans		
		Apriori	*Closure*	*MaxClosure*
`t5i2d100k`				
0.25	438	790/6	1/3	3/6
0.33	405	671/4	1/2	2/4
0.5	327	390/4	1/2	2/4
0.75	231	45/3	1/2	1/3
1	165	15/2	1/1	1/1
1.5	83	3/2	1/1	1/1
2	32	1/2	1/1	1/1
`t10i4d100k`				
0.25	1559	1055/8	10/4	22/8
0.33	999	1011/8	7/4	14/8
0.5	646	887/8	5/4	7/8
0.75	426	659/3	3/2	3/3
1	362	498/2	3/1	3/2
1.5	232	66/2	3/1	3/1
2	159	18/2	3/1	3/1

Table 2.2: Results for databases with 1000 items

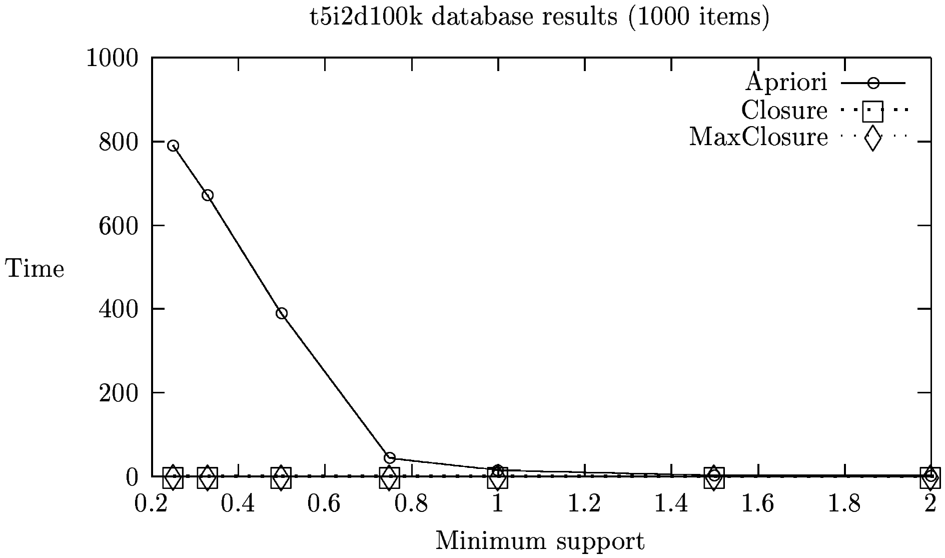


Figure 2.2: Results for `t5i2d100k`

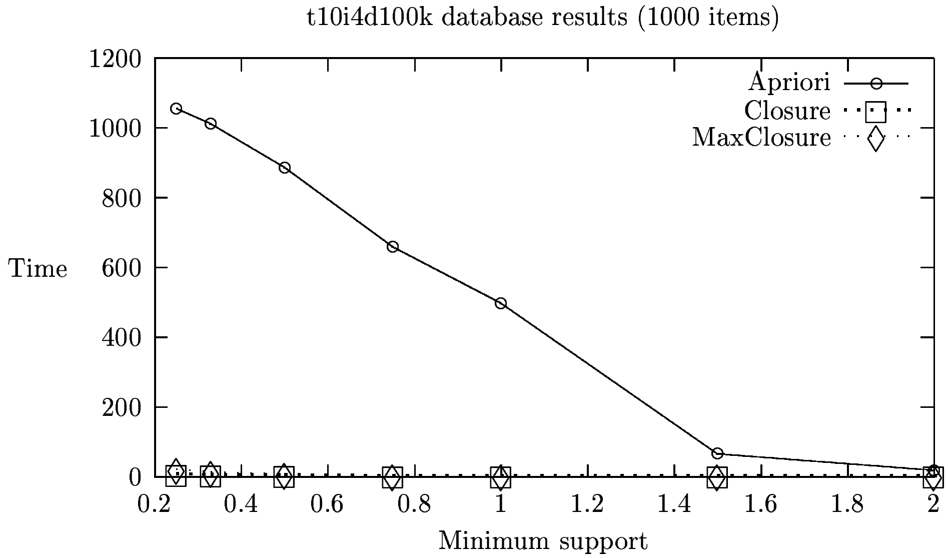


Figure 2.3: Results for `t10i4d100k` (1000 items)

the previous experiment. The results from Table 2.3 show that all algorithms scale linearly with respect to the increase of the size of the data. The plots of the results are shown in Figure 2.4, Figure 2.5, and Figure 2.6. We can notice that *MaxClosure* performs faster than *Closure*, even though it performs more database scans. This happens because of the large number of frequent itemsets, and because *MaxClosure* needs to compute the support for fewer itemsets than *Closure*. As minsupp is increased and the number of frequent itemsets decreases, we can notice that the time taken by *MaxClosure* becomes comparable to the time required by *Closure*. The difference between the performance of *Closure* and *Apriori* is less visible in the case of `t10i4d50k` and `t10i4d100k`, but is more evident for `t10i4d1M` where we can notice differences of up to 400 seconds.

MaxClosure usually performs the same number of steps as *Apriori*, with some exceptions occurring in the experiment involving `t5i2d100k`. If we include Step 4*, the performance of *MaxClosure* degrades for lower values of minsupp. We conclude

minsupp (%)	Large	Time(seconds)/Data scans		
	itemsets	*Apriori*	*Closure*	*MaxClosure*
`t10i4d50k`				
0.5	6410	213/9	205/5	54/9
0.7	3810	83/9	61/5	24/9
0.9	2603	30/8	31/4	16/8
1	2248	26/8	27/4	13/8
2	657	11/7	9/4	6/7
3	320	7/7	4/4	4/7
5	154	4/7	2/4	3/7
`t10i4d100k`				
0.5	6481	394/9	389/5	87/9
0.7	3798	170/9	134/5	45/9
0.9	2689	59/8	62/4	31/8
1	2243	51/8	54/4	26/8
2	651	21/7	18/4	12/7
3	318	14/7	8/4	9/7
5	155	9/7	4/4	6/7
`t10i4d1M`				
0.5	6384	4178/9	3864/5	689/9
0.7	3828	1651/9	1212/5	414/9
0.9	2708	576/8	618/4	297/8
1	2197	510/8	527/4	255/8
2	647	217/7	180/4	128/7
3	316	148/7	91/4	96/7
5	147	98/7	42/4	60/7

Table 2.3: Results for databases with 100 items

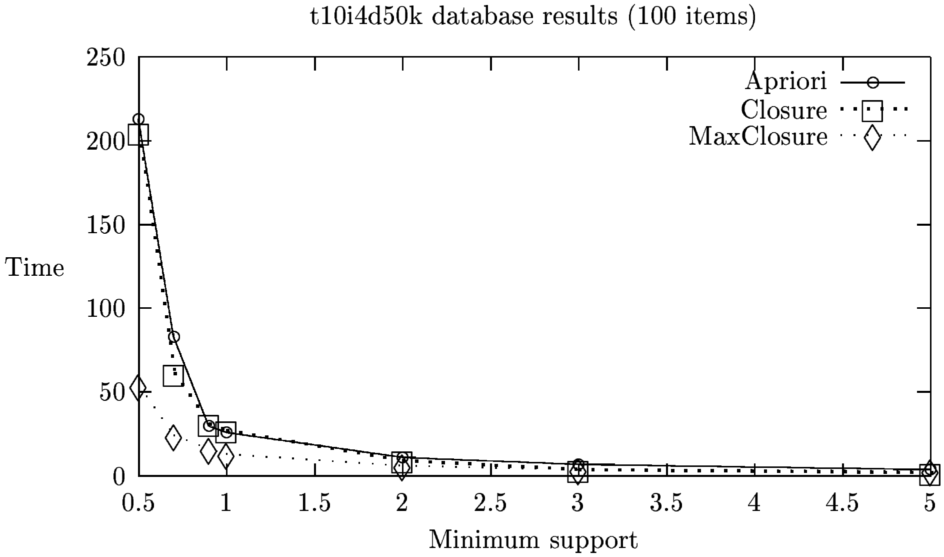


Figure 2.4: Results for `t10i4d50k`

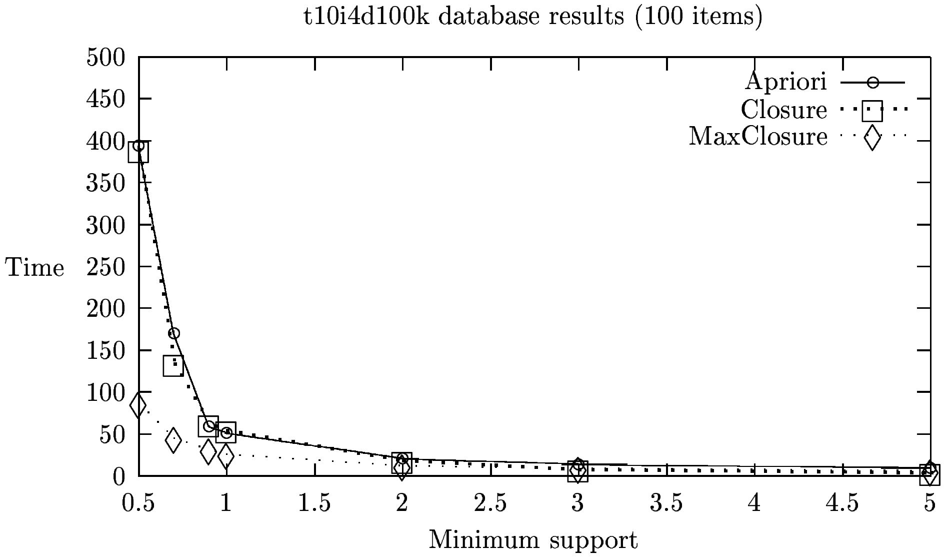


Figure 2.5: Results for `t10i4d100k` (100 items)

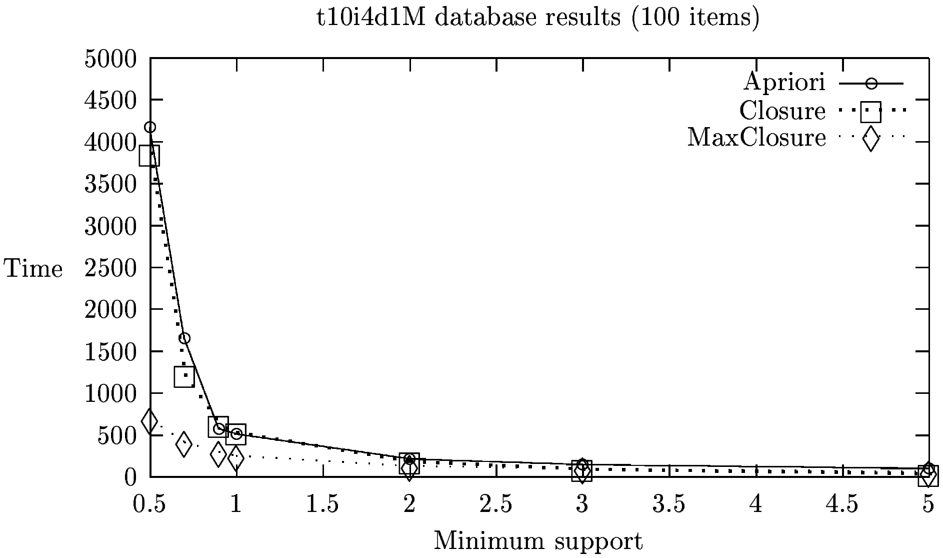


Figure 2.6: Results for `t10i4d1M`

that, on synthetic data, Step 4* is better omitted from the implementation of Algorithm 2.2.

From these and other experiments we have noticed that the performance of *Closure* tends to get closer to that of *Apriori* when the database is denser. *MaxClosure* runs faster than both *Apriori* and *Closure* when the minsupp values are low, and is thus indicated when executing these algorithms becomes unfeasible.

2.6 Conclusions

We introduced a formalization of the association rules problem based on the concept of Galois connections. Based on some of the properties that we proved, we introduced two new algorithms: *Closure*, for mining all frequent itemsets, and *MaxClosure*, for mining the large itemsets. *Closure* performs fewer database passes than *Apriori* and generally outperforms *Apriori*, its performance approaching that of *Apriori* when the data is denser. *MaxClosure* usually performs as many database passes as *Apriori*, but executes faster because it computes the support of a smaller number of itemsets. When the value of minsupp decreases, *MaxClosure* performs better in comparison with *Apriori* and *Closure*. We also noticed that the I/O cost of working with binary files has a restricted impact on the performance of the algorithms, and thus the difference between the execution times of *Apriori* and *Closure* is smaller than if they were run on databases with more costly access time.

CHAPTER 3

Mining association rules in single-table databases

The generation of association rules starting from the set of frequent itemsets can be performed efficiently using the procedure *ap-genrules*, introduced in [AS94a]. The only problem with this method is that, in practice, the number of association rules generated can become too large and can easily overwhelm a human analyst. Several methods have been proposed to address this problem. Measures of interestingness can be used to assess the interestingness of a rule and thus generate only interesting rules or prune non-interesting ones. Various such interestingness measures were discussed in [BA99], [LHM99], and [JS01]; a survey of interestingness measures was made in [HH99]. [SLR99] presented a set of pruning rules for removing semantically redundant association rules. [PT98] and [PT00] introduced the concept of rules that are unexpected with respect to prior beliefs and analyzed the generation of such rules. [PBT99a], [BPT00], and [Zak00] gave various definitions of the concept of redundant rules. Out of these definitions, the one from [PBT99a] is of special interest to us, because in that paper the redundancy of an association rule is determined by whether or not that rule can be inferred from other association rules through the use of inference rules.

Our work ([CS02]) continues the line of research from [PBT99a] by introducing a new rule of inference for association rules and by defining the concept of a cover of the association rules as a minimal set of rules that are non-redundant with respect to this new inference rule. Our concept of cover is different from the rule cover concept defined in [TKR95], which referred to the pruning of sets of association rules with identical consequent and which was not related to the use of inference rules.

We consider that the use of inference rules in the definition of redundant association rules is an important approach because it offers a great potential for minimizing the number of rules generated and also because it provides a general mechanism for computer assisted exploration of the set of association rules. The inference rule that we introduce in this chapter materializes such potential because of its interesting properties and simplicity.

3.1 Rules of inference for association rules

In [PBT99a], association rules were divided into two categories which we define below.

Definition 3.1 *An association rule with confidence* 1 *is called an* ***exact*** *association rule. An association rule that is not exact is called an* ***approximative*** *association rule.* □

An exact association is denoted by $X \Rightarrow Y$. Not all authors require the antecedent and consequent of a rule to be disjoint, as mentioned in Definition 1.1, so this requirement should be ignored in the case of results that we quote from other papers.

Definition 3.2 *An association rule is said to be* ***valid*** *[PBT99c], if its support and confidence are greater or equal to* minsupp *and, respectively,* minconf*.* □

3.1.1 Inference rules and bases for association rules

An important issue in association rule mining is that many times the number of association rules generated is overwhelming for the user of the mining system. One solution for this problem consists of generating only the association rules that are non-redundant in the sense that they cannot be inferred from other rules by using certain rules of inference. A minimal set of such association rules was called **basis** in [PBT99a]. To avoid confusion, we mention here that the single word rule will only be used in the sense of an association rule and will never be used to denote an inference rule.

Based on the results of [GD86] and [Lux91], Pasquier and colleagues [PBT99a] introduced the **Guigues-Duquenne basis** for exact association rules and the **Luxenburger basis** for approximative association rules, which together form a basis for the valid association rules.

The Guigues-Duquenne basis is a minimal set of exact association rules from which the complete set of exact association rules can be inferred using the following two inference rules:

1. $X \Rightarrow Y, W \Rightarrow Z \vdash XW \Rightarrow YZ$

2. $X \Rightarrow Y, Y \Rightarrow Z \vdash X \Rightarrow Z$

The Guigues-Duquenne basis does not allow us to infer the support of the rules and in fact, by ignoring the support values, the first inference rule can lead to association rules that have inferior support compared to the rules used in its generation. Note also that the rules of inference used here are similar to the Armstrong rules of inference for functional dependencies [ST95]. It is easy to verify that the first rule mentioned above is exchangeable for the Armstrong augmentation rule

$$X \Rightarrow Y \vdash XZ \Rightarrow YZ,$$

that is, given one of them we can infer the other.

In Chapter 5 we introduce the concept of purity dependency as a generalization of the concept of functional dependency and prove that this new type of dependency satisfies properties similar to the Armstrong rules. Given that the Armstrong rules for functional dependencies transcribe well to exact association rules, we might expect that the properties of the generalizations of functional dependencies also hold for the approximative association rules. This is not the case, however, because an augmented rule can fail to achieve both minimum support and minimum confidence, and from the two transitivity properties that we prove in Chapter 5, only one of them is holding in the case of association rules, as the next theorem shows.

Theorem 3.1 *Let $X \to Y$ and $Y \Rightarrow Z$ be two valid association rules. Then $X \to Z$ is a valid association rule.*

Proof. We have $\mathsf{supp}(Y) = \mathsf{supp}(YZ)$, so $\mathsf{supp}(XY) = \mathsf{supp}(XYZ) \leq \mathsf{supp}(XZ)$, which implies that the rule $X \to Z$ has minimum support. The confidence of this rule is $\frac{\mathsf{supp}(XZ)}{\mathsf{supp}(X)} \geq \frac{\mathsf{supp}(XY)}{\mathsf{supp}(X)}$ based on the observation we made before. Therefore our rule also has minimum confidence, so it is valid. ∎

The Luxenburger basis is a minimal set of approximative rules from which the complete set of approximative rules can be deduced using the two properties introduced in [Lux91]:

1. the association rule $X \to Y$ has the same support and confidence as the rule $\mathsf{cl}(X) \to \mathsf{cl}(Y)$

2. for any three closed itemsets X, Y, and Z, such that $X \subseteq Y \subseteq Z$, the confidence of the rule $X \rightarrow Z$ is equal to the product of the confidences of the rules $X \rightarrow Y$ and $Y \rightarrow Z$, and its support is equal to the support of the rule $Y \rightarrow Z$

The first property indicates that we only need to generate association rules whose antecedent and consequent are closed itemsets, and the second property allows us to eliminate transitive rules between closed itemsets (see also [Zak00]). Both these properties can be regarded as new inference rules and they permit the inference of both the support and confidence of the resulting rules.

Together, the Guigues-Duquenne basis and the Luxenburger basis, provide a minimal basis for association rules, which we will denote as the **GD-L basis**. Starting from this basis, however, it is possible to infer rules that are not valid, as was noticed in the presentation of the Guigues-Duquenne basis.

Both [BPT00] and [Zak00] have introduced other bases starting from new definitions of redundancy that had a lesser emphasis on the use of inference rules, and emphasized more the format of the association rules, based on the idea of presenting the user with the most informative rules. As a result, these new bases are larger than the GD-L basis.

3.1.2 A new inference rule for association rules

We now introduce a rule of inference for association rules that does not take into account their support and confidence, these being easily computed in a data mining application by using the results obtained during the generation of all frequent itemsets. The reason for this approach is that the inference of association rules with exact support and confidence is not a fundamental requirement because it is hard to imagine the user trying to infer manually other rules than those presented to him. On the other hand, the data mining application already has all the information regarding the support of the frequent itemsets and this information can be used to compute the exact support and confidence of any rule that we know is valid.

Theorem 3.2 *Let r, r' be two association rules such that $\mathit{items}(r') \subseteq \mathit{items}(r)$ and $\mathit{supp}(\mathit{antc}(r')) \leq \mathit{supp}(\mathit{antc}(r))$. Then,*

$$\mathit{supp}(r') \geq \mathit{supp}(r) \text{ and } \mathit{conf}(r') \geq \mathit{conf}(r).$$

Proof. $\mathsf{items}(r') \subseteq \mathsf{items}(r)$, so $\mathsf{supp}(\mathsf{items}(r')) \geq \mathsf{supp}(\mathsf{items}(r))$, and thus, rule r' has greater support. Also, for the confidence of this rule we can write

$$\mathsf{conf}(r') = \frac{\mathsf{supp}(r')}{\mathsf{supp}(\mathsf{antc}(r'))} \geq \frac{\mathsf{supp}(r)}{\mathsf{supp}(\mathsf{antc}(r))} = \mathsf{conf}(r)$$

because $\mathsf{supp}(\mathsf{antc}(r')) \leq \mathsf{supp}(\mathsf{antc}(r))$. ∎

This justifies the introduction of the inference rule:

$$\frac{r, \mathsf{items}(r') \subseteq \mathsf{items}(r), \mathsf{supp}(\mathsf{antc}(r')) \leq \mathsf{supp}(\mathsf{antc}(r))}{r'}.$$

Note that this inference rule is sound because by applying it to a valid association rule we can only derive other valid association rules.

Because this inference rule requires knowledge of the support of the rules and of the support of the antecedents of the association rules, it cannot be used as a purely syntactical inference rule. This is because, although $\mathsf{supp}(\mathsf{antc}(r))$ can be obtained by computing the ratio of the support and confidence of r, we need the additional information about $\mathsf{supp}(\mathsf{antc}(r'))$ before we can apply the inference rule. Examples of purely syntactical rules are the Guigues-Duquenne rules and the rule of Theorem 3.1, because they only rely on knowledge of association rules characteristics (antecedent, consequent, support, and confidence). In this respect, our rule is similar to the Luxenburger rules which can be applied only to association rules whose antecedents and consequents are known to be closed itemsets.

The new inference rule is very powerful because by starting from a valid rule it can allow us to infer many other valid rules, as shown in the next example.

Example 3.1 *Consider the data from Table 3.1 that was used in [PBT99a]. We will examine this table assuming* $\mathsf{minsupp} = \frac{2}{5}$ *and* $\mathsf{minconf} = \frac{1}{2}$.

tid	A	B	C	D	E
1	1	0	1	1	0
2	0	1	1	0	1
3	1	1	1	0	1
4	0	1	0	0	1
5	1	1	1	0	1

Table 3.1: The sample binary table of [PBT99a]

Note that the rule $B \rightarrow ACE$ *is valid and that* B, C, *and* E, *are the most frequent items in the table, so by using Theorem 3.2, we can also infer the valid rules:* $E \rightarrow BCE$, $B \rightarrow E$, $E \rightarrow B$, $AC \rightarrow BE$, *to enumerate just a few. The exact support and confidence of these rules can be computed based on the information we have about the frequent itemsets.* □

Definition 3.3 *If for two association rules,* r_1 *and* r_2, *it is possible to infer* r_2 *from* r_1 *using Theorem 3.2, then we say that rule* r_2 *is* ***covered*** *by rule* r_1 *(and that* r_1 ***covers*** r_2*), and we write* $r_1 \prec r_2$. *The* ***coverage relation*** $\prec$ *consists of all ordered pairs of rules* (r_1, r_2), *such that* $r_1 \prec r_2$. □

Because of Definition 3.3, we will also refer to the property of Theorem 3.2 as the **coverage rule**.

Definition 3.4 *Two association rules, r_1 and r_2, are called* ***equipotent*** *if $r_1 \prec r_2$ and $r_2 \prec r_1$.* □

The next theorem shows that equipotent rules must have the same sets of items and that the support of their antecedents must be equal.

Theorem 3.3 *Let r_1, r_2 be two association rules. Then r_1 and r_2 are equipotent if and only if $\mathsf{items}(r_1) = \mathsf{items}(r_2)$ and $\mathsf{supp}(\mathsf{antc}(r_1)) = \mathsf{supp}(\mathsf{antc}(r_2))$.*

Proof. This statement follows immediately from Definition 3.4. ∎

Example 3.2 *Consider again the data from Table 3.1.*

The rules $B \to ACE$, $C \to ABE$, $E \to ABC$, and $BE \to AC$ are all equipotent because they have the same set of items, and because $\mathsf{supp}(B) = \mathsf{supp}(C) = \mathsf{supp}(E) = \mathsf{supp}(BE) = \frac{4}{5}$. □

The following theorem shows that the coverage relation is a preorder on the set of association rules.

Theorem 3.4 *The coverage relation $\prec$ is reflexive and transitive but is not antisymmetric.*

Proof. Reflexivity results from Theorem 3.2 and the lack of antisymmetry follows from Definition 3.4 and Theorem 3.3.

To prove the transitivity property, let $r_1 = X_1 \to Y_1$, $r_2 = X_2 \to Y_2$, and $r_3 = X_3 \to Y_3$, be three rules such that $r_1 \prec r_2$ and $r_2 \prec r_3$. This means that

$\mathsf{supp}(X_2) \leq \mathsf{supp}(X_1)$ and $\mathsf{supp}(X_3) \leq \mathsf{supp}(X_2)$, which imply that $\mathsf{supp}(X_3) \leq \mathsf{supp}(X_1)$. Also, we have that $X_2Y_2 \subseteq X_1Y_1$ and $X_3Y_3 \subseteq X_2Y_2$, which imply that $X_3Y_3 \subseteq X_1Y_1$. Therefore, because the conditions of Theorem 3.2 are met, it results that $r_1 \prec r_3$. ▮

Equipotent rules are important because they are interchangeable from the point of view of the coverage relation.

Corollary 3.5 *If r_1 and r_2 are equipotent and $r_1 \prec r_3$ for some r_3, then also $r_2 \prec r_3$.*

Proof. Because r_1 and r_2 are equipotent, we have $r_2 \prec r_1$ so, from the hypothesis and the transitivity of $\prec$, we also obtain $r_2 \prec r_3$. ▮

3.2 Covers for association rules

Based on Theorem 3.2, we define the notion of a cover of the set of valid association rules:

Definition 3.5 *Let $\mathcal{R}$ be the set of all valid association rules extracted from a table τ. A **cover** of $\mathcal{R}$ is a minimal set $\mathcal{C} \subseteq \mathcal{R}$, such that any rule from $\mathcal{R} - \mathcal{C}$ is covered by a rule in $\mathcal{C}$. A rule belonging to $\mathcal{C}$ is called a **$\mathcal{C}$-cover rule**.* □

Note that $\mathcal{R}$ does not have to have a unique cover. Because of the existence of equipotent rules, a set of association rules can have several covers.

Proposition 3.6 *Given a cover $\mathcal{C}$ of $\mathcal{R}$, such that one of its rules r_1 is equipotent with a rule r_2, $(\mathcal{C} - \{r_1\}) \cup \{r_2\}$ is another cover of $\mathcal{R}$.*

Proof. The proof is immediate from Corollary 3.5. ▮

Example 3.3 *Consider again the table from Example 3.1.*

If we mine the table using $\mathsf{minsupp} = \frac{2}{5}$ *and* $\mathsf{minconf} = \frac{1}{2}$*, then we obtain 50 association rules which we omit listing here. The GD-L basis contains six rules:* $\{A \Rightarrow C, B \Rightarrow E, E \Rightarrow B, AC \rightarrow BE, BE \rightarrow C, C \rightarrow A\}$*. For this table, one possible cover is:* $\{B \rightarrow ACE\}$*, containing only one rule. It can be easily verified that all the rules in the GD-L basis are covered by the rule* $B \rightarrow ACE$*. Other possible covers are* $\{C \rightarrow ABE\}$*,* $\{E \rightarrow ABC\}$*, and* $\{BE \rightarrow AC\}$*, which contain rules that are equipotent to* $B \rightarrow ACE$*.* □

The next theorem expresses some important properties of cover rules.

Theorem 3.7 *Let* $\mathcal{C}$ *be a cover of a set of association rules* $\mathcal{R}$ *extracted from a table* τ*. The following statements hold:*

1. *If* $r_1, r_2 \in \mathcal{C}$*, then* $\mathsf{items}(r_1) \neq \mathsf{items}(r_2)$*.*

2. *If* r *is a* $\mathcal{C}$*-cover rule, then for any* $r' \in \mathcal{R}$ *such that* $\mathsf{items}(r') = \mathsf{items}(r)$*, we have*

$$\mathsf{supp}(r) \leq \mathsf{supp}(r') \text{ and } \mathsf{conf}(r) \leq \mathsf{conf}(r').$$

3. *If* r *is a* $\mathcal{C}$*-cover rule, then there is no* $r' \in \mathcal{R}$ *such that* $\mathsf{antc}(r) = \mathsf{antc}(r')$ *and* $\mathsf{cons}(r) \subset \mathsf{cons}(r')$*.*

Proof.

1. If two $\mathcal{C}$-cover rules have the same set of items, then one of them will be covered by the other and therefore does not belong to the cover.

2. This is immediate from the fact that the coverage rule only allows the inference of rules with greater or equal values of support and confidence.

3. Suppose that r is the $\mathcal{C}$-cover rule $I \rightarrow X$ and that $r' \in \mathcal{R}$ has the form $I \rightarrow Y$ such that $X \subset Y$. Note that $r' \prec r$ and the two rules are not equipotent since $\mathsf{items}(r) \neq \mathsf{items}(r')$. If r' is a cover rule, then r is redundant and can't belong to the cover, so r' cannot be a cover rule. Therefore, there is a $\mathcal{C}$-cover rule r_1, such that $r_1 \prec r'$ and $r_1 \neq r$. Thus, we have $r_1 \prec r$, so r is redundant, which contradicts our hypothesis.

∎

Out of the set of possible covers, some of them are more informative than others:

Definition 3.6 *An **informative cover** is a cover where for each cover rule r there is no equipotent rule r' such that $\mathsf{antc}(r') \subset \mathsf{antc}(r)$.* □

Theorem 3.8 *Let $\mathcal{C}$ be an informative cover of a set of association rules $\mathcal{R}$ extracted from a table τ. If r is a $\mathcal{C}$-cover rule, then there is no valid rule r' such that $\mathsf{items}(r') = \mathsf{items}(r)$ and $\mathsf{antc}(r') \subset \mathsf{antc}(r)$.*

Proof. Assume r has the form $X \rightarrow I - X$ and suppose there exists a valid rule r' of the form $Y \rightarrow I - Y$ with $Y \subset X$. If this happens, then the rule $X \rightarrow I - X$ is covered by $Y \rightarrow I - Y$, because if $Y \subset X$, then we have $\mathsf{supp}(Y) \geq \mathsf{supp}(X)$. The rule $Y \rightarrow I - Y$ cannot be a cover rule itself, because of the first property from Theorem 3.7, so it must be covered by some rule r and, by the transitivity of the coverage relation, it follows that $X \rightarrow I - X$ is covered by r. Because $X \rightarrow I - X$ is a cover rule, this can happen only if $r = X \rightarrow I - X$ is equipotent to $I \rightarrow I - Y$, but this contradicts Definition 3.6. Thus, the rule $Y \rightarrow I - Y$ with $Y \subset X$ cannot exist. ∎

Note that it is possible to have an informative cover rule r and a valid rule r', such that $\mathsf{items}(r) = \mathsf{items}(r')$ and $|\mathsf{antc}(r)| > |\mathsf{antc}(r')|$, as the next example shows.

Example 3.4 *Consider the data from Table 3.2. We will examine this table assuming* $\mathsf{minsupp} = 0.3$ *and* $\mathsf{minconf} = 0.7$.

tid	A	B	C
1	1	0	0
2	1	0	0
3	1	0	0
4	1	1	0
5	1	1	1
6	1	1	1
7	1	1	1
8	0	1	0
9	0	1	0
10	0	1	0

Table 3.2: Binary table illustrating that a valid rule can have smaller antecedent than a rule of an informative cover

There are six valid association rules: $\{C \Rightarrow B, C \Rightarrow A, BC \Rightarrow A, AC \Rightarrow B, AB \rightarrow C, C \Rightarrow AB\}$. *The informative cover is* $\{AB \rightarrow C\}$. *Note that* $C \Rightarrow AB$ *has the same items as the cover rule but has a smaller antecedent.* □

An informative cover is interesting because it can be smaller than the bases mentioned in Section 3.1, it allows the inference of all valid rules and only the inference of valid rules, and, for given sets of items, it contains rules with minimum antecedent (among all possible covers) and maximum consequent. This last property is similar to the criteria used to judge the informativeness of a rule in [BPT00].

The reverse of this is the fact that cover rules have smaller support and confidence, which happens because the coverage rule only leads to rules with higher or equal support and confidence.

A cover summarizes the set of valid rules in a similar way in which the large itemsets summarize the set of frequent itemsets [Bay98]. A cover can also be used to simplify the presentation of association rules to users: initially, only cover rules could be shown to a user, then the user could select a cover rule r and retrieve a subset[1] of all rules covered by r, and then the process could be repeated. In this manner, the user could guide his search for association rules without being overwhelmed by their number. A similar type of rule exploration has been proposed in [LHM99], in the context of the so called *direction setting rules.*

3.3 The *CoverRules* algorithm

The following pseudocode describes algorithm *CoverRules* that generates an informative cover for the set of valid association rules.

Algorithm 3.1 ***(CoverRules)***

Input: *set of frequent itemsets.*

Output: *the list of cover rules.*

[1] For example, this subset could consist of the rules covered by r that have the same size and antecedent size, and of the covered rules of size inferior by 1 that have minimal antecedent size.

```
Uses: queue of frequent itemsets $\mathcal{Q}$, list of cover rules $\mathcal{C}$.

1 Initialize $\mathcal{Q}$ by enqueuing into it all maximal frequent itemsets, in decreasing
  order of their size.
2 If $\mathcal{Q}$ is empty, then output $\mathcal{C}$ and exit; else extract an itemset $I$ from $\mathcal{Q}$.
3 For all strict non-empty subsets of $I$, $I_i$ with $i = 1 \ldots 2^{|I|} - 2$, sorted primarily
  by their support values (decreasingly) and secondarily by their cardinality
  (increasingly), do:
    3.1 If the rule $I_i \rightarrow I - I_i$ is valid, then add it to $\mathcal{C}$ if it is not covered by a
        rule already in $\mathcal{C}$. Go to step 2.
    3.2 If $i = 1$ and $|I| > 2$, then add to $\mathcal{Q}$ each subset of $I$ that has size $|I| - 1$
        and that is not already included in an itemset from $\mathcal{Q}$. Continue step 3.
4 Return $\mathcal{C}$.
```

∎

Algorithm *CoverRules* starts from the set of maximal frequent itemsets and examines them in decreasing order of their cardinalities (steps 1–2). For each such itemset I, we search for a subset S having maximum support, such that $S \rightarrow I - S$ is a valid association rule (step 3). Such a rule is a candidate cover rule and, once found, the search stops and the rule is added to the set of cover rules $\mathcal{C}$, if it is not covered by one of the rules of $\mathcal{C}$ (step 3.1). During the examination of each subset S of I, we may encounter some subsets such that they cannot be used as an antecedent of a rule based on the items of I. For these subsets, we will have to verify whether they can be antecedents of rules based on subsets of I. This is why, in step 3.2 of the algorithm, we add to $\mathcal{Q}$ all the subsets of I. Those subsets that are already included in an itemset of $\mathcal{Q}$, however, do not need to be added. Step 3.2 needs to be performed only once, so we perform it if the first subset examined in step 3 cannot be used as an antecedent. The collection $\mathcal{Q}$ is a queue because we want to examine the large itemsets in decreasing order of their size before we examine their subsets (added in step 3.2). We examine these itemsets in decreasing order of their size because an association rule whose set of items is larger cannot be covered by an association rule whose set of items is smaller. This ensures that a cover rule added to $\mathcal{C}$ cannot be covered by another cover rule that we may discover later. Each time that we intend to add a rule to $\mathcal{C}$, however, we still need to check whether that rule can be covered by one of the rules already in $\mathcal{C}$.

The strategy of examining first the maximal frequent itemsets and then their subsets, in decreasing order of their size, guarantees that the set of rules that we generate is minimal. Step 3.2 guarantees that all valid rules can be inferred from the rules in set $\mathcal{C}$. Together, these ensure that the resulting set $\mathcal{C}$ is a minimal set of rules from which all valid rules can be inferred, and thus, $\mathcal{C}$ is a cover. The cover is informative because, in step 3, for subsets having the same support, we examine first those with smaller cardinality.

There are several optimizations that can be made to Algorithm 3.1. The examination of the subsets from step 3 of *CoverRules* can be done in stages: at each stage we can examine subsets of equal cardinality, and we can reduce the number of subsets that need to be examined during the next stage, based on the results obtained in the current stage. For example, if during the examination of the subsets of size k we discover one with high support that can be used to generate a possible cover rule, then in the next stages, when examining subsets of cardinality greater than k, we will ignore all those subsets whose support is less than the support of this antecedent. The search from step 3 should also be done using binary search. For the sake of clarity, the description of Algorithm 3.1 was intentionally kept simple and omitted the details of such optimizations.

3.4 Experimental results

We implemented in Java (see [Cri02]) the optimized version of algorithm *Cover-Rules* and we tested the implementation on several databases. In a first experiment, we executed the algorithm on the Mushroom database obtained from the UCI Repository of Machine Learning Databases [BM98]. We present in Table 3.3 the results that we obtained, as well as the results obtained for this database by [PBT99a]. Note that the UCI repository contains two versions of the Mushroom database. We have used the version containing fewer rows, which was used in the experiments of [PBT99a].

Mushroom database (minsupp = 30%)			
minconf	Valid rules	Cover	GD-L Basis
90%	20399	238	382
70%	45147	176	453
50%	64179	159	479
30%	78888	78	493

Table 3.3: Results for Mushroom database

From these results it can be observed that the cover contains fewer rules than the GD-L basis. Another interesting result is the fact that the size of the cover decreases as minconf is lowered (see also Figure 3.1). This may seem surprising at first, but is due to the fact that, as minconf is lowered, more valid rules exist, the redundancy of these rules is greater, and thus they can be summarized better. In fact, for minconf = 30%, the size of the cover is identical to the number of large itemsets existing in the mushroom database (for minsupp = 30% there are 78 such maximal frequent itemsets), and this happens because all rules that can be generated using subsets of a maximal frequent itemset are valid. Notice that, in this case, the cover size is one order of magnitude smaller than the size of the GD-L basis, and three orders of magnitude smaller than the total number of valid association rules.

For minconf = 30%, all cover rules have the item veil-type = partial as antecedent. Interestingly, this item is common to all the mushrooms described in the database, so its support value is 1 — the maximum possible support value. By looking at a cover rule separately, the fact that the rule has the most frequent item as antecedent might make us think that the rule is trivial. Knowing that this is a cover rule, however, its antecedent being the most frequent item takes new meaning because it implies that any association rule that we can build from the items of the cover rule will be a valid association rule. Usually, the most frequent items are known to the users of the database, so a cover rule having such an item as antecedent can be easily interpreted, even without the help of the computer.

In the case of the Mushroom database, the *CoverRules* algorithm is about as fast as the *Apriori ap-genrules* procedure for generating all valid rules. Both algorithms finished their processing in a couple of seconds, so we do not include their detailed time results here.

We also tested our algorithm on synthetically generated data. For this purpose, we used our own implementation of the synthetic data generator described in [AS94a]. Our synthetic data generator is integrated in ARtool [Cri02]. We generated database SPARSE with 100,000 transactions of average size 10, having

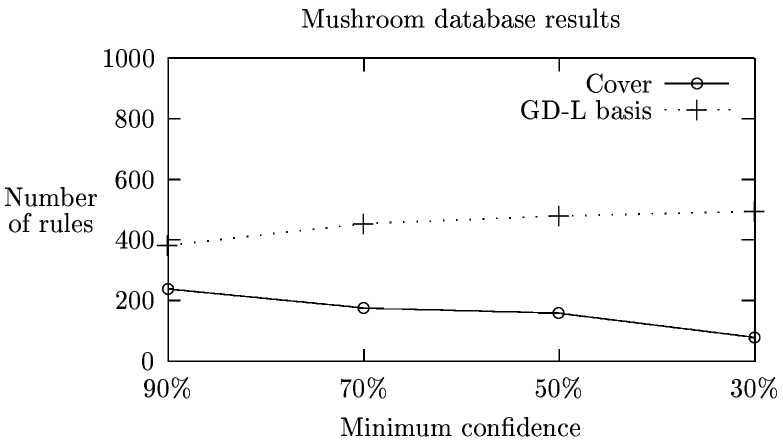


Figure 3.1: Graphical plot of the results obtained on the Mushroom database

100 items, and containing 300 patterns of average size 5. This is a sparse database that we mined for minsupp = 5%, thus discovering 207 maximal frequent itemsets. For all our experiments on this database, the times taken by *CoverRules* and *ap-genrules* were well below 1 second, so we omit them again. The number of association rules discovered and the corresponding cover size are presented in Table 3.4.

We can observe that, in this experiment, the cover size increases initially as minconf decreases. This happens because the database is sparse, so the redundancy is poor and rules that are discovered when the confidence threshold is lowered do not necessarily allow the inference of rules with higher confidence. For minconf = 10%, we obtain all valid rules and lowering the confidence threshold further does not bring any new rules. In fact, the 194 cover rules correspond to the large itemsets that have cardinality greater than one, because there are 13 such maximal frequent itemsets of size one.

SPARSE database (minsupp = 5%)		
minconf	Valid rules	Cover
90%	3	2
80%	19	13
70%	42	25
60%	87	55
50%	186	124
40%	321	196
30%	455	240
20%	658	257
10%	880	194
5%	880	194
1%	880	194

Table 3.4: Results for SPARSE database

For our final experiment, we generated a dense synthetic database, which we will call DENSE, with 100,000 transactions of average size 15, having 100 items, and containing 100 patterns of average size 10. Our strategy for obtaining dense synthetic databases consists of choosing fewer and longer patterns. We mined this database for minsupp = 5% and we obtained 3,182 maximal frequent itemsets. For this experiment, the times taken by the *CoverRules* and *ap-genrules* algorithms became noticeable and we include them in Table 3.5.

We can see from these results that again, for this dense database, the cover size generally tends to decrease as we lower the confidence threshold. All valid rules are discovered for confidence 5%, so lowering minconf further does not result in more rules. There is only one large itemset of size one, which accounts for the

DENSE database (minsupp = 5%)				
minconf	Valid rules	*ap-genrules*	Cover	*CoverRules*
		Time(seconds)		Time(seconds)
90%	87722	9	8875	215
80%	344001	30	9375	236
70%	511191	46	9020	220
60%	574554	49	7878	178
50%	603861	50	6483	130
40%	630706	52	6506	133
30%	656724	51	5496	104
20%	682076	53	5674	99
10%	703373	52	3416	41
5%	703924	52	3181	37
1%	703924	52	3181	37

Table 3.5: Results for DENSE database

difference between the number of large itemsets and the cover size obtained in this case. The time taken by the rule generation algorithms is more significative and allows us to notice that *CoverRules*'s performance tends to improve with the lowering of the confidence threshold, while *ap-genrules* tends to take more time as minconf is decreased. *ap-genrules* runs initially faster than *CoverRules*, which performs better for lower values of minconf. These results, however, do not include the time necessary to output the generated association rules, which would have added to the time spent by *ap-genrules*. The space requirements of *ap-genrules* are more significant than those of *CoverRules*, and in some experiments we had to increase the memory available to the Java Virtual Machine so that *ap-genrules* would not run out of memory. This is why, for this experiment, we ran our Java programs on a Sun Ultra-10 with 512MB RAM that did not have a JIT compiler, which is another important reason why the time taken by the algorithms became more noticeable than in the previous experiments.

In general, as expected, the performance of *CoverRules*, as well as that of *ap-genrules*, slows down when the databases are denser, and when the number of large itemsets increases. The performance of the algorithms varies differently with the change of minconf. For dense databases, the size of the cover is one–two orders of magnitude smaller than the number of valid association rules and shows the tendency of getting smaller as the redundancy in the generated association rules increases.

3.5 Conclusions

We presented a new inference rule for association rules and, based on it, we introduced the concepts of cover and informative cover for the set of valid association rules. We discussed the properties of these covers and their relation to the previous research performed on the topic of bases for association rules, and we presented algorithm *CoverRules* for the generation of an informative cover. Our experimental results show that *CoverRules* is about as efficient as the *ap-genrules* method, and that the informative cover is usually smaller than both the number of association rules and the size of the GD-L basis. In some cases, the size of the informative cover is one or two orders of magnitude smaller than the GD-L basis. Another important property is the fact that the size of the cover tends to decrease when the redundancy of the association rules increases, which is due to the fact that the informative cover takes better advantage of the redundancy to provide a useful summarization of the set of association rules.

CHAPTER 4

Mining frequent itemsets and association rules in multiple-table databases

The data mining algorithms presented in the previous chapters handle databases consisting of a single table. In this chapter, we address the problem of mining association rules in databases consisting of multiple tables and designed using the entity-relationship model (see [Mai83], [ST95]). We discuss previous approaches to this problem, point out some unaddressed issues, and present a couple of algorithms to address these issues. We also analyze the possibility of extending our algorithms to database schemas more complex than a star schema.

There are very few published results on how to mine association rules when data reside in more than one table. An instance of this problem has been addressed in a machine learning setting by Dehaspe and De Raedt [DR97], for the special case of mining a deductive relational database containing knowledge about some type of entity (for example, entities could be kids or sentences) and several weak entity sets dependent on this entity set. The work in [DR97], however, does not analyze how mining should be performed in databases involving relationships between multiple entities as is usually the case in a relational database.

The term Multi-Relational Data Mining has been introduced by Knobbe, Blockeel, Siebes, and van der Wallen in [KBS99], to describe the problem of finding interesting patterns about sets of tuples belonging to a user selected table, named **target table**. The output of a MRDM algorithm is usually a decision tree [KSW99]. Despite its general name, MRDM does not concern itself with the discovery of association rules, which is the problem that we address in this chapter.

Recently, Jensen and Soparkar [JS00] have addressed the problem of mining association rules in multiple tables for the special and important case when the database is organized in a star schema. They proposed replacing the application of the *Apriori* [AS94b] algorithm on a table obtained by joining all schema's tables by a technique (which we will call the *JS* algorithm) that in a first stage looks for frequent itemsets in each separate table using a slightly modified version of *Apriori*, and then, in a second stage, finds all frequent itemsets whose items belong to more than one table. Finally, this method would yield the same results as those yielded by *Apriori* when executed on the joined schema tables, and would have better performance then the latter method.

We show ([CS01]) that both methods, the *Apriori* algorithm and the *JS* algorithm, can produce rules which may not reflect accurately the actual relationships existing in data in cases like the one used as an example in [JS00], and we investigate how association rule mining should be performed in these cases.

4.1 Problems in previous approaches

We begin with a few definitions related to relational database theory [ST95]. We view a table as a triple $\tau = (T, H, \rho)$, where T is the name of the table, $H = A_1 \ldots A_n$ is its heading, and $\rho \subseteq \mathsf{Dom}(A_1) \times \cdots \times \mathsf{Dom}(A_n)$ is the content of τ.

The star database design that we consider here is derived from an entity relationship design that involves k entity sets and a set of k-ary relationships R that involve the instances of entity sets $E_1, \ldots, E_k$. Tables that represent entity sets are referred to as **entity tables**, and tables that represent relationship sets are called **relationship tables**. The tables are denoted by the same letter as their respective set. We denote the attributes of an entity set E by $\mathsf{attr}E$.

To simplify our definitions and discussions, we will assume that R has no other attributes than the foreign keys for the entity sets $E_1, \ldots, E_k$. Note that this is not a restrictive assumption because if the relationship set R contains other attributes than foreign keys, then we can simply consider them as being attributes of an extra entity set E_{k+1}, whose instances participate exactly once in a relationship from R.

We also assume that each relationship of R must involve an entity from each of the entity sets $E_1, \ldots, E_k$. This assumption corresponds to imposing referential integrity to R.

Let $\mathcal{E} = \{E_1, \ldots, E_k\}$ be a collection of k entity sets, R be a set of k-ary relationships between $E_1, \ldots, E_k$, `Join` be the join of tables $R, E_1, \ldots, E_k$, and `OuterJoin` be the outer join of tables $R, E_1, \ldots, E_k$. We use these notations throughout the chapter when discussing star schemas.

We redefine the notions of Definition 1.1 in the terms of the new relational database context.

Definition 4.1 *An **item** in a table $\tau = (T, H, \rho)$ is a pair $\langle A, a\rangle$, where $A \in H$ is an attribute and $a \in \mathsf{Dom}(A)$ is an attribute value. An **itemset** is a set of items $\{\langle A_1, a_1\rangle \ldots \langle A_n, a_n\rangle\}$, such that, if $i \neq j$, then $A_i \neq A_j$ for any $i, j \in \{1, \ldots, n\}$. In other words, an itemset contains no two pairs of items with an identical attribute component. The **support** of an itemset $I = \{\langle A_1, a_1\rangle, \ldots, \langle A_n, a_n\rangle\}$ with respect to a table $\tau = (T, H, \rho)$ is $\mathsf{supp}(I) = |\{t \in \rho \mid t[A_i] = a_i \text{ for all } i \in \{1, \ldots, n\}\}|/|\rho|$.*

*An **association rule** is an ordered pair $\langle a, c\rangle$ of itemsets such that they do not contain items with an identical attribute component. The first itemset in the*

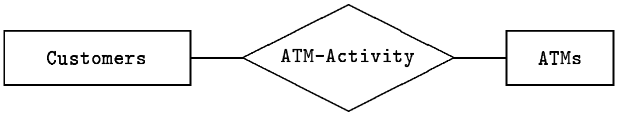


Figure 4.1: Entity-relationship diagram of Bank database

pair is called ***antecedent*** *and the second is called* ***consequent****. We represent the association rule as $a \rightarrow c$. The* ***support*** *of an association rule is defined to be the support of the union of its antecedent and consequent itemsets. The* ***confidence*** *of an association rule is defined to be the ratio between the support of the rule and the support of the antecedent.* □

To illustrate our arguments, we use an example similar to the one presented in [JS00], with the difference that we changed the data to make some points easier to observe.

The example is that of a simplified banking environment organized in a star schema (see Figure 4.1) consisting of two tables that reflect entity sets: the `Customers` table with attributes {*acct#*, *balance*, *age*}, the `ATMs` table with attributes {*atm#*, *type*, *limit*}, and a table that represents a set of binary relationships named `ATM-Activity` with attributes {*acct#*, *atm#*, *amount*}. The contents of these tables are presented in Table 4.1[1].

The table `Join` obtained from joining the three tables is identical to the table that we would obtain by performing an outer join and is presented in Table 4.2.

[1] Note that `ATM-Activity` has an attribute (*amount*) that indicates the value of the transaction; this differs from our assumption that R should contain only foreign key attributes. To satisfy our assumption we could consider a third entity set, the transactions, resulting in a star schema that involves three entities and a ternary relationship.

`Customers`

acct#	*balance*	*age*
1	50000-100000	30-40
2	1000-5000	30-40
3	100-1000	20-30

`ATM-Activity`

acct#	*atm#*	*amount*
1	1	500-1000
2	2	50-100
1	2	50-100
3	2	0-50
1	1	500-1000

`ATMs`

atm#	*type*	*limit*
1	in	1000
2	out	500

Table 4.1: The tables of the Bank database

We question whether an algorithm that mines this star schema database for association rules is useful when it finds exactly the same results obtained by executing the *Apriori* algorithm on the joined tables. We claim that the generation of correct rules requires the knowledge of the entities and relationships existing in the database, and we will look at some examples that illustrate this idea.

Let us consider the rule $age = 30 - 40 \rightarrow balance = 50000 - 100000$ which involves attributes *age* and *balance* belonging to entity table `Customers`. If we consider the `Join` table, then the support of this rule would be 60% (3/5) and its confidence would be 75% (3/4). Another approach, however, would be to examine only the `Customers` table, in which case the support of the rule would be 33.3% (1/3) and its confidence would be 50% (1/2). A question now arises about which

`Join`

acct#	*atm#*	*amount*	*balance*	*age*	*type*	*limit*
1	1	500-1000	50000-100000	30-40	in	1000
2	2	50-100	1000-5000	30-40	out	500
1	2	50-100	50000-100000	30-40	out	500
3	2	0-50	100-1000	20-30	out	500
1	1	500-1000	50000-100000	30-40	in	1000

Table 4.2: The join of the tables of the Bank database

of these two results is the correct one. Because the rule involves only attributes of the `Customers` table, we argue that the support and confidence of the rule should be based on this table only, so that we should obtain the second result. This is because `Customers` represents an entity set and properties of its attributes should be determined only by looking at the set of instances of customers. When executed on the joined tables, however, *Apriori* would generate the first result, thus producing what we consider to be an association rule with misleading support and/or confidence values.

It is also worth mentioning that, because the support of a set of attribute values corresponding to one entity can be smaller with respect to the joined table than with respect to the entity table, the itemset may not even be discovered by *Apriori*, much less be used to generate an association rule. For example, the support of *age=20-30* is 20% (1/5) with respect to `Join` and 33.33% (1/3) with respect to table `Customers`. Such an itemset would be missed by the *JS* algorithm when mining with minsupp = 30%.

Let us now consider another example in the rule $age = 30-40 \rightarrow type = in$. In this case, the rule contains attribute *age* from entity `Customers` and attribute *type* from entity `ATMs`. Because the rule contains attributes from two different entities and these entities are related through relationship `ATM-Activity`, it makes sense to compute the support and confidence of this rule with respect to the table obtained by joining `Customers`, `ATM-Activity`, and `ATMs`, which in our case is equivalent to using the table `Join`. By looking at the `Join` table, we can compute the support for this rule as 40% (2/5) and its confidence as 50% (2/4).

From these examples we can draw several conclusions:

1. Running the *Apriori* algorithm on the join of the tables of a database can fail to produce all existing rules or may produce rules whose support and confidence do not properly reflect the knowledge embedded in the data.

2. The entity-relationship model of a database provides important information concerning the relations between entities, and this information should be used by data mining algorithms.

3. When looking for association rules, rules among attributes of the same entity should be analyzed with respect to that entity set. When looking for association rules among attributes of several entities, we need to look at how these attributes appear together, so we need to analyze rules with respect to the relationships existing between the entities. Note that if several relationships exist between two or more entities, then the association rules between their attributes must be examined with respect to each such relationship. We discuss this issue in Section 4.3.

We have thus identified a problem that appears when mining a database built from an entity-relationship model using standard mining algorithms. New algorithms are needed to address this problem and in the next sections we discuss such algorithms. Initially, we address the basic case of star schemas and then we discuss the case of more complex schemas.

4.2 Mining association rules in star schemas

We examine extending the *Apriori* algorithm to make it work on the join of all tables, and then we investigate a method for mining association rules without joining the tables, assuming a star schema organization of the database tables.

Definition 4.2 *An **entity itemset** is an itemset* $\{\langle A_1, a_1\rangle, \ldots, \langle A_n, a_n\rangle\}$*, such that* $\{A_i \mid i \in \{1, \ldots, n\}\} \subseteq \mathsf{attr}E$ *for some entity set* $E \in \mathcal{E}$*.*

*A **join itemset** is an itemset* $I = \{\langle A_1, a_1\rangle, \ldots, \langle A_n, a_n\rangle\}$*, with the property that* $\bigcup_{i=1}^{n} A_i \subseteq \bigcup_{j=1}^{k} \mathsf{attr}E_j$*, and such that* I *is not an entity itemset. In other words, a join itemset is an itemset whose attributes do not belong to the same entity.*

*The support of an entity itemset with respect to its entity table is called **entity support**.*

The support of an entity or join itemset with respect to the table **Join** *is called **join support**.* □

Note that we can compute the join support for any itemset, but the entity support is only defined for entity itemsets.

It is impossible to predict the relative magnitude of the join support and entity support for an itemset. The entity support may be greater or smaller than the join support. In the example presented in the second section we have seen that the join support of itemset $\{age = 30 - 40, balance = 50000 - 100000\}$ (60%) was greater than its entity support (33.3%). On the other hand, the join support of itemset $\{age = 20 - 30\}$ (20%) was smaller than its entity support (33.3%). This means that, in the execution of an *Apriori* algorithm, for a candidate entity itemset we should compute both its entity support and its join support. If the entity support is greater than minsupp, then the itemset should be considered frequent; otherwise, if the join support is greater than minsupp, then the itemset is not frequent but it might be a subset of a frequent join itemset and thus should be used in the candidate generation procedure. Finally, if both supports are smaller than minsupp, then the itemset should be discarded.

We propose to modify the *Apriori* algorithm as follows: whenever we compute and evaluate the support of a candidate itemset, we consider two possibilities:

1. If the itemset is an entity itemset, then both its join and entity supports should be computed. The itemset should be considered frequent if its entity support is greater than minsupp; otherwise, if its join support is greater than minsupp, then the itemset should be used in generating new candidate itemsets for the next step of the algorithm. If none of these conditions is met, then the itemset should be discarded.

2. If the itemset is a join itemset, then its join support should be computed and checked against the minimum support to determine whether the itemset is frequent; otherwise, the itemset should be discarded.

In the *apriori-gen* procedure, which combines two itemsets of length k into a candidate itemset of length $k+1$, we would again have two possible cases:

1. If the resulting candidate itemset is an entity itemset, then we need to check that all its subsets have entity support greater than minsupp or that all its subsets have a join support greater than minsupp (so that the itemset can be used later for the generation of a candidate join itemset); otherwise, the candidate should not be generated.

2. If the resulting candidate itemset is a join itemset, then we should just verify that all its subsets have join support greater than minsupp; otherwise we should not generate the candidate.

Next, we discuss how these changes can be used to obtain an algorithm for mining the joined tables and an algorithm for mining the star schema.

4.2.1 The *AprioriJoin* algorithm

In this section we want to devise an algorithm capable to mine a table containing all the data existing in a star schema. Note that such a table can be obtained by performing an outer join on all tables of the star schema. A simple join would not be sufficient because we could miss entity instances that do not participate in the relationship.

In order for *Apriori* to generate rules with proper support and confidence values, it has to know about the existing entities that appear in the table on which it is run. This can be done if the algorithm knows for each attribute what is the key attribute of the respective entity. We focus on the problem of finding large itemsets because from these we can construct association rules by applying standard methods like *ap-genrules*. The algorithm described in this section is named *AprioriJoin*; we assume that its input table is the `OuterJoin` table.

To compute the support of an itemset we need to differentiate between entity itemsets and join itemsets.

For an entity itemset, we compute its entity support as follows. We first count the number of rows of `OuterJoin` that contain the itemset and that have a distinct entity key. Then we divide this count by the number of distinct entity keys to find the percentage of the entity support. To compute the join support we have to count the number of rows that contain the itemset and that belong to `Join` but not to `OuterJoin`, and then divide this value by the cardinality of `Join`. Identifying the rows which belong to `OuterJoin` but not to `Join` is quite simple: these are the rows that contain null entity keys. The rows of `Join` are then identified as those that do not contain null entity keys.

For a join itemset, we compute its join support simply by counting the rows that contain the itemset and then dividing this value by the cardinality of `Join`.

There are few implementation difficulties in adapting an *Apriori* algorithm to mine the `OuterJoin` table. The main steps of the resulting algorithm are presented in Algorithm 4.1.

Algorithm 4.1 ***(AprioriJoin)***

Input: *minsupp, the table* ***`OuterJoin`****, and the mapping of each attribute to an entity key.*

Output: *the list $\mathcal{F}$ of frequent itemsets.*

Uses: *list of candidate itemsets $\mathcal{C}$, list of frequent itemsets $\mathcal{K}$.*

1 Initialize $\mathcal{C}$ to contain all 1-itemsets.

2 Scan table ***`OuterJoin`*** *and compute the join support and, if appropriate, the entity support for all candidate itemsets.*

3 Place into collection $\mathcal{K}$ all itemsets from $\mathcal{C}$ that have entity support or join support greater than minsupp. *Place into collection $\mathcal{F}$ the entity itemsets that have entity support greater than* minsupp *and the join itemsets that have join support greater than* minsupp. *Clear $\mathcal{C}$.*

4 Generate new candidates by applying the apriori-gen method (modified as described at the beginning of this section) to $\mathcal{K}$. Place all newly generated itemsets into $\mathcal{C}$. Clear $\mathcal{K}$.

5 If $\mathcal{C}$ is empty, then return $\mathcal{F}$.

6 Go to step 2

∎

4.2.2 The *AprioriStar* algorithm

In this section we present *AprioriStar*, an algorithm that correctly finds the frequent itemsets in a set of tables organized in a star schema. The pseudocode of *AprioriStar* is described in Algorithm 4.2.

Algorithm 4.2 *(**AprioriStar**)*

Input: minsupp *and a star schema database with entity tables $E_1, \ldots, E_n$ and relationship table R.*

Output: *the list $\mathcal{F}$ of frequent itemsets.*

Uses: *list of candidate itemsets $\mathcal{C}$, list of frequent itemsets $\mathcal{K}$.*

1 Scan R and store the number of occurrences of every value of each foreign key (a step identical to the one in the JS algorithm).

2 Initialize $\mathcal{C}$ to contain all 1-itemsets (note also that these are entity itemsets)

3 Scan tables $E_1, \ldots, E_n$ and, during the scan of each table E_i, compute the entity support for all itemsets from E_i as well as their join support using the information collected during step 1 (whenever we see a row containing the itemset, the support of the itemset is increased with the number of occurrences of the row's key value in the R table).

4 Place into collection $\mathcal{K}$ all itemsets from $\mathcal{C}$ that have entity support or join support greater than minsupp*. Place into collection $\mathcal{F}$ the entity itemsets that have entity support greater than* minsupp *and the join itemsets that have join support greater than* minsupp*. Clear $\mathcal{C}$.*

5 Generate new candidates by applying the apriori-gen method (modified as described in the beginning of this section) to $\mathcal{K}$. Place all newly generated itemsets into $\mathcal{C}$. Clear $\mathcal{K}$.

6 If $\mathcal{C}$ is empty, then return $\mathcal{F}$.

7 For all entity itemsets in C, belonging to a table E_i, scan E_i and compute their join and entity supports.

8 Compute on the fly the join of tables $R, E_1, \ldots, E_n$ and then, for all join itemsets in $\mathcal{C}$, compute their join support with respect to the joined tables.

9 Go to step 4

■

Note that, in step 8, it is not required to compute the join of all tables if the candidate itemsets do not contain attributes of all entities. For example, if the candidate itemsets would just contain attributes of entities E_1 and E_2, we would just need to join R, E_1, E_2 and we could ignore tables $E_3, \ldots, E_n$.

The main differences between this algorithm and the *JS* algorithm are:

1. *AprioriStar* uses the join and entity supports in determining frequent itemsets. By considering the entity support, *AprioriStar* does not eliminate from the result the entity itemsets that are frequent with respect to their entity table but not with respect to the relationship table, and it also allows the computation of correct support and confidence for rules existing among attributes of the same entity table.

2. *AprioriStar* does not compute the support of the join itemsets in a single data scan, as it was done in the *JS* algorithm. Our approach avoids the explosion of join candidates that is present in the final step of the *JS* algorithm.

4.2.3 Experimental results

For our experiments, we implemented in Java the algorithms *AprioriJoin* and *AprioriStar*, and we executed them on binary files containing synthetically generated data. The synthetic data constitute a star schema with three entity sets and a relationship set. The entity tables were indexed for the faster random access needed by the implementation of *AprioriStar*.

The test data was built as follows: first we generated synthetic data for the *AprioriStar* algorithm by creating a database with three entity sets and a relationship set, and then we outer joined these tables to obtain the table on which we executed the *AprioriJoin* algorithm. For the generation of the entity tables we have used our implementation (see [Cri02]) of the synthetic data generator described in [AS94a], and for the generation of the relationship table we have designed and used a synthetic relation generator. The synthetic relation generator yields a number of relationships for each of the tuples of a specified entity table. The average and standard deviation for this number of relationships are specified by the user and are generated using a normal distribution. Also, for each such tuple, we randomly select a sample of tuples from every other entity table, so that the selected tuples can be involved in the relationships to be generated. The size of the sample selected from each entity table is produced using a Poisson distribution with mean specified by the user. Finally, we generate relationships by randomly selecting the participating tuples from the samples previously constructed.

We first generated two types of databases: a sparse one having a small average number of items per transaction and having fewer frequent itemsets, and a dense one having a larger average number of items per transaction and more frequent itemsets. We call these databases SPARSE and DENSE, respectively.

We verified the scalability of the algorithms by doubling twice the contents of the database SPARSE, thus obtaining databases SPARSEx2 and SPARSEx4. The doubling of the database SPARSE was done in the following way: first we duplicated the rows of all entity tables and then we duplicated the relations and made the duplicate relations refer to the duplicate entity instances instead of the original ones. Thus, we have obtained a database whose tables had twice the size of the original table, but which contained the same rules as the original database[2].

For both the SPARSE and DENSE databases, all the entity instances are participating in relationships from R, which is not the case for a third database that we generated, called OUTER, where only half of the transactions of each entity

[2] Note that the complexity of the mining operation is derived from the number of attributes of the database, and from the patterns existing in the data. Thus, to verify the scalability of the algorithms with respect to the variation of the database size, we have maintained constant both the number of attributes and the patterns existing in the data.

table participate in relationships of R. OUTER was obtained by generating two different sets of entity tables and then concatenating them. For this database, the relationship table R consists of the relationship table generated for the first set of tables.

The main characteristics of these datasets are presented in Table 4.3. For OUTER, we separated by a slash the characteristics of the first and second sets of tables that were used for its generation.

We have executed the two algorithms, *AprioriStar* and *AprioriJoin*, on a Sun Ultra-10, running Sun OS 5.7, with 512MB of memory. We used Sun's JDK 1.2.2 for compiling and running the programs, and we did not benefit from the use of a JIT compiler. The time was measured using the /usr/bin/time command.

Table 4.4 presents the results of our experiments. In parentheses we have indicated the number of frequent itemsets and the largest size of a frequent itemset for the respective algorithm execution. For example, (13/3) means that the algorithm has found 13 frequent itemsets and the largest one consisted of 3 items. Because SPARSEx2 and SPARSEx4 contain the same patterns as SPARSE, we have not duplicated this information in their case and we have only displayed the time taken by the algorithms.

There are a couple of important observations to be made from the test results. First, it can be observed that the algorithms scale linearly as the number of tuples of the database increases. With respect to the relative performance of the algorithms, we can note that the *AprioriJoin* algorithm is almost always a little faster than *AprioriStar* when working on databases where all entity instances are participating to the relation R. This isn't surprising, given that on such databases *AprioriStar* would find frequent join itemsets at each step, and would thus have to

Test database characteristics

Characteristics	SPARSE	DENSE	OUTER
Entity E_1			
Number transactions	10000	10000	20000
Average transaction size	7	20	7/12
Number of items	60	60	60
Number of frequent patterns	100	50	100
Average pattern length	5	10	4/10
Entity E_2			
Number transactions	100	100	200
Average transaction size	4	4	4/7
Number of items	10	10	10
Number of frequent patterns	10	10	10
Average pattern length	2	2	2/5
Entity E_3			
Number transactions	1000	1000	2000
Average transaction size	5	10	5/9
Number of items	20	20	20
Number of frequent patterns	50	20	50
Average pattern length	3	5	3/7
Relationship R			
Total number relationships	100380	95753	95985
Average and standard deviation of number of relationships for each tuple of entity one	10,8	10,5	10,5
Second entity sample mean	10	5	20
Third entity sample mean	20	20	50
Outer Join			
Number tuples	100380	95753	107085

Table 4.3: Characteristics of the synthetically generated databases

Test results

Minimum support	SPARSE	SPARSEx2	SPARSEx4	DENSE	OUTER
AprioriJoin results					
0.5	5m01s (13/3)	10m02s	20m23s	23m19s (912/7)	7m39s (26/3)
0.4	6m47s (38/4)	13m14s	27m23s	33m52s (3275/8)	7m58s (77/4)
0.3	6m59s (100/4)	13m54s	28m04s	1h11m21s (14788/10)	10m18s (217/5)
0.2	9m15s (340/5)	18m26s	37m28s	4h38m15s (106219/11)	13m31s (878/6)
0.1	17m06s (2145/7)	34m10s	1h08m49s	-	24m56s (5968/9)
AprioriStar results					
0.5	5m27s (13/3)	11m02s	20m44s	24m45s (912/7)	5m12s (26/3)
0.4	7m50s (38/4)	15m28s	30m45s	35m22s (3275/8)	7m40s (77/4)
0.3	7m59s (100/4)	15m38s	31m33s	1h14m15s (14788/10)	10m12s (217/5)
0.2	11m00s (340/5)	21m39s	43m38s	4h28m51s (106219/11)	13m30s (878/6)
0.1	21m00s (2145/7)	41m30s	1h22m31s	-	22m50s (5968/9)

Table 4.4: Results for databases SPARSE, SPARSEx2, SPARSEx4, DENSE, and OUTER

build the join of the tables as often as *AprioriJoin* would perform its scan. This means that *AprioriStar* would end up doing more I/O operations than *AprioriJoin* because it also has to scan the entity tables in a separate step. In the case of database OUTER, however, we can notice that *AprioriStar* outperforms in all cases *AprioriJoin*. This is due to a combination of two factors. The first factor is that, for database OUTER, *AprioriStar* generates only entity candidates during its last steps, so it will avoid performing a scan of the join of the tables and will just scan some of the entity tables. This is the main reason for the more significant difference obtained in the experiment where we used a minimum support value of 0.1. The second factor is that *AprioriJoin* works on a table that is obtained from an outer join and that is thus larger than the join table built on the fly by *AprioriStar*, which makes the I/O requirements of the algorithms to be comparable, hence the rather small difference in performance for the minimum support values 0.2–0.5.

In the next section we introduce the difficulties related to mining more complex database schemas.

4.3 Mining association rules in more general schemas

To handle more complex database designs, we start from the fact that the entity-relationship diagram of a database $\mathcal{D}$ is a bipartite, connected graph $\mathcal{G}_{\mathcal{D}}$ whose set of nodes is partitioned into entity sets and relationship sets. Let $\mathcal{E}$ be the collection of entity sets and let $\mathcal{R}$ be the collection of relationship sets involved in the model. For a set of attributes X, let $\mathcal{E}(X)$ be the set of all entity sets in $\mathcal{E}$ that contain some attributes in X. Also, for a collection of entity sets $\mathcal{F}$, where $\mathcal{F} \subseteq \mathcal{E}$, denote by $M(\mathcal{F})$ the collection of all minimal relationship sets $\mathcal{S}$, such that the subgraph of $\mathcal{G}_{\mathcal{D}}$ generated by $\mathcal{F} \cup \mathcal{S}$ is connected. For example, if $\mathcal{F} = \{E_1, E_2, E_3\}$ is a collection of entity sets (shown in Figure 4.2), then $M(\mathcal{F})$ consists of $\{\{R_1\}, \{R_2, R_3\}\}$. We refer to these members of $M(\mathcal{F})$ as *connectors* of $\mathcal{F}$.

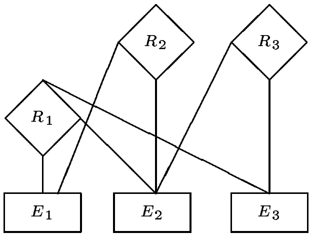


Figure 4.2: An example of an entity-relationship diagram

For a set of attributes X and every connector $\mathcal{Q}$ of $M(\mathcal{E}(X))$, we consider the subgraph $\mathcal{G}_{X,\mathcal{Q}}$ of the entity-relationship diagram determined by $\mathcal{E}(X) \cup \mathcal{Q}$. Each such subgraph is a union of "star" graphs and one can define the support of X

relative to the join of the tables that represent the entity sets and the relationship sets corresponding to the vertices of the graph $\mathcal{G}_{X,\mathcal{Q}}$.

For example, consider a small university database having two entities: *Professors* and *Students* (see Figure 4.3).

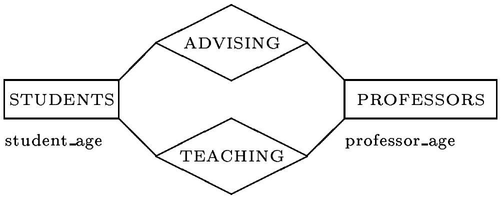


Figure 4.3: Entity-relationship diagram of University database

There are two relationships between the entities *ADVISING* and *TEACHING*. For the set of attributes $X = \mathit{student\_age}\ \mathit{professor\_age}$ we have:

$$\begin{aligned} \mathcal{E}(X) &= \mathit{STUDENTS\ PROFESSORS}, \\ M(\mathcal{E}(X)) &= \{\{ADVISING\}, \{TEACHING\}\}. \end{aligned}$$

Thus, for a rule like $student\_age > 30 \rightarrow professor\_age > 40$, one could contemplate the support relative to the table obtained by joining *ADVISING*, *STUDENTS*, and *PROFESSORS*, or the support relative to the table obtained by joining *TEACHING*, *STUDENTS*, and *PROFESSORS*.

These observations suggest some promising new research directions involving the design of efficient algorithms that compute the support of itemsets relative to the various connectors that may exist, and the analysis of the effect of the topology of the entity-relationship diagram on the resources required for these computations.

4.4 Conclusions

We approached the problem of mining association rules in databases consisting of several tables organized in a schema obtained from an entity-relationship design. We have focused mainly on the problem of mining a star schema database. We noticed that previous algorithms did not take advantage of the knowledge already embedded in an entity relationship model regarding the relationships between the database entities. To address this issue, we introduced the concepts of entity and join support, and we presented two algorithms: algorithm *AprioriJoin*, for mining the outer join of the tables using knowledge of the star schema organization of the tables, and algorithm *AprioriStar*, for directly mining the star schema database. These algorithms were tested on synthetically generated databases and the results of our tests show that both algorithms scale linearly with respect to the size of the input database. The results also show that in the case of a star schema there is no clear winner between the two algorithms in terms of time performance.

For schemas that are more complex than star schemas, the problem becomes more complicated because for a given itemset we may need to compute as many support values as there are connectors for the respective itemset. This can lead to an explosion of the number of support values that need to be computed for an itemset. We thus require new algorithms and techniques for handling such complex mining problems. This represents an interesting direction for future research.

CHAPTER 5

Mining purity dependencies in databases

Functional dependencies are an important concept in the design of databases, due to their role in normalization theory, which aims to minimize redundancy and anomalies in relational databases. The identification of these dependencies satisfied by database schemas is an important topic in data mining literature ([KM95], [HKP98]).

We propose ([SCC02b], [SCC02a]) a generalization of the notion of functional dependency starting from the notion of impurity of a subset of a set S relative to a partition of S; this notion is extended to the notion of relative impurity of two partitions. Because sets of attributes of a table of a relational database naturally generate partitions on the set of tuples (as we show in Section 5.2), it becomes possible to define the relative impurity of two sets of attributes. When this impurity is below a certain limit we say that the table satisfies a **purity dependency**. Purity dependencies have properties that are similar to those of functional dependencies and, as we show in the final section of this chapter, they can be useful in approximate classifications, that is, in classifications where certain errors are tolerable.

Unless stated otherwise, all sets considered in this chapter are finite. We begin with a few notations and definitions of terms.

The set

$$\{(p_1, \ldots, p_k) \in \mathbb{R}^k \mid p_i \geq 0 \text{ and } p_1 + \cdots + p_k = 1\}.$$

will be denoted by $\mathsf{SIMPLEX}_{k-1}$ and will be called **k-dimensional simplex**.

A function $f : \mathbb{R} \longrightarrow \mathbb{R}$ is:

- **concave** on a set $S \subseteq \mathbb{R}$ if $f(\alpha x + (1-\alpha)y) \geq \alpha f(x) + (1-\alpha)f(y)$ for $\alpha \in [0,1]$ and $x, y \in S$;
- **sub-additive** on S if $f(x+y) \leq f(x) + f(y)$ for $x, y \in S$.

For example, $x - x^2$ is concave on the set $\mathbb{R}$ and $-\log x$ is sub-additive on the set $[0,1]$.

In [CHH99], a **concave impurity measure** is defined as a real-valued function $i : \mathsf{SIMPLEX}_{k-1} \longrightarrow \mathbb{R}$ that satisfies the following conditions:

(i) $i(\alpha \mathbf{p} + (1-\alpha)\mathbf{q}) \geq \alpha i(\mathbf{p}) + (1-\alpha) i(\mathbf{q})$ for any $\alpha \in [0,1]$ and $\mathbf{p}, \mathbf{q} \in \mathsf{SIMPLEX}_{k-1}$, with equality if and only if $\mathbf{p} = \mathbf{q}$;

(ii) if $\mathbf{p} = (p_1, \ldots, p_k)$, then $i(\mathbf{p}) = 0$ if $p_j = 1$ for some j, $1 \leq j \leq k$.

The corresponding **frequency-weighted impurity measure** is the function $I : \mathbb{N}^k \longrightarrow \mathbb{R}$ given by

$$I(n_1, \ldots, n_k) = N i\left(\frac{n_1}{N}, \ldots, \frac{n_k}{N}\right),$$

where $N = \sum_{j=1}^{k} n_j$.

In [CHH99], the authors show that both the Gini impurity measure and the entropy measure can be generated using a simple one-argument function that satisfies certain conditions. In this paper we additionally require sub-additivity of this function, as shown in the next definition.

Definition 5.1 *A function $f : [0,1] \longrightarrow \mathbb{R}$ is a* ***generator****, if it is concave, subadditive, and $f(0) = f(1) = 0$.*

The ***monogenic impurity measure*** *induced by the generator f is the impurity measure generated by the concave impurity measure i having the form*

$$i(p_1, \ldots, p_k) = f(p_1) + \cdots + f(p_k),$$

where $(p_1, \ldots, p_k) \in \mathsf{SIMPLEX}_{k-1}$. □

It is easy to verify that such functions as $f_{\mathsf{gini}}(p) = p - p^2$, $f_{\mathsf{ent}}(p) = -p \log p$, $f_{\mathsf{sq}}(p) = \sqrt{p} - p$, or

$$f_{\mathsf{peak}}(p) = \begin{cases} p & \text{if } 0 \leq p \leq 0.5 \\ 1 - p & \text{if } 0.5 < p \leq 1 \end{cases}$$

are generators. (In the definition of f_{ent} we assume that $0 \cdot \infty = 0$.) Thus, both the Gini impurity measure, induced by f_{gini}, and the entropy measure, induced by f_{ent}, are monogenic impurity measures.

Our definition of the entropy measure is an alternative approach to the axiomatic definitions of entropy, a subject much discussed in information theory (see [Khi56], [Khi57], [IU62], [GT66], and [MR75]), and has the advantage of opening the possibility of some useful generalizations. For an axiomatic presentation of partition entropies see [SJ01].

Further examples of generators are $f_{\mathsf{sin}}(p) = \sin \pi p$, $f_{\mathsf{circle}} = \sqrt{p - p^2}$, $f_0(p) = 1 - e^{p^2 - p}$, where $p \in [0,1]$, or

$$f(p) = \begin{cases} 1 & \text{if } 0 < p < 1 \\ 0 & \text{if } p = 0 \text{ or } p = 1. \end{cases}$$

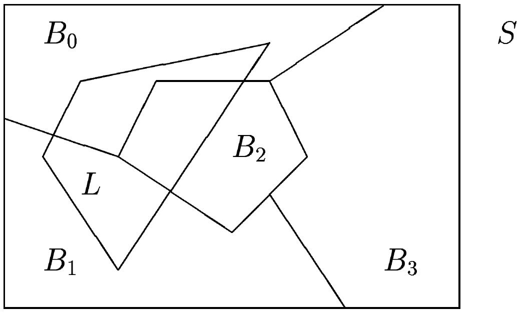


Figure 5.1: A 4-block partition of S and an impure subset L

For a concave function f, the inequality

$$f(p_1) + \cdots + f(p_k) \leq kf\left(\frac{1}{k}\right) \tag{5.1}$$

holds for every $(p_1, \ldots, p_k) \in \mathsf{SIMPLEX}_{k-1}$, and is known as *Jensen's inequality.* This implies that the largest value of the sum $f(p_1) + \cdots + f(p_k)$ is achieved if and only if $p_1 = \cdots = p_k = \frac{1}{k}$. Therefore, for the monogenic impurity measure generated by the function f, we have $0 \leq i(p_1, \ldots, p_k) \leq kf(\frac{1}{k})$ for $(p_1, \ldots, p_k) \in \mathsf{SIMPLEX}_{k-1}$.

5.1 Impurity of sets and partitions

In this section we introduce the notion of impurity of a subset of a set S relative to a partition of S. In turn, this notion is used to define the impurity of a partition relative to another partition.

Definition 5.2 *Let f be a generator, S be a set and let $\mathsf{PART}(S)$ be the set of all partitions of S. The* ***impurity of a subset*** *L* ***of*** *S* ***relative to a partition*** *$\pi \in \mathsf{PART}(S)$* ***and generated by*** *f is the monogenic impurity measure induced* *by the generator f:*

$$\mathsf{IMP}^f_\pi(L) = |L| \left(f\left(\frac{|L \cap B_1|}{|L|}\right) + \cdots + f\left(\frac{|L \cap B_n|}{|L|}\right)\right), \tag{5.2}$$

where $\pi = \{B_1, \ldots, B_n\}$.

The ***specific impurity of*** *L* ***relative to*** *π* ***and generated by*** *f is*

$$\mathsf{imp}^f_\pi(L) = \frac{\mathsf{IMP}^f_\pi(L)}{|L|} = f\left(\frac{|L \cap B_1|}{|L|}\right) + \cdots + f\left(\frac{|L \cap B_n|}{|L|}\right). \tag{5.3}$$

*When the subset L is included in one of the blocks of partition π, $\mathsf{IMP}^f_\pi(L) = 0$ and L is called a π-**pure** set; otherwise, L is called a π-**impure** set.* □

Note that if L is π-impure, then $|L| \geq 2$.

In Figure 5.1, we show an impure set L that intersects three of the four blocks of a partition of a set S.

Note that Jensen's inequality (5.1) implies that the impurity of a set $L \subseteq S$ relative to a partition $\pi = \{B_1, \ldots, B_n\}$ of S, as defined in (5.2), cannot exceed the value

$$m^f_{L,\pi} = |L| \cdot n \cdot f\left(\frac{1}{n}\right). \tag{5.4}$$

The impurity measure is superadditive and monotone, as shown by Theorem 5.1 and Corollary 5.2.

Theorem 5.1 *Let K, L be two disjoint subsets of the set S and let $\pi \in \mathsf{PART}(S)$. Then, we have*

$$\mathsf{IMP}^f_\pi(K \cup L) \geq \mathsf{IMP}^f_\pi(K) + \mathsf{IMP}^f_\pi(L).$$

Proof. The definition of imp^f_π allows us to write

$$\begin{aligned}\mathsf{imp}^f_\pi(K \cup L) &= \sum_{\ell=1}^{n} f\left(\frac{|(K\cup L)\cap B_\ell|}{|K\cup L|}\right)\\ &= \sum_{\ell=1}^{n} f\left(\frac{|K\cap B_\ell| + |L \cap B_\ell|}{|K\cup L|}\right),\end{aligned}$$

because K and L are disjoint.

Because $\frac{|K\cap B_\ell|+|L\cap B_\ell|}{|K\cup L|}$ is a convex combination of $\frac{|K\cap B_\ell|}{|K|}$ and $\frac{|L\cap B_\ell|}{|L|}$, the concavity of f allows us to write

$$f\left(\frac{|K\cap B_\ell| + |L\cap B_\ell|}{|K\cup L|}\right) \geq \frac{|K|}{|K|+|L|} f\left(\frac{|K\cap B_\ell|}{|K|}\right) + \frac{|L|}{|K|+|L|} f\left(\frac{|L\cap B_\ell|}{|L|}\right),$$

so

$$\mathsf{imp}^f_\pi(K\cup L) \geq \frac{|K|}{|K|+|L|}\mathsf{imp}^f_\pi(K) + \frac{|L|}{|K|+|L|}\mathsf{imp}^f_\pi(L),$$

which immediately gives the desired inequality. ∎

Corollary 5.2 *If K, L are subsets of S such that $K \subseteq L$ and $\pi \in \mathsf{PART}(S)$, then $\mathsf{IMP}^f_\pi(K) \leq \mathsf{IMP}^f_\pi(L)$.*

Proof. Let $H = L - K$. By Theorem 5.1, because K and H are disjoint, we have:

$$\mathsf{IMP}^f_\pi(L) = \mathsf{IMP}^f_\pi(K\cup H) \geq \mathsf{IMP}^f_\pi(K) + \mathsf{IMP}^f_\pi(H),$$

so $\mathsf{IMP}^f_\pi(L) \geq \mathsf{IMP}^f_\pi(K)$. ∎

Let π and σ be two partitions in $\mathsf{PART}(S)$. We write $\pi \leq \sigma$ if each block of the partition π is included in a block of the partition σ; in this case we say that π **is a finer partition than** σ. The following theorem shows that the impurity of a set increases if the partition with respect to which the impurity is computed is finer.

Theorem 5.3 *Let $\pi = \{B_1, \dots, B_n\}$ and $\sigma = \{C_1, \dots, C_m\}$ be two partitions of a set S. If $\pi \leq \sigma$, then $\mathsf{IMP}^f_\sigma(K) \leq \mathsf{IMP}^f_\pi(K)$ for every subset K of S.*

Proof. Because $\pi \leq \sigma$, every block C_j of σ is the union of some blocks of the partition π: $C_j = \bigcup\{B_h \mid h \in H_j\}$, where $H_1, \dots, H_m$ is a partition of the set $\{1, \dots, n\}$.

Therefore,

$$\mathsf{imp}^f_\sigma(K) = \sum_{j=1}^{m} f\left(\frac{|K \cap C_j|}{|K|}\right) \leq \sum_{j=1}^{m} \sum_{h \in H_j} f\left(\frac{|K \cap B_h|}{|K|}\right) = \mathsf{imp}^f_\pi(K),$$

because of the sub-additivity of f. This implies the inequality of the theorem. ▮

Lemma 5.4 *Let $\pi = \{B_1, \dots, B_n\}$, $\zeta = \{D_1, \dots, D_p\}$ be two partitions of a set S. If K is a subset of S such that $K \subseteq D_k$ for some block $D_k \in \zeta$, then $\mathsf{IMP}^f_{\pi \wedge \zeta}(K) = \mathsf{IMP}^f_\pi(K)$.*

Proof. Because the blocks of the partition $\pi \cap \zeta$ have the form $B_h \cap D_j$, we have

$$\mathsf{imp}^f_{\pi \wedge \zeta}(K) = \sum_{h=1}^{n} \sum_{j=1}^{p} f\left(\frac{|K \cap B_h \cap D_j|}{|K|}\right).$$

Each sum $\sum_{j=1}^{p} f\left(\frac{|K \cap B_h \cap D_j|}{|K|}\right)$ contains at most one non-null term, namely

$$f\left(\frac{|K \cap B_h \cap D_k|}{|K|}\right) = f\left(\frac{|K \cap B_h|}{|K|}\right),$$

if $K \cap B_h \neq \emptyset$, which implies the desired equality. ▮

Definition 5.3 *Let S be a finite set and let $\pi, \sigma \in \mathsf{PART}(S)$ be two partitions, where $\pi = \{B_1, \dots, B_n\}$ and $\sigma = \{C_1, \dots, C_m\}$. The* ***impurity of σ with respect to π generated by f*** *is given by*

$$\mathsf{IMP}^f_\pi(\sigma) = \max_{C \in \sigma} \mathsf{IMP}^f_\pi(C). \tag{5.5}$$

A partition σ is ***α-impure with respect to partition*** *π if $\mathsf{IMP}^f_\pi(\sigma) \leq \alpha$.* □

The impurity between two partitions can also be defined using the following formula:

$$\mathsf{IMP}^f_\pi(\sigma) = \sum_{j=1}^{m} \frac{|C_j|}{|S|} \mathsf{IMP}^f_\pi(C_j), \tag{5.6}$$

which takes into consideration the impurities of all blocks of σ with respect to π. Both definitions lead to measures that have similar properties. We use (5.5) rather than (5.6) because, as we will show in Section 5.2, its values are easier to interpret. The form (5.6) was used to define the generalized conditional entropy between two partitions in [SCC00].

Informally, the impurity of a partition σ with respect to a partition π gives us a measure of how well do the blocks of σ fit inside the blocks of π. When $\sigma \leq \pi$, this impurity will be zero. A proof of this is given in the corollary for the following theorem.

Theorem 5.5 *Let S be a finite set and let $\pi, \sigma, \zeta \in \mathsf{PART}(S)$ be partitions of S. We have:*

(i) *if $\sigma \leq \pi$, then $\mathsf{IMP}^f_\pi(\zeta) \leq \mathsf{IMP}^f_\sigma(\zeta)$ and $\mathsf{IMP}^f_\zeta(\sigma) \leq \mathsf{IMP}^f_\zeta(\pi)$;*

(ii) *$\mathsf{IMP}^f_{\pi\wedge\zeta}(\sigma \wedge \zeta) \leq \mathsf{IMP}^f_\pi(\sigma)$.*

Proof. The first part of the Theorem is an immediate consequence of Theorem 5.3.

To prove the second part, assume that D is a block of ζ, $\pi = \{B_1, \ldots, B_n\}$, and $\sigma = \{C_1, \ldots, C_m\}$. Using Lemma 5.4, we have $\mathsf{IMP}^f_{\pi\wedge\zeta}(C_j \cap D) = \mathsf{IMP}^f_\pi(C_j \cap D)$, so $\mathsf{IMP}^f_{\pi\wedge\zeta}(C_j \cap D) \leq \mathsf{IMP}^f_\pi(C_j)$ because of Corollary 5.2. Thus, $\mathsf{IMP}^f_{\pi\wedge\zeta}(C_j \cap D) \leq \mathsf{IMP}^f_\pi(\sigma)$ for every block $C_j \cap D$ of $\sigma \wedge \zeta$ and this implies the second inequality. ∎

Corollary 5.6 *Let σ and π be two partitions of a set S. We have $\sigma \leq \pi$ if and only if $\mathsf{IMP}^f_\pi(\sigma) = 0$.*

Proof. It is easy to see that $\mathsf{IMP}^f_\sigma(\sigma) = 0$. Therefore, by the first part of Theorem 5.5, we have $\mathsf{IMP}^f_\pi(\sigma) = 0$.

Conversely, if $\mathsf{IMP}^f_\pi(\sigma) = 0$, then $\mathsf{IMP}^f_\pi(D) = 0$ for every block D of σ. This implies $f\left(\frac{|B \cap D|}{|D|}\right) = 0$ for every block B of π, so we have either $B \cap D = \emptyset$, or $B \cap D = D$ (i.e., $D \subseteq B$). This implies $\sigma \leq \pi$. ▮

5.2 Purity dependencies

In this section, based on the notion of impurity measure between two partitions, we introduce the notion of purity dependency as a generalization of the concept of functional dependency.

Let $\tau = (T, H, \rho)$ be a table, where T is the name of the table, $H = A_1 \ldots A_n$ is the heading of the table, and $\rho \subseteq \mathsf{Dom}(A_1) \times \cdots \times \mathsf{Dom}(A_n)$, where $\mathsf{Dom}(A_i)$ is the domain of the attribute A_i for $1 \leq i \leq n$. The projection of a tuple $t \in \rho$ on a set of attributes X will be denoted by $t[X]$.

The notion of the active domain of an attribute of a table is extended to sets of attributes as follows. The **active domain** of the set of attributes X of the table τ is the set of all values that appear under X in τ, that is, $\mathsf{aDom}_\tau(X) = \{t[X] \mid t \in \rho\}$. For relational terminology and notations see [Mai83] and [ST95].

For $X \subseteq H$, we define the equivalence $\equiv_X$ on the set of tuples ρ by $u \equiv_X v$ if $u[X] = v[X]$; the corresponding partition of the set of tuples ρ is denoted by π_X. Clearly, π_X is the partition of the tuples of ρ that would be obtained using a `group by` X clause in SQL.

Note that if U, V are two subsets of H such that $U \subseteq V$, then $\pi_V \leq \pi_U$.

Example 5.1 *Our running example is the Mushroom database obtained from the UCI Repository of Machine Learning Databases [BM98]. This dataset*[1] *describes 23 attributes of 8124 different types of North American mushrooms. We adopted this dataset for several reasons: it is well-known and well-documented, its attributes are easy to understand, it has a large enough number of tuples, and it also has more categorical attributes than other UCI datasets. Thus, there was a good chance of finding interesting patterns embedded in the data.*

The class attribute specifies whether a mushroom is edible or poisonous, so $|\mathsf{aDom}(class)| = 2$. *Similarly, an attribute like odor has 7 values: almond, anise, creosote, fishy, foul, musty, none, pungent, and spicy, so the corresponding partition* π_{odor} *has seven blocks.* □

It is easy to see that a table satisfies a functional dependency $X \rightarrow Y$ if and only if $\pi_X \leq \pi_Y$, or equivalently, if and only if $\mathsf{IMP}^f_{\pi_Y}(\pi_X) = 0$, according to Corollary 5.6. This amounts to requiring that every block B of the partition π_X is a π_Y-pure set. Thus, functional dependencies could be generalized by imposing an upper bound α on the impurity of the blocks of the partition π_X relative to the partition π_Y. This suggests the following definition:

Definition 5.4 *Let* $\tau = (T, H, \rho)$ *be a table and let* $X, Y \subseteq H$ *be two sets of attributes.* τ *satisfies the purity dependency* $X \xrightarrow{f,\alpha} Y$ *if* $\mathsf{IMP}^f_{\pi_Y}(\pi_X) \leq \alpha$. □

In other words, the table τ satisfies the purity dependency $X \xrightarrow{f,\alpha} Y$ if the largest f-impurity of a block B of π_X relative to the partition π_Y does not exceed α.

[1] *This Mushroom dataset is not the same as the one used in the experiment from Chapter 3; the dataset used here is the slightly longer version of the Mushroom database.*

Example 5.2 *For the Mushroom database, we are interested in finding purity dependencies of the form* $X \xrightarrow{f,\alpha} class$; *in other words, we are interested in finding sets of attributes* X *such that* $\mathsf{IMP}^{f}_{\pi_{class}}(\pi_X) \leq \alpha$. *Thus, by examining the values of the attributes* X, *we could predict the value of the attribute class with a certain error margin. Of course, given the nature of this dataset, there would be little use in predicting whether a mushroom is edible or poisonous with less than 100% accuracy. As we will show in Section 5.3, however, even though we searched for purity dependencies with an* α *value different than zero, some of the facts that we discovered presented 100% accuracy.*

In a rule of the form $X \xrightarrow{f,\alpha} A$, *with* $|\mathsf{aDom}_\tau(A)| = 2$, *the properties of the blocks of* π_X *depend on the choice of the function* f.

For example, if the Mushroom dataset contained a purity dependency of the form $odor \xrightarrow{f_{gini},\alpha} class$, *then for every set of mushrooms* M *having the same odor attribute value, if* $M_{ed} \subseteq M$ *represents the set of edible mushrooms and* $M_{po} \subseteq M$ *represents the set of poisonous mushrooms, then we have:*

$$\frac{2 \cdot |M_{ed}| \cdot |M_{po}|}{|M_{ed}| + |M_{po}|} \leq \alpha.$$

In other words, the harmonic average of the sizes of M_{ed} *and* M_{po} *does not exceed* α. *Thus, one of the sets has fewer than* α *elements and the other has at least* $|U| - \alpha$ *elements.*

If the Mushroom database satisfies the purity dependency $odor \xrightarrow{f_{peak},\alpha} class$, *then at least one of the sets* M_{ed}, M_{po} *will have size less than* $\alpha/2$, *so the other set will have at least* $|U| - \frac{\alpha}{2}$ *elements.* □

Next, we generalize the Armstrong inference rules for functional dependencies (see [Mai83], [ST95]) to purity dependencies.

Theorem 5.7 *Let $\tau = (T, H, \rho)$ be a table and let X, Y, Z be subsets of H. The following statements hold:*

1. *if τ satisfies $X \rightarrow Y$ and $Y \xrightarrow{f,\alpha} Z$, then τ satisfies $X \xrightarrow{f,\alpha} Z$ (left transitivity property);*
2. *if τ satisfies $X \xrightarrow{f,\alpha} Y$ and $Y \rightarrow Z$, then τ satisfies $X \xrightarrow{f,\alpha} Z$ (right transitivity property);*
3. *if τ satisfies $X \xrightarrow{f,\alpha} Y$, then $X \cup Z \xrightarrow{f,\alpha} Y \cup Z$ (the augmentation property).*

Proof. Suppose that τ satisfies $X \rightarrow Y$ and $Y \xrightarrow{f,\alpha} Z$. Then, we have $\pi_X \leq \pi_Y$ and $\mathsf{IMP}^f_{\pi_Z}(\pi_Y) \leq \alpha$. By Theorem 5.5, we have $\mathsf{IMP}^f_{\pi_Z}(\pi_X) \leq \mathsf{IMP}^f_{\pi_Z}(\pi_Y)$, so $\mathsf{IMP}^f_{\pi_Z}(\pi_X) \leq \alpha$, which justifies the first part of the theorem.

Now, suppose that τ satisfies $X \xrightarrow{f,\alpha} Y$ and $Y \rightarrow Z$, so $\mathsf{IMP}^f_{\pi_Y}(\pi_X) \leq \alpha$ and $\pi_Y \leq \pi_Z$. Again, by Theorem 5.5, we have $\mathsf{IMP}^f_{\pi_Z}(\pi_X) \leq \mathsf{IMP}^f_{\pi_Y}(\pi_X)$, which implies that τ satisfies $X \xrightarrow{f,\alpha} Z$.

To prove the augmentation property observe that $\pi_{U \cup V} = \pi_U \wedge \pi_V$ for all sets of attributes U, V of τ. The second part of Theorem 5.5 implies that

$$\mathsf{IMP}^f_{\pi_{Y \cup Z}}(\pi_{X \cup Z}) = \mathsf{IMP}^f_{\pi_Y \wedge \pi_Z}(\pi_X \wedge \pi_Z) \leq \mathsf{IMP}^f_{\pi_Y}(\pi_X) \leq \alpha,$$

which means that τ satisfies the purity dependency $X \cup Z \xrightarrow{f,\alpha} Y \cup Z$. ∎

Theorem 5.8 *If the table τ satisfies the dependency $X \xrightarrow{f,\alpha} Y$, $X \subseteq X'$, and $Y' \subseteq Y$, then τ satisfies both $X' \xrightarrow{f,\alpha} Y$ and $X \xrightarrow{f,\alpha} Y'$.*

Proof. Because τ satisfies $X \xrightarrow{f,\alpha} Y$ we have $\mathsf{IMP}^f_{\pi_Y}(\pi_X) \leq \alpha$. As we noticed above, $X \subseteq X'$ and $Y' \subseteq Y$ imply $\mathsf{IMP}^f_{\pi_Y}(\pi_{X'}) \leq \mathsf{IMP}^f_{\pi_Y}(\pi_X) \leq \alpha$ and $\mathsf{IMP}^f_{\pi_{Y'}}(\pi_X) \leq \mathsf{IMP}^f_{\pi_Y}(\pi_X) \leq \alpha$, which give the desired conclusions.

Alternatively, the statement follows from the transitivity properties. ∎

The purity dependencies that we introduced are essentially based on the generalization of the notion of entropy formulated in terms of partitions. In a different but related direction, Kivinen and Manilla ([KM95]) studied functional dependencies that are approximately satisfied by tables. They introduced three pairs of measures (denoted by G_i, g_i with $1 \leq i \leq 3$) that evaluate the extent to which a table violates a functional dependency. Specifically, $G_1(X \rightarrow Y, \tau)$ equals the number of pairs of tuples in ρ that violate $X \rightarrow Y$, $G_2(X \rightarrow Y, \tau)$ gives the number of tuples that participate in such a violation, and $G_3(X \rightarrow Y, \tau)$ is the minimum number of tuples that must be removed from ρ to obtain a relation that satisfies $X \rightarrow Y$. The g_i measures are given by $g_1(X \rightarrow Y, \tau) = \frac{G_1(X \rightarrow Y, \tau)}{|\rho|^2}$ and $g_i(X \rightarrow Y, \tau) = \frac{G_i(X \rightarrow Y, \tau)}{|\rho|}$ for $i = 2, 3$. The measures G_1 and G_2 can be expressed using the partitions generated by attribute sets, but they are not impurity measures in the sense used in this chapter. Indeed, let X, Y be two sets of attributes of the table $\tau = (T, H, \rho)$, and let $\pi_X = \{B_1, \ldots, B_n\}$, and $\pi_Y = \{C_1, \ldots, C_m\}$. We can write:

$$G_1(X \rightarrow Y, \tau) = \sum_{i=1}^{n} \sum_{\substack{j,k = 1 \\ j \neq k}}^{m} |B_i \cap C_j| \cdot |B_i \cap C_k|.$$

Similarly, $G_2(X \rightarrow Y, \tau) = \sum\{|B_i| \mid B_i \text{ is } \pi_Y - \text{impure}\}$.

5.3 Approximate classification using purity dependencies

Let X be a set of attributes of a table $\tau = (T, H, \rho)$ and let A be an attribute of the same table. Suppose that the tuples of τ are classified in groups based on the values of A and that we must determine the groups where the tuples belong based on tests on the remaining attributes. Of course, we are interested in using a minimal number of tests in the classification process. This task is frequently encountered in areas such as biology, medicine, social sciences, to name just a few.

We mention here an important difference between this problem and the problem of finding a decision tree: this is the fact that, whereas in a decision tree we can use a large number of attributes on different paths of the decision tree, here we are looking for a fixed set of attributes, whose examination will allow us to classify a tuple. We will look for such sets of attributes that are *minimal* in the sense that any of their subsets will not allow us to perform the classification with the desired precision.

Note that, if τ satisfies the purity dependency $X \xrightarrow{f,\alpha} A$, then the impurity of every group of tuples defined by a common X-value relative to the partition generated by A is less than α; this implies that the A blocks are approximate unions of X-groups.

Theorem 5.8 shows that if τ satisfies the purity dependency $X \xrightarrow{f,\alpha} A$, then τ also satisfies $X' \xrightarrow{f,\alpha} A$ for every superset X' of X. Thus, it is important to determine those sets X that are minimal with respect to set inclusion, such that $X \xrightarrow{f,\alpha} A$. In Algorithm 5.1 we present the pseudocode of an algorithm for finding these minimal sets.

Algorithm 5.1 *(generation of purity dependencies)*

Input: *α and attribute A.*

Output: *the list of all minimal sets of attributes X such that $\mathit{IMP}^{f}_{\pi_A}(\pi_X) \leq \alpha$.*

Uses: *the lists of sets of attibutes $\mathcal{C}$, $\mathcal{M}$, and $\mathcal{N}$.*

```
1 Put into $\mathcal{C}$ all sets of attributes (distinct from $A$) having size one.
```

```
2 While $\mathcal{C}$ is not empty do:
    2.1 For each set of attributes $X$ from $\mathcal{C}$, compute $\mathsf{IMP}^{f}_{\pi_A}(\pi_X)$. If the impurity is less than $\alpha$, then remove $X$ from $\mathcal{C}$ and add it to $\mathcal{M}$.
    2.2 Add to $\mathcal{N}$ the sets of attributes generated by the procedure apriori-gen when applied to $\mathcal{C}$. Clear $\mathcal{C}$. Copy elements of $\mathcal{N}$ into $\mathcal{C}$. Clear $\mathcal{N}$.
3 Return $\mathcal{M}$.
```

■

The *apriori-gen* procedure works for sets of attributes as it worked for itemsets. Note that, like itemsets, the sets of attributes are actually implemented as lists in which the attributes are enumerated in the order in which they appear in the heading of the table.

This algorithm resembles *Apriori*; however, whereas in *Apriori* the frequent set property is hereditary (i.e., is inherited from a set by its subsets), we use in our algorithm the fact that the property $\{X \mid X \subseteq H, X \xrightarrow{f,\alpha} A\}$ contains all the supersets of any of its members, which means that it is dually hereditary.

We implemented in C++ this algorithm and we executed it on the Mushroom database. A was chosen to be the *class* attribute of the database.

The results of an experiment using four different types of functions for different values of α are summarized in Table 5.1.

Among the minimal sets returned by the algorithm when using the f_{peak} generator and an α value of 250, we have the two sets of attributes $\{$*odor*$\}$ and $\{$*cap_color*, *spore_print_color*$\}$.

We verified that the *odor* attribute is indeed a very good classifier criterion. The edible mushrooms are classified using *odor* in one of three sets of cardinalities

	Number of Minimal Sets of Attributes Found				Total number of candidate sets examined			
α	f_{gini}	f_{ent}	f_{peak}	f_{sq}	f_{gini}	f_{ent}	f_{peak}	f_{sq}
2000	46	120	113	18	153	1367	834	57
1750	53	177	109	45	278	2110	1128	160
1500	86	238	121	69	451	3926	1750	290
1250	95	310	142	93	770	5275	2365	524
1000	128	433	264	107	1217	9062	4101	952
750	170	530	332	216	2905	16897	8295	2387
500	376	914	434	425	7559	39659	20017	7742
250	796	1434	664	984	33189	126141	76257	35274

Table 5.1: Results for Mushroom database

400, 400, and 3408. The poisonous mushrooms are classified using *odor* in one of seven blocks of cardinalities 192, 2160, 36, 120, 256, 576, and 576. There is only one set characterized by *odor* that contains both edible and poisonous mushrooms, and this set corresponds to the value ”none” (no odor) and contains 3408 edible mushrooms and 120 poisonous mushrooms. This supports the observation made in Example 5.2, because the smaller intersection — the 120 poisonous mushrooms — is less than half of the value of α (which was 250 in this example). We can conclude that we can classify very well mushrooms based on *odor*, the classification needing to be refined only in the case of odorless mushrooms. Note that, even though we searched for rules that were impure, we discovered properties that hold with 100 percent accuracy for the recorded instances.

For the pair of attributes, *cap_color* and *spore_print_color*, the edible mushrooms are classified in 25 sets. The poisonous mushrooms are classified in 20 sets. There are 11 sets characterized by particular values of attributes *cap_color* and *spore_print_color* that contain both edible and poisonous mushrooms, and 23 sets that contain only edible or poisonous mushrooms. The sets for which the classification is ambiguous are presented in Table 5.2 and the exact classifications are given in Table 5.3.

cap_color	*spore_print_color*	edible	poisonous
cinnamon	white	32	12
red	white	48	876
gray	black	440	32
gray	brown	440	32
brown	black	488	64
brown	brown	536	64
brown	white	96	892
white	chocolate	16	96
white	black	256	96
white	brown	280	96
white	white	144	8

Table 5.2: Approximate mushroom classification resulting from the analysis of attributes *cap_color* and *spore_print_color*

As Table 5.2 shows, the ambiguous classifications tend to contain predominantly either poisonous or edible mushrooms. Still, we discovered 23 exact classifications, which are shown in Table 5.3, and each of these can also be viewed as an exact association rule. Some of these association rules have significant support, for example, the rule $\{cap\_color = gray, spore\_print\_color = chocolate\} \Rightarrow \{class = poisonous\}$ is supported by 744 instances.

To compare the results obtained using various generator functions in constructing the set of all minimal sets of attributes X, with $X \xrightarrow{f,\alpha} A$, consider the set $\mathcal{G}$ of all generator functions and define an equivalence $\sim$ on the set $\mathcal{G} \times \mathbb{R}_+$ by $(f, \alpha) \sim (f', \alpha')$ if

$$\frac{\alpha}{f\left(\frac{1}{|\mathsf{aDom}(A)|}\right)} = \frac{\alpha'}{f'\left(\frac{1}{|\mathsf{aDom}(A)|}\right)}.$$

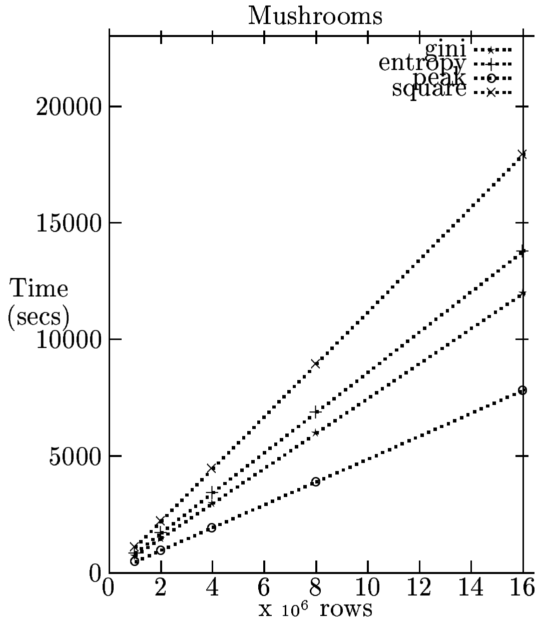


Figure 5.2: Graphical plot of execution time versus number of tuples

If $(f, \alpha) \sim (f', \alpha')$, then it is easy to see that we have $m^{f}_{L,\pi} \leq \alpha$ if and only if $m^{f'}_{L,\pi} \leq \alpha'$, where $m^{f}_{L,\pi}$ was introduced by Equation 5.4. This gives us a base for comparing the results obtained for several generator functions. For example, in the case of a binary attribute A, the partition π_A has two blocks. Therefore, we have $(f, \alpha) \sim (f', \alpha')$ if $\frac{\alpha}{f(0.5)} = \frac{\alpha'}{f'(0.5)}$. Thus, the experimental results obtained in determining the purity dependencies of the form $X \xrightarrow{f,\alpha} A$ for $\alpha = 1000$ and $f = f_{\mathsf{gini}}$ are comparable to results obtained in other experiments for values of α described in Table 5.4.

Our algorithm is scalable with respect to the number of tuples as shown by the results from Figure 5.2. In this experiment we sought to determine the minimal sets of attributes X that satisfy a purity dependency $X \xrightarrow{f,\alpha} A$ for a fixed A, by increasing the threshold α in proportion with the number of tuples. The set of purity dependencies was kept constant across these experiments by replicating the initial dataset for a sufficient number of times to achieve the desired number of tuples.

5.4 Conclusions

In this chapter we introduced purity dependencies as generalizations of functional dependencies by using the notion of relative impurity of partitions. They can also be regarded as reflecting an "approximate" satisfaction of functional dependencies by tables in relational databases. Regardless of the generator function used for the impurity measure, purity dependencies have properties that are similar to Armstrong's rules of inference for functional dependencies, as shown by Theorem 5.7.

The approach to classification that we propose in this paper generalizes well-known classification techniques in data mining such as the one proposed in the CART system which is based on the Gini impurity measure (see [BFO84]). Algorithm 5.1 is reasonably fast and is scalable with respect to the size of the database.

cap_color	*spore_print_color*	class	number
buff	chocolate	poisonous	96
buff	green	poisonous	24
buff	white	edible	48
red	black	edible	288
red	brown	edible	288
gray	chocolate	poisonous	744
gray	white	edible	152
brown	buff	edible	48
brown	orange	edible	48
brown	yellow	edible	48
pink	black	poisonous	32
pink	brown	poisonous	32
pink	green	poisonous	24
pink	white	edible	56
green	chocolate	edible	16
purple	chocolate	edible	16
white	green	poisonous	24
white	purple	edible	24
yellow	chocolate	poisonous	648
yellow	black	edible	176
yellow	brown	edible	200
yellow	purple	edible	24
yellow	white	poisonous	24

Table 5.3: Exact mushroom classification resulting from the analysis of attributes *cap_color* and *spore_print_color*

	f_{gini}	f_{ent}	f_{peak}	f_{sq}
α	1000	2000	2000	828

Table 5.4: Values of α that determine comparable results for different types of generators

CHAPTER 6

Conclusions

Although data mining has come a long way since the term was coined at the beginning of the 1990's, there are many problems that are still open. For this reason, data mining is an exciting field of study, and its importance will certainly grow in the next decade. It is both surprising and pleasing to see that new algorithms continue to appear for mining association rules, seven years after the introduction of the problem in [AIS93b]. This shows that DM problems are far from being exhausted, even if they have been around for some time. Because new applications can appear for any new type of data, data mining is a field that will continue to attract the interest of researchers.

In this thesis we have presented our contributions to the problems of mining association rules and generalizing the concept of functional dependency.

6.1 Mining association rules

In Chapter 2 we have introduced two new algorithms: *Closure* and *MaxClosure*, for mining frequent itemsets and maximal frequent itemsets, respectively. These algorithms scale linearly relative to the size of the input database. The *MaxClosure* algorithm is recommended when the time required by traditional algorithms that mine all frequent itemsets, like *Apriori* and *Closure*, becomes too large.

After the mining of the frequent itemsets, the generation of association rules raises another problem. Interesting rules can be hidden among thousands of less relevant rules. This can be regarded as a secondary data mining problem, because we now want to extract the interesting rules from the set of all possible associations. Our approach, presented in Chapter 3, consists of finding a subset of all association rules — the cover — from which it is possible to infer all other rules by using the inference rule from Theorem 3.2. As shown by the results of our experiments, the cover can be computed efficiently using algorithm *CoverRules*, is considerably smaller than the set of association rules, and it can summarize better the sets of associations that contain more redundant rules. The cover is also smaller than the Guigues-Duquenne–Luxenburger basis, which was a similar concept introduced in [PBT99a].

The mining of association rules in multiple-table databases is a problem that has received less attention. In Chapter 4, we have analyzed the special but important case of mining a star schema database. The algorithms that we introduced, *AprioriJoin* and *AprioriStar*, both scale relative to the database size. The mining of arbitrary database schemas raises more problems, due to the necessity of computing the support of an itemset relative to each *connector* that links its corresponding entities in the schema. Therefore, this can represent an interesting future research direction.

6.2 Purity dependencies

The purity dependency concept that we introduced in Chapter 5 as a generalization of the notion of functional dependency can be used for performing approximate classifications. As shown by the results of our experiments, knowing what purity dependencies exist in data can lead us to the discovery or rules that hold with 100 percent accuracy. It is also useful to have such a measure for quantifying the degree in which the values of a set of attributes can be determined by examining the values of another set of attributes that is disjoint relative to the first.

The relative impurity of two sets of attributes also forms the basis for defining the generalized conditional entropy between the partitions determined by the respective sets of attributes, with applications to clustering [SCC00].

The results of our experiments show that similar purity dependencies can be obtained using various generators, as long as we adjust the required impurity value accordingly. An interesting future research topic would be to determine a criteria for selecting one generator function over another or for determining in what situations a generator function is better suited than another.

References

[AAP00] Ramesh C. Agarwal, Charu C. Aggarwal, and V.V.V. Prasad. "Depth First Generation of Long Patterns." In *Proceedings of the 6th ACM-SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 108–118, 2000.

[AAP01] Ramesh C. Agarwal, Charu C. Aggarwal, and V.V.V. Prasad. "A Tree Projection Algorithm for Generation of Frequent Itemsets." *Journal of Parallel and Distributed Computing, Special Issue on High Performance Data Mining*, **61**(3):350–371, 2001.

[AIS93a] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. "Database Mining: A Performance Perspective." *IEEE Transactions on Knowledge and Data Engineering, Special Issue on Learning and Discovery in Knowledge-Based Databases*, pp. 914–925, 1993.

[AIS93b] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. "Mining Association Rules between Sets of Items in Large Databases." In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pp. 207–216, 1993.

[AMS96] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, and A. Inkeri Verkamo. "Fast Discovery of Association Rules." In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pp. 307–328. AAAI/MIT Press, Menlo Park, 1996.

[AS94a] Rakesh Agrawal and Ramakrishnan Srikant. "Fast Algorithms for Mining Association Rules." RJ 9839, IBM Almaden Research Center, Almaden, California, 1994.

[AS94b] Rakesh Agrawal and Ramakrishnan Srikant. "Fast Algorithms for Mining Association Rules in Large Databases." In *Proceedings of the 20th International Conference on Very Large Databases*, pp. 487–499, 1994.

[AS95a] Rakesh Agrawal and Ramakrishnan Srikant. "Mining Generalized Association Rules." In *Proceedings of the 21st International Conference on Very Large Databases*, pp. 407–419, 1995.

[AS95b] Rakesh Agrawal and Ramakrishnan Srikant. "Mining Sequential Patterns." RJ 9910, IBM Almaden Research Center, Almaden, California, 1995.

[AS95c] Rakesh Agrawal and Ramakrishnan Srikant. "Mining Sequential Patterns." In *Proceedings of the 11th International Conference on Data Engineering*, pp. 3–14, 1995.

[AS96a] Rakesh Agrawal and Ramakrishnan Srikant. "Mining Quantitative Association Rules in Large Relational Tables." In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pp. 1–12, 1996.

[AS96b] Rakesh Agrawal and Ramakrishnan Srikant. "Mining Sequential Patterns: Generalizations and Performance Improvements." In *Proceedings of the 5th International Conference on Extending Database Technology*, pp. 3–17, 1996.

[BA99] Roberto J. Bayardo and Rakesh Agrawal. "Mining the Most Interesting Rules." In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 145–154, 1999.

[Bay98] Roberto J. Bayardo. "Efficiently Mining Long Patterns from Databases." In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pp. 85–93, 1998.

[BFO84] L. Breiman, J. H. Friedman, R. A. Ohlsen, and C. J. Stone. *Classification and Regression Trees.* Chapman & Hall/CRC, Boca Raton, 1984. Republished 1993.

[Bir73] Garrett Birkhoff. *Lattice Theory.* American Mathematical Society Colloquium Publications, Rhode Island, 1973.

[BM98] C. L. Blake and C. J. Merz. "University of California, Irvine: Repository of machine learning databases.", 1998. http://www.ics.uci.edu/~mlearn/MLRepository.html.

[BMU97] Sergey Brin, Rajeev Motwani, Jeffrey Ullman, and Shalom Tsur. "Dynamic Itemset Counting and Implication Rules for Market Basket Data." In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pp. 255–264, 1997.

[BPT00] Yves Bastide, Nicolas Pasquier, Rafik Taouil, Gerd Stumme, and Lotfi Lakhal. "Mining Minimal Non-Redundant Association Rules using Frequent Closed Itemsets." In *Proceedings of the First International Conference on Computational Logic*, pp. 972–986, 2000.

[CCS00] Dana Cristofor, Laurentiu Cristofor, and Dan A. Simovici. "Galois Connections and Data Mining." *Journal of Universal Computer Science*, **6**(1):60–73, 2000.

[CGL01] Frans Coenen, Graham Goulbourne, and Paul H. Leng. "Computing Association Rules Using Partial Totals." In *Proceedings of the 5th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pp. 54–66, 2001.

[CHH99] Don Coppersmith, Se June Hong, and Jonathan R.M. Hosking. "Partitioning Nominal Attributes in Decision Trees." *Data Mining and Knowledge Discovery*, **3**(2):197–217, 1999.

[Cri02] Laurentiu Cristofor. "ARtool: Association Rule Mining Algorithms and Tools.", 2002. http://www.cs.umb.edu/∼laur/ARtool/.

[CS01] Laurentiu Cristofor and Dan A. Simovici. "Mining Association Rules in Entity-Relationship Modeled Databases." TR 01-1, University of Massachusetts at Boston, Boston, Massachusetts, 2001.

[CS02] Laurentiu Cristofor and Dan A. Simovici. "Generating an informative cover for association rules." TR 02-1, University of Massachusetts at Boston, Boston, Massachusetts, 2002.

[DR97] Luc Dehaspe and Luc De Raedt. "Mining Association Rules in Multiple Relations." In *Proceedings of the 7th International Workshop on Inductive Logic Programming*, pp. 125–132, 1997.

[GCL00] Graham Goulbourne, Frans Coenen, and Paul H. Leng. "Algorithms for Computing Association Rules Using a Partial-Support Tree." *Knowledge Based Systems*, **13**(2-3):141–149, 2000.

[GD86] J.L. Guigues and V. Duquenne. "Familles minimales d'implications informatives résultant d'un tableau de données binaires." *Mathématiques et Sciences Humaines*, **24**(95):5–18, 1986.

[GHK80] Gerhard Gierz, Karl Heinrich Hoffman, Klaus Keimel, Jimmie D. Lawson, Michael W. Mislove, and Dana S. Scott. *A Compendium of Continuous Lattices.* Springer-Verlag, Berlin, 1980.

[GKM97] Dimitrios Gunopulos, Roni Khardon, Heikki Mannila, and Hannu Toivonen. "Data Mining, Hypergraph Transversal, and Machine Learning." In *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 209–216, 1997.

[GT66] S. Guiasu and R. Theodorescu. *Mathematical Information Theory (in Romanian).* Editura Academiei, Bucharest, 1966.

[HH99] Robert J. Hilderman and Howard J. Hamilton. "Knowledge discovery and interestingness measures: A survey." Technical Report CS 99-04, University of Regina, Regina, Saskatchewan, Canada, 1999.

[HKP98] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. "Efficient Discovery of Functional and Approximate Dependencies Using Partitions." In *Proceedings of the 14th International Conference on Data Engineering*, pp. 392–401, 1998.

[HPY00] Jiawei Han, Jian Pei, and Yiwen Yin. "Mining Frequent Patterns without Candidate Generation." In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pp. 1–12, 2000.

[IU62] R. S. Ingarden and K. Urbanik. "Information without Probability." *Coll. Math.*, **1**:281–304, 1962.

[JS00] Viviane Crestana Jensen and Nandit Soparkar. "Frequent Itemset Counting Across Multiple Tables." In *Proceedings of the 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 49–61, 2000.

[JS01] Szymon Jaroszewicz and Dan A. Simovici. "A General Measure of Rule Interestingness." In *Proceedings of the 5th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pp. 253–265, 2001.

[KBS99] Arno J. Knobbe, Hendrik Blockeel, Arno Siebes, and Daniel M.G. van der Wallen. "Multi-Relational Data Mining." INS-R 9908, CWI, The National Research Institute for Mathematics and Computer Science in the Netherlands, Amsterdam, Netherlands, 1999.

[Khi56] A. Ia. Khinchin. "On The Fundamental Theorem of Information Theory (in Russian)." *Usp. Mat. Nauk*, **11**:17–75, 1956.

[Khi57] A. Ia. Khinchin. *Mathematical Foundations of Information Theory.* Dover Publ., New York, 1957.

[KM95] Jyrki Kivinen and Heikki Mannila. "Approximate Dependency Inference from Relations." *Theoretical Computer Science*, **149**(1):129–149, 1995.

[KSW99] Arno J. Knobbe, Arno Siebes, and Daniel M.G. van de Wallen. "Multi-Relational Decision Tree Induction." In *Proceedings of the 3rd European Conference on Principles and Practice of Knowledge Discovery in Databases*, pp. 378–383, 1999.

[LHM99] Bing Liu, Wynne Hsu, and Yiming Ma. "Pruning and Summarizing the Discovered Associations." In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 125–134, 1999.

[Lux91] Michael Luxenburger. "Implications partielles dans un contexte." *Mathématiques, Informatique et Sciences Humaines*, **29**(113):35–55, 1991.

[Mai83] David Maier. *The Theory of Relational Databases.* Computer Science Press, Rockville, Maryland, 1983.

[Mit97] Tom M. Mitchell. *Machine Learning.* WCB McGraw-Hill, New York, 1997.

[MR75] A. M. Mathai and P. N. Rathie. *Basic Concepts in Information Theory and Statistics — Axiomatic Foundations and Applications.* Halsted Press, John Wiley & Sons, New York, 1975.

[Mue95] Andreas Mueller. "Fast Sequential and Parallel Algorithms for Association Rule Mining: A Comparison." CS-TR 3515, University of Maryland, College Park, Maryland, 1995.

[PBT99a] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. "Closed Set Based Discovery of Small Covers for Association Rules." In *Proceedings of the 15th Conference on Advanced Databases*, pp. 361–381, 1999.

[PBT99b] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. "Discovering Frequent Closed Itemsets for Association Rules." In *Proceedings of the 7th International Conference on Database Theory*, pp. 398–416, 1999.

[PBT99c] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. "Efficient Mining of Association Rules Using Closed Itemset Lattices." *Information Systems*, **24**(1):25–46, 1999.

[PT98] Balaji Padmanabhan and Alexander Tuzhilin. "A Belief-Driven Method for Discovering Unexpected Patterns." In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, pp. 94–100, 1998.

[PT00] Balaji Padmanabhan and Alexander Tuzhilin. "Small is beautiful: discovering the minimal set of unexpected patterns." In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 54–63, 2000.

[Qui93] John R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufman, San Mateo, 1993.

[SCC00] Dan A. Simovici, Dana Cristofor, and Laurentiu Cristofor. "Generalized Entropy and Projection Clustering of Categorical Data." In *Proceedings of the 4th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pp. 619–625, 2000.

[SCC02a] Dan A. Simovici, Dana Cristofor, and Laurentiu Cristofor. "Impurity Measures in Databases." *Acta Informatica*, **38**(5):307–324, 2002.

[SCC02b] Dan A. Simovici, Dana Cristofor, and Laurentiu Cristofor. "Mining for Purity Dependencies in Databases." In *Proceedings of Journées Francophones d'Extraction et de Gestion des Connaissances*, pp. 257–268, 2002.

[SJ01] Dan A. Simovici and Szymon Jaroszewicz. "An Axiomatization of Generalized Entropy of Partitions." In *Proceedings of the 31st IEEE International Symposium on Multiple-Valued Logic*, pp. 259–266, Warsaw, Poland, 2001.

[SLR99] Devavrat Shah, Laks V. S. Lakshmanan, Krithi Ramamritham, and S. Sudarshan. "Interestingness and Pruning of Mined Patterns." In *1999 ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 1999.

[SON95] Ashoka Savasere, Edward Omiecinski, and Shamkant B. Navathe. "An Efficient Algorithm for Mining Association Rules in Large Databases." In *Proceedings of the 21st International Conference on Very Large Databases*, pp. 432–444, 1995.

[ST95] Dan A. Simovici and Richard L. Tenney. *Relational Database Systems.* Academic Press, New York, 1995.

[TKR95] Hannu Toivonen, Mika Klemettinen, Pirjo Ronkainen, Kimmo Hätönen, and Heikki Mannila. "Pruning and Grouping Discovered Association Rules." In *Proceedings of the European Conference on Machine Learning Workshop on Statistics, Machine Learning, and Knowledge Discovery in Databases*, pp. 47–52, 1995.

[Zak00] Mohammed J. Zaki. "Generating Non-Redundant Association Rules." In *Proceedings of the 6th ACM-SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 34–43, 2000.

[ZH99] Mohammed J. Zaki and Ching-Jui Hsiao. "ChARM: An Efficient Algorithm for Closed Association Rule Mining." Technical Report 99-10, Renssaeler Polytechnic Institute, Troy, New York, 1999.

[ZH02] Mohammed J. Zaki and Ching-Jui Hsiao. "CHARM: An Efficient Algorithm for Closed Itemset Mining." In *Proceedings of the 2nd SIAM International Conference on Data Mining*, pp. 457–473, 2002.

[ZO98] Mohammed J. Zaki and Mitsunori Ogihara. "Theoretical Foundations of Association Rules." In *Proceedings of the 3rd SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pp. 7:1–7:8, 1998.

[ZPO97a] Mohammed J. Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. "New Algorithms for Fast Discovery of Association Rules." In *Proceedings of the AAAI International Conference on Knowledge Discovery in Databases*, pp. 283–286, 1997.

[ZPO97b] Mohammed J. Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. "New Algorithms for Fast Discovery of Association Rules." TR 651, University of Rochester, Rochester, New York, 1997.