Holon Academic Institute of Technology
Department of Computer Science

A Final Project in:

A B.Sc.T.E Degree in Computer Science

Mining Association and Inverse Association Rules in Large Databases

Supervisor: Prof. M. Schneider

Written by:
Oren Nahum (0-2725262-6)
Tali Ambar Orad (0-3841350-6)

June 2001

המחלקה למדעי המחשב

חיפוש קשרים וקשרים הפוכים במסדי נתונים

מאת

אורן נחום (0-2725262-6)
טלי אמבר אורעד (0-3841350-6)

Mining Association and Inverse Association Rules in Databases

By
Oren Nahum and Tali Ambar Orad
Dep. of Computer Science
Holon Academic Institute of Technology
Israel

Table of content

Table of content 3
Abstract 5
1. Knowledge Discovery in Databases and Data Mining 5
  1.1. Introduction 5
  1.2. Links Between KDD and Related Fields 6
  1.3. A Definition of Knowledge Discovery in Databases 6
  1.4. The KDD Process 8
  1.5. An Overview of Data Mining Methods 9
    1.1.1. The Primary Tasks of Data Mining 10
    1.1.2. The Components of Data Mining Algorithms 11
  1.6. A Discussion of Popular Data Mining Methods 12
    1.6.1. Decision Trees and Rules 12
    1.6.2. Nonlinear Regression and Classification Methods 13
    1.6.3. Example-based Methods 13
    1.6.4. Probabilistic Graphical Dependency Models 14
    1.6.5. Relational Learning Models 14
  1.7. Application Issues 15
    1.7.1. Guidelines for Selecting a Potential KDD Application 16
    1.7.2. Privacy and Knowledge Discovery 17
    1.7.3. Research and Application Challenges for KDD 18
2. Association Rules 20
  2.1. Introduction to Association Rules 20
  2.2. Formal Model 20
  2.3. Related work 21
  2.4. Discovering Large Itemsets 22
    2.4.1. AIS Algorithm 22
    2.4.2. Apriori Algorithm 24
  2.5. Algorithm for Mining Association Rules 25
  2.6. Example 1 26
3. Inverse Associations 31
  3.1. Formal Model 31
  3.2. Algorithm for Mining Inverse Association Rules 32
  3.3. Example 2 32
4. Implementation of the Algorithms 43
  4.1. Data Insertion 43
  4.2. Resizing the Data Table 44

4.3. Algorithms 45
4.4. The Mining Procedure 45
4.5. Reports 50
4.6. The About Box, and More 51
5. The Source Code 53
5.1. DataInuptUnit Unit 53
5.2. NativeUnit Unit 62
5.3. AprioriUnit Unit 69
5.4. FinalAssociationUnit unit 78
5.5. ProgressUnit Unit 82
5.6. ResizeTableUnit Unit 83
5.7. AboutUnit Unit 85
5.8. Sets Unit 86
5.9. LinkedList Unit 92
5.10. HashTable Unit 94
6. Summary 98
6.1. Future Work 98
7. Acknowledgments 100
8. References 101

Abstract

In this paper, we introduce the concept of *association rules*, which is a part of a larger filed, called *data mining*. We show algorithms developed by Ageawal and his team, from the IBM Almaden Research Center, to discover association among data in large databases, we also consider an extension to that algorithm, to discover *inverse association rules*, and implementation of both algorithms.

1. Knowledge Discovery in Databases and Data Mining[1]

1.1. Introduction

The starve for knowledge is the main characteristic of the human being. The path to knowledge lies through accumulating data and interpreting it. Throughout the years, we have accumulated a huge amount of data. Advances in data collection, the computerization of many business and government transactions, the width spread use of credit cards and so on, have flooded us with information, and created a great demand for new techniques and powerful tools that can intelligently and automatically assist us in transforming this data into useful task-oriented knowledge. These tools and techniques are the subject of the rapidly emerging field of *knowledge discovery in databases* (KDD).

Examples for this growing amount of data are easy to find. Most health-care transactions in the U.S. are being stored in computers, yielding multi-gigabyte databases, which many large companies are beginning to analyze in order to control costs and improve quality. There are huge scientific databases as well. The human genome database project has collected gigabytes of data on the human genetic code and much more is expected. A database housing a sky object catalog from a major astronomy sky survey consists of billions of entries with raw image data sizes measured in terabytes. The NASA Earth Observing System of orbiting satellites and other spaceborne instruments is projected to generate on the order of 50 gigabytes of remotely sensed image data per *hour* when operational in the late 1990s and early in the next century.

This huge amount of data, obviously, cannot be treated with the traditional manual methods of data analysis such as spreadsheets. Those methods can create reports, such as distribution analysis, but they can’t analyze the contents of those reports to focus us on important knowledge. In efforts to satisfy the need for new powerful tools and techniques, researchers have been exploring ideas and methods developed in machine learning, pattern recognition, statistical data analysis, data visualization, neural nets, etc. Those tools and techniques has the ability to intelligently and automatically assist humans in analyzing the growing amount of data, and they are the subject of the emerging field of knowledge discovery in databases (KDD).

Historically the notion of finding useful patterns in raw data has been given various names, including knowledge discovery in databases, data mining, knowledge

[1] Mostly based on [6].

extraction, information discovery, information harvesting, data archaeology, and data pattern Processing. The term *knowledge discovery in databases,* or KDD for short, was coined in 1989 to refer to the broad process of finding knowledge in data, and to emphasize the "high-level" application of particular *data mining* methods. The term *data mining* has been commonly used by statisticians, data analysts and the MIS (Management Information Systems) community, while KDD has been mostly used by artificial intelligence and machine learning researchers. In this paper we adopt the view that KDD refers to the overall *process* of discovering useful knowledge from data while *data mining* refers to the application of algorithms for extracting patterns from data without the additional steps of the KDD process.

1.2. Links Between KDD and Related Fields

KDD is of interest to researchers in machine learning, pattern recognition, databases, statistics, artificial intelligence, knowledge acquisition for expert systems, and data visualization. KDD systems typically draw upon methods, algorithms, and techniques from these diverse fields. The unifying goal is extracting knowledge from data in the context of large databases.

In the fields of machine learning and pattern recognition, overlap with KDD lies in the study of theories and algorithms for systems which extract patterns and models from data (mainly data mining methods). KDD focuses on the extension of these theories and algorithms to the problem of finding special patterns (ones that may be interpreted as *useful or interesting knowledge*) in large sets of real-world data. KDD also has much in common with statistics, particularly exploratory data analysis (EDA). KDD systems often embed particular statistical procedures for modeling data and handling noise within an overall knowledge discovery framework.

Machine discovery which targets the discovery of empirical laws from observation and experimentation, and *causal modeling* for the inference of causal models from data are related research areas.

Another related area is *data warehousing,* which refers to the recently popular MIS trend for collecting and cleaning transactional data and making them available for on-line retrieval. A popular approach for analysis of data warehouses has been called OLAP *(on-line analytical processing),* after a new set of principles proposed by Codd (1993). OLAP tools focus on providing multi-dimensional data analysis, which is superior to SQL (standard query language) in computing summaries and breakdowns along many dimensions. We view both knowledge discovery and OLAP as related facets of a new generation of intelligent information extraction and management tools.

1.3. A Definition of Knowledge Discovery in Databases

We first start with a general statement of this definition in words:

Knowledge discovery in databases is the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data.

Let us examine these terms in more detail.

- *Data* is a set of facts F (e.g., cases in a database).
- *Pattern* is an expression E in a language L describing facts in a subset F_E of F. E is called a pattern if it is simpler (in some sense, see below) than the enumeration of all facts in F_E. For example, the pattern: "If income<\$*t*, then person has defaulted on the loan" would be one such pattern for an appropriate choice of t.
- *Process:* Usually in KDD, process is a multi-step process, which involves data preparation, search for patterns, knowledge evaluation, and refinement involving iteration after modification. The process is assumed to be non-trivial - that is, to have some degree of search autonomy.
- *Validity:* The discovered patterns should be valid on new data with some degree of certainty. A measure of certainty is a function C mapping expressions in L to a partially or totally ordered measurement space M_C. An expression E in L about a subset $F_E \subset F$ can be assigned a certainty measure $c=C(E,F)$.
- *Novel:* The patterns are novel (at least to the system). Novelty can be measured with respect to changes in data (by comparing current values to previous or expected values) or knowledge (how a new finding is related to old ones). In general, we assume this can be measured by a function $N(E,F)$, which can be a boolean function or a measure of degree of novelty or unexpectedness.
- *Potentially Useful:* The patterns should potentially lead to some useful actions, as measured by some utility function. Such a function U maps expressions in L to a partially or totally ordered measure space M_u: hence, $u=U(E,F)$.
- *Ultimately Understandable:* A goal of KDD is to make patterns understandable to humans in order to facilitate a better understanding of the underlying data. While this is difficult to measure precisely, one frequent substitute is the simplicity measure. Several measures of simplicity exist, and they range from the purely syntactic (e.g., the size of a pattern in bits) to the semantic (e.g., easy for humans to comprehend in some setting). We assume this is measured, if possible, by a function S mapping expressions E in L to a partially or totally ordered measure space M_s: hence, $s= S(E,F)$.

An important notion, called *interestingness,* is usually taken as an overall measure of pattern value, combining validity, novelty, usefulness, and simplicity. Some KDD systems have an explicit interestingness function $i=I(E,F,C,N,U,S)$ which maps expressions in L to a measure space M_I. Other systems define interestingness indirectly via an ordering of the discovered patterns.

The purpose of this definition given above, is specify what an algorithm used in a KDD process may consider knowledge.

- *Knowledge:* A pattern $E \in L$ is called knowledge if for some user-specified threshold $i \in M_I$, $I(E,F,C,N,U,S) > i$.

Note that this definition of knowledge is by no means absolute. In fact, it is purely user-oriented, and determined by whatever functions and thresholds the user chooses. For example, one instantiation of this definition is to select some thresholds $c \in M_C$,

$s \in M_S$, and $u \in M_U$, and calling a pattern E knowledge if and only if $C(E,F)>c$ and $SIE,F)>s$ and $U(S,F)>u$.

By appropriate settings of thresholds, one can emphasize accurate predictors or useful (by some cost measure) patterns over others. Clearly, there is an infinite space of how the mapping I can be defined. Such decisions are left to the user and the specifics of the domain.

Data Mining is a step in the KDD process consisting of particular data mining algorithms that, under some acceptable computational efficiency limitations, produces a particular enumeration of patterns Ej over F.

Note that the space of patterns is often infinite, and the enumeration of patterns involves some form of search in this space. The computational efficiency constraints place severe limits on the subspace that can be explored by the algorithm.

KDD Process is the process of using data mining methods (algorithms) to extract (identify) what is deemed knowledge according to the specifications of measures and thresholds, using the database F along with any required preprocessing, subsampling, and transformations of F.

The data mining component of the KDD process is mainly concerned with means by which patterns are extracted and enumerated from the data. Knowledge discovery involves the *evaluation* and possibly *interpretation* of the patterns to make the decision of what constitutes knowledge and what does not. It also includes the choice of encoding schemes, preprocessing, sampling, and projections of the data prior to the data-mining step.

1.4. The KDD Process

The KDD process is interactive and iterative, involving numerous steps with many decisions being made by the user. Brachman & Anand ([8]) give a practical view of the KDD process emphasizing the interactive nature of the process. Here we broadly outline some of its basic steps:

1. Developing an understanding of the application domain, the relevant prior knowledge, and the goals of the end-user.
2. Creating a target data set: selecting a data set, or focusing on a subset of variables or data samples, on which discovery is to be performed.
3. Data cleaning and preprocessing: basic operations such as the removal of noise or outliers if appropriate, collecting the necessary information to model or account for noise, deciding on strategies for handling missing data fields, accounting for time sequence information and known changes.
4. Data reduction and projection: finding useful features to represent the data depending on the goal of the task. Using dimensionality reduction or transformation methods to reduce the effective number of variables under consideration or to find invariant representations for the data.

5. Choosing the data mining task: deciding whether the goal of the KDD process is classification, regression, clustering, etc.
6. Choosing the data mining algorithm(s): selecting method(s) to be used for searching for patterns in the data. This includes deciding which models and parameters may be appropriate (e.g. models for categorical data are different than models on vectors over the reals) and matching a particular data mining method with the overall criteria of the KDD process (e.g. the end-user may be more interested in understanding the model than its predictive capabilities).
7. Data mining: searching for patterns of interest in a particular representational form or a set of such representations: classification rules or trees, regression, clustering, and so forth. The user can significantly aid the data mining method by correctly performing the preceding steps.
8. Interpreting mined patterns, possible return to any of steps 1-7 for further iteration.
9. Consolidating discovered knowledge: incorporating this knowledge into the performance system, or simply documenting it and reporting it to interested parties. This also includes checking for and resolving potential conflicts with previously believed (or extracted) knowledge.

The KDD process can involve significant iteration and may contain loops between any two steps. The basic flow of steps (although not the potential multitude of iterations and loops) is illustrated in Figure 1. Most previous work on KDD has focused on step 7 - the data mining. However, the other steps are of considerable importance for the successful application of KDD in practice.

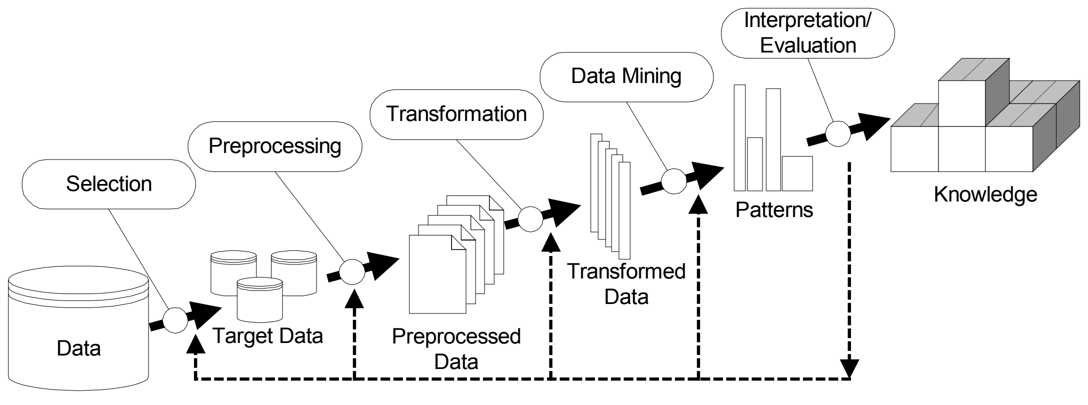


Figure 1 – An Overview of the steps comprising the KDD process

Having defined the basic notions and introduced the KDD process, we now focus on the data mining component, which has by far received the most attention in the literature.

1.5. An Overview of Data Mining Methods

The data mining component of the KDD process often involves repeated iterative application of particular data mining methods. The objective of this section is to present a unified overview of some of the most popular data mining methods in current use. We use the terms *patterns* and *models* loosely throughout this chapter: a pattern can be thought of as instantiation of a model, e.g., $f(x)=3x^2+x$ is a pattern whereas $f(z)=\alpha x^2+\beta x$ is considered a model.

Data mining involves fitting models to, or determining patterns from, observed data. The fitted models play the role of inferred knowledge: whether or not the models reflect *useful* or *interesting* knowledge is part of the overall, interactive KDD process where subjective human judgment is usually required. There are two primary mathematical formalisms used in model fitting: the *statistical* approach allows for non-deterministic effects in the model (for example, $f(x)=\alpha x+e$ where e could be a Gaussian random variable), whereas a *logical* model is purely deterministic ($f(x)=\alpha x$) and does not admit the possibility of uncertainty in the modeling process. We will focus primarily on the statistical/probabilistic approach to data mining: this tends to be the most widely-used basis for practical data mining applications given the typical uncertainty about the exact nature of real-world data-generating processes.

Most data mining methods are based on concepts from machine learning, pattern recognition and statistics: classification, clustering, graphical models, and so forth. The array of different algorithms for solving each of these problems can often be quite bewildering to both the experienced data analyst and the novice. In this section, we offer a brief overview of data mining methods and in particular try to convey the notion that most (if not all) methods can be viewed as extensions or hybrids of a few basic techniques and principles.

The section begins by discussing the primary tasks of data mining and then shows that the data mining methods to address these tasks consist of three primary algorithmic components: *model representation, model evaluation,* and *search.* The section concludes by discussing particular data mining algorithms within this framework.

1.5.1. The Primary Tasks of Data Mining

The two "high-level" primary goals of data mining in practice tend to be *prediction* and *description.* Prediction involves using some variables or fields in the database to predict unknown or future values of other variables of interest. Description focuses on finding human-interpretable patterns describing the data. The relative importance of prediction and description for particular data mining applications can vary considerably. However, in the context of KDD, description tends to be more important than prediction. This is in contrast to pattern recognition and machine learning applications (such as speech recognition) where prediction is often the primary goal.

The goals of prediction and description are achieved by using the following primary data mining tasks.

- *Classification* is learning a function that maps (classifies) a data item into one of several predefined classes. Examples of classification methods used as part of knowledge discovery applications include classifying trends in financial markets and automated identification of objects of interest in large image databases.

- *Regression* is learning a function, which maps a data item to a real-valued prediction variable. Regression applications are many, e.g., predicting the amount of biomass present in a forest given remotely-sensed microwave measurements, estimating the probability that a patient will die given the results of a set of diagnostic tests, predicting consumer demand for a new product as a function of advertising expenditure, and time series prediction where the input variables can be time-lagged versions of the prediction variable.
- *Clustering* is a common descriptive task where one seeks to identify a finite set of categories or clusters to describe the data. Given a set of attritional descriptions of some entities, a description language for characterizing classes of such entities, and a classification quality criterion, the problem is to partition entities into classes in a way that maximizes the classification quality criterion, and simultaneously to determine general (extensional) descriptions of the classes in the given description language. The categories may be mutually exclusive and exhaustive, or consist of a richer representation such as hierarchical or overlapping categories. Examples of clustering applications in a knowledge discovery context include discovering homogeneous sub-populations for consumers in marketing databases and identification of sub-categories of spectra from infrared sky measurements. Closely related to clustering is the task of *probability density estimation*, which consists of techniques for estimating from data the joint multivariate probability density function of all of the variables/fields in the database.
- *Summarization* involves methods for finding a compact description for a subset of data. A simple example would be tabulating the mean and standard deviations for all fields. More sophisticated methods involve the derivation of summary rules, multivariate visualization techniques, and the discovery of functional relationships between variables. Summarization techniques are often applied to interactive exploratory data analysis and automated report generation.
- *Dependency Modeling* consists of finding a model, which describes significant *dependencies* between variables. Dependency models exist at two levels: the *structural* level of the model specifies (often in graphical form) which variables are locally dependent on each other, whereas the *quantitative* level of the model specifies the strengths of the dependencies using some numerical scale. For example, probabilistic dependency networks use conditional independence to specify the structural aspect of the model and probabilities or correlations to specify the strengths of the dependencies. Probabilistic dependency networks are increasingly finding applications in areas as diverse as the development of probabilistic medical expert systems from databases, information retrieval, and modeling of the human genome.
- *Change and Deviation Detection* focuses on discovering the most significant changes in the data from previously measured or normative values.

1.5.2. The Components of Data Mining Algorithms

Having outlined the primary tasks of data mining, the next step is to construct algorithms to solve them. One can identify three primary components in any data mining algorithm: *model representation, model evaluation,* and *search.* This reductionist view is not necessarily complete or fully encompassing: rather, it is a convenient way to express the key concepts of data mining algorithms in a relatively unified and compact manner.

- *Model Representation* is the language *L* for describing discoverable patterns. If the representation is too limited, then no amount of training time or examples will produce an accurate model for the data. For example, a decision tree representation, using univariate (single-field) node-splits, partitions the input space into hyper-planes, which are parallel to the attribute axes. Such a decision-tree method cannot discover from data the formula $x=y$ no matter how much training data it is given. Thus, it is important that a data analyst fully comprehend the representational assumptions, which may be inherent to a particular method. It is equally important that an algorithm designer clearly state which representational assumptions are being made by a particular algorithm. Note that more powerful representational power for models increases the danger of overfitting the training data resulting in reduced prediction accuracy on unseen data. In addition the search becomes much more complex and interpretation of the model is typically more difficult.
- *Model Evaluation* estimates how well a particular pattern (a model and its parameters) meet the criteria of the KDD process. Evaluation of predictive accuracy (validity) is based on cross validation. Evaluation of descriptive quality involves predictive accuracy, novelty, utility, and understandability of the fitted model. Both logical and statistical criteria can be used for model evaluation. For example, the maximum likelihood principle chooses the parameters for the model, which yield the best fit to the training data.
- *Search Method* consists of two components: *Parameter Search* and *Model Search.* In *parameter search* the algorithm must search for the parameters which optimize the model evaluation criteria given observed data and a fixed model representation. For relatively simple problems there is no search: the optimal parameter estimates can be obtained in closed form. Typically, for more general models, a closed form solution is not available: greedy iterative methods are commonly used, e.g., the gradient descent method of backpropagation for neural networks. *Model Search* occurs as a loop over the parameter search method: the model representation is changed so that a family of models are considered. For each specific model representation, the parameter search method is instantiated to evaluate the quality of that particular model. Implementations of model search methods tend to use heuristic search techniques since the size of the space of possible models often prohibits exhaustive search and closed form solutions are not easily obtainable.

1.6. A Discussion of Popular Data Mining Methods

There exist a wide variety of data mining methods: here we only focus on a subset of popular techniques. Each method is discussed in the context of model representation, model evaluation, and search.

1.6.1. Decision Trees and Rules

A decision tree can be transformed into a set of decision rules (a ruleset) by traversing all paths from the root to individual leaves. Such rules can often be simplified by detecting superfluous conditions in them. Decision trees and rules that use univariate splits have a simple representational form, making the inferred model relatively easy to comprehend by the user. However, the restriction to a particular tree or rule representation can significantly restrict the functional form (and thus the approximation power) of the model. If one enlarges the model space to allow more general expressions (such as multivariate hyper-planes at arbitrary angles), then the model is more powerful for prediction but may be much more difficult to comprehend. There are a large number of decision trees and rule induction algorithms described in the machine learning and applied statistics literature.

To a large extent they are based on likelihood-based model evaluation methods with varying degrees of sophistication in terms of penalizing model complexity. Greedy search methods, which involve growing and pruning rule and tree structures, are typically employed to explore the super-exponential space of possible models. Trees and rules are primarily used for predictive modeling, both for classification and regression, although they can also be applied to summary descriptive modeling.

1.6.2. Nonlinear Regression and Classification Methods

These methods consist of a family of techniques for prediction, which fit linear and non-linear combinations of basis functions (sigmoids[2], splines, and polynomials) to combinations of the input variables. Examples include feedforward neural networks, adaptive spline methods, projection pursuit regression, and so forth. In terms of model evaluation, while networks of the appropriate size can universally approximate any smooth function to any desired degree of accuracy, relatively little is known about the representation properties of fixed size networks estimated from *finite* data sets. In terms of model evaluation, the standard squared error and cross entropy loss functions used to train neural networks can be viewed as log-likelihood functions for regression and classification respectively. Backpropagation is a parameter search method, which performs gradient descent in parameter (weight) space to find a local maximum of the likelihood function starting from random initial conditions. Nonlinear regression methods, though powerful in representational power, can be very difficult to interpret.

1.6.3. Example-based Methods

The representation is simple: use representative examples from the database to approximate a model, i.e., predictions on new examples are derived from the properties of "similar" examples in the model whose prediction is known. Techniques include nearest-neighbor classification and regression algorithms and case-based reasoning systems.

A Potential disadvantage of example-based methods (compared with tree-based methods for example) is that a well-defined distance metric for evaluating the distance between data points is required. Model evaluation is usually based on cross-validation

[2] Having the shape of the letter S.

estimates of a prediction error: "parameters" of the model to be estimated can include the number of neighbors to use for prediction and the distance metric itself. Like non-linear regression methods, example-based methods are often asymptotically quite powerful in terms of approximation properties, but conversely can be difficult to interpret since the model is implicit in the data and not explicitly formulated. Related techniques include kernel density estimation and mixture modeling.

1.6.4. Probabilistic Graphical Dependency Models

Graphical models specify the probabilistic dependencies which underlie a particular model using a graph structure. In its simplest form, the model specifies which variables are directly dependent on each other. Typically, these models are used with categorical or discrete-valued variables, but extensions to special cases, such as Gaussian densities, for real-valued variables are also possible. Within the artificial intelligence and statistical communities these models were initially developed within the framework of probabilistic expert systems: the structure of the model and the parameters (the conditional probabilities attached to the links of the graph) were elicited from experts. More recently, there has been significant work in both the AI and statistical communities on methods whereby both the structure and parameters of graphical models can be learned from databases directly. Model evaluation criteria are typically Bayesian in form and parameter estimation that can be a mixture of closed form estimates and iterative methods depending on whether a variable is directly observed or hidden. Model search can consist of greedy hill-climbing methods over various graph structures. Prior knowledge, such as a partial ordering of the variables based on causal relations, can be quite useful in terms of reducing the model search space. Although still primarily at the research phase, graphical model induction methods are of particular interest to KDD since the graphical form of the model lends itself easily to human interpretation.

1.6.5. Relational Learning Models

While decision-trees and rules have a representation restricted to propositional logic, relational learning (also known as inductive logic programming) uses the more flexible pattern language of first-order logic. A relational learner can easily find formulas such as $X=Y$. Most research so far on model evaluation methods for relational learning are logical in nature. The extra representational power of relational models comes at the price of significant computational demands in terms of search.

Given the broad spectrum of data mining methods and algorithms, our brief overview is inevitably limited in scope: there are many data mining techniques, particularly specialized methods for particular types of data and domains, which were not mentioned specifically in the discussion. We believe the general discussion on data mining tasks and components has general relevance to a variety of methods. For example, consider time series prediction: traditionally this has been cast as a predictive regression task (autoregressive models and so forth). Recently, more general models have been developed for time series applications such as non-linear basis function, example-based, and kernel methods. Furthermore, there has been significant interest in *descriptive* graphical and local data modeling of time series rather than purely *predictive* modeling. Thus, although different algorithms and applications may appear quite different on the surface, it is not uncommon to find that they share many common components. Understanding data mining and model induction at this component level clarifies the task of any data mining algorithm and makes it easier for the user to understand its overall contribution and applicability to the KDD process.

We would like to remind the reader that our discussion and overview of data mining methods has been both cursory and brief. There are two important points we would like to make clear:

1. *Automated Search:* Our brief overview has focused mainly on automated methods for extracting patterns and/or models from data. While this is consistent with the definition we gave earlier, it does not necessarily represent what other communities might refer to as *data mining.* For example, some use the term to designate any manual search of the data, or search assisted by queries to a DBMS or humans visualizing patterns in data as data mining. In other communities, it is used to refer to the automated correlation of data from transactions or the automated generation of transaction reports. We choose to focus only on methods that contain certain degrees of search autonomy.
2. *Beware the Hype:* The state-of-the-art in automated methods in data mining is still in a fairly early stage of development. There are no established criteria for deciding which methods to use in which circumstances, and many of the approaches are based on crude heuristic approximations to avoid the expensive search required to find optimal or even good solutions. Hence, the reader should be careful when confronted with overstated claims about the great ability of a system to mine useful information from large (or even small) databases.

1.7. Application Issues

In the business world, the most successful and widespread application of KDD is "Database Marketing," which is a method of analyzing customer databases, looking for patterns among existing customer preferences and using those patterns for more targeted selection of future customers. Business Week, a popular business magazine in the United States, carried a cover story on Database Marketing that estimated that over 50% of all retailers are using or planning to use database marketing. The reason is simple - significant results can be obtained using this approach: e.g., a 15-20-percentage increase in credit-card purchases reported by American Express.

Another major business use of data mining methods is the analysis and selection of stocks and other financial instruments. There are already numerous investment companies, which pick stocks using a variety of advanced data mining methods.

Several successful applications have been developed for analysis and reporting on change in data. These include Coverstory from IRI, Spotlight from A. C. Nielsen for supermarket sales data, and KEFIR from GTE, for health care databases.

Fraud detection and prevention is another area where KDD plays a role. While there have been many applications, published information is, for obvious reasons, not readily available. Here we mention just a few noteworthy examples. A system for detecting healthcare provider fraud in electronically submitted claims, has been developed at Travelers Insurance. The Internal Revenue Service has developed a pilot system for selecting tax returns for audits. Neural network based tools, such as Nestor FDS have been developed for detecting credit-card fraud and are reportedly watching millions of accounts.

A number of interesting and important scientific applications of KDD have also been developed. Example application areas in science include

- *Astronomy:* The SKICAT system from JPL/Caltech is used by astronomers to automatically identify stars and galaxies in a large-scale sky survey for cataloging and scientific analysis.
- *Molecular Biology:* Systems have been developed for finding patterns in molecular structures and in genetic data.
- *Global Climate Change Modeling:* Spatio-temporal patterns such as cyclones are automatically found from large simulated and observational datasets.

1.7.1. Guidelines for Selecting a Potential KDD Application

The criteria for selecting applications can be divided into practical and technical. The practical criteria for KDD projects are similar to those for other application of advanced technology, while the technical ones are more specific to KDD.

Practical criteria include consideration of the **potential for significant impact of an application**. For business applications, this could be measured by criteria such as greater revenue, lower costs, higher quality, or savings in time. For scientific applications, the impact can be measured by the novelty and quality of the discovered knowledge and by increased access to data via automating manual analysis processes. Another important practical consideration is that **no good alternatives exist**: the solution is not easily obtainable by other standard means. Hence, the ultimate user has a strong vested interest in insuring the success of the KDD venture**. Organizational support** is another consideration: there should be a champion for using new technology; e.g. a domain expert who can define a proper interestingness measure for that domain as well as participate in the KDD process. Finally, an important practical consideration is the **potential for privacy/legal issues**. This applies primarily to databases on people where one needs to guard against the discovered patterns raising legal or ethical issues of invasion of privacy.

Technical criteria include considerations such as the **availability of sufficient data (cases)**. The number of examples (cases) required for reliable inference of useful patterns from data varies a great deal with each particular application. In general, the more fields there are and the more complex are the patterns being sought, the more data are needed. However, strong prior knowledge can reduce the number of needed cases significantly. Another consideration is the **relevance of attributes**. It is important to have data attributes relevant to the discovery task: no amount of data will allow prediction based on attributes that do not capture the required information.

Furthermore, **low noise levels (few data errors)** is another consideration. High amounts of noise make it hard to identify patterns unless a large number of cases can mitigate random noise and help clarify the aggregate patterns. A related consideration is whether one can attach **confidence intervals** to extracted knowledge. In some applications, it is crucial to attach confidence intervals to predictions produced by the KDD system. This allows the user to calibrate actions appropriately.

Finally, and perhaps one of the most important considerations is **prior knowledge**. It is very useful to know something about the domain - what are the important fields, what are the likely relationships, what is the user utility function, what patterns are already known, and so forth. Prior knowledge can significantly reduce the search in the data mining step and all the other steps in the KDD process.

1.7.2. Privacy and Knowledge Discovery

When dealing with databases of personal information, governments and businesses have to be careful to adequately address the legal and ethical issues of invasion of privacy. Ignoring this issue can be dangerous, as Lotus found in 1990, when they were planning to introduce a CDROM with data on about 100 million American households. The stormy protest led to the withdrawal of that product.

Current discussion centers around guidelines for what constitutes a proper discovery. The Organization for Economic Cooperation and Development (OECD) guidelines for data privacy, which have been adopted by most European Union countries, suggest that data about specific living individuals should not be analyzed without their consent. They also suggest that the data should only be collected for a specific purpose. Use for other purposes is possible only with the consent of the data subject or by authority of the law.

In the U.S. there is ongoing work on draft principles for fair information use related to the National Information Infrastructure (NII), Commonly known as the “information superhighway.” These principles permit the use of “transactional records,” such as telephone numbers called, credit card payments, etc., as long as such use is compatible with the original notice. The use of transactional records can be seen to also include discovery of patterns.

In many cases (e.g. medical research, socioeconomic studies) the goal is to discover patterns about groups, not individuals. While group pattern discovery appears not to validate the restrictions on personal data retrieval, an ingenious combination of several group patterns, especially in small databases, may allow identification of specific personal information. Solutions which allow group pattern discovery while avoiding the potential invasion of privacy include removal or replacement of identifying fields, performing queries on random subsets of data, and combining individuals into groups and allowing only queries on groups.

1.7.3. Research and Application Challenges for KDD

We outline some of the current primary research and application challenges for knowledge discovery. This list is by no means exhaustive. The goal is to give the reader a feel for the types of problems that KDD practitioners wrestle with.

- *Larger databases.* Databases with hundreds of fields and tables, millions of records, and multi-gigabyte size are quite commonplace, and terabyte (10^{12} bytes) databases are beginning to appear. For example, next in this paper we present efficient algorithms for enumerating all association rules exceeding given confidence thresholds over large databases. Other possible solutions include sampling, approximation methods, and massively parallel processing.
- *High dimensionality.* Not only is there often a very large number of records in the database, but there can also be a very large number of fields (attributes, variables) so that the dimensionality of the problem is high. A high dimensional data set creates problems in terms of increasing the size of the search space for model induction in a combinatorially explosive manner. In addition, it increases the chances that a data mining algorithm will find spurious patterns that are not valid in general. Approaches to this problem include methods to reduce the effective dimensionality of the problem and the use of prior knowledge to identify irrelevant variables.
- *Overfitting.* When the algorithm searches for the best parameters for one particular model using a limited set of data, it may over fit the data, resulting in poor performance of the model on test data. Possible solutions include cross-validation, regularization, and other sophisticated statistical strategies.
- *Assessing statistical significance.* A problem (related to overfitting) occurs when the system is searching over many possible models. For example, if a system tests N models at the 0.001 significance level, then on average, with purely random data, $N/1000$ of these models will be accepted as significant. This point is frequently missed by many initial attempts at KDD. One way to deal with this problem is to use methods which adjust the test statistic as a function of the search.
- *Changing data and knowledge.* Rapidly changing (non-stationary) data may make previously discovered patterns invalid. In addition, the variables measured in a given application database may be modified, deleted, or augmented with new measurements over time. Possible solutions include incremental methods for updating the patterns and treating change as an opportunity for discovery by using it to cue the search for patterns of change only.
- *Missing and noisy data.* This problem is especially acute in business databases. U.S. census data reportedly has error rates of up to 20%. Important attributes may be missing if the database was not designed with discovery in mind. Possible solutions include more sophisticated statistical strategies to identify hidden variables and dependencies.
- *Complex relationships between fields.* Hierarchically structured attributes or values, relations between attributes, and more sophisticated means for representing knowledge about the contents of a database will require algorithms that can effectively utilize such information. Historically, data mining algorithms have been developed for simple attribute-value records, although new techniques for deriving relations between variables are being developed.

- *Understandability of patterns.* In many applications, it is important to make the discoveries more understandable by humans. Possible solutions include graphical representations, rule structuring with directed acyclic graphs, natural language generation and techniques for visualization of data and knowledge. Rule refinement strategies can be used to address a related problem: the discovered knowledge may be implicitly or explicitly redundant.
- *User interaction and prior knowledge.* Many current KDD methods and tools are not truly *interactive* and cannot easily incorporate prior knowledge about a problem except in simple ways. The use of domain knowledge is important in all of the steps of the KDD process. Bayesian approaches use prior probabilities over data and distributions as one form of encoding prior knowledge.
- *Integration with other systems.* A stand-alone discovery system may not be very useful. Typical integration issues include integration with a DBMS (e.g. via a query interface), integration with spreadsheets and visualization tools, and accommodating real-time sensor readings.

2. Association Rules

2.1. Introduction to Association Rules

Data mining, or the efficient discovery of interesting patterns from large collection of data, has been recognized as an important area of database research. The most commonly sought patterns are *association rules* as introduced in [3]. Intuitively, an association rule identifies a frequently occurring pattern of information in a database. For example, consider a department store database where the set of items purchased by a single customer is recorded as a transaction. The department store owner may be interested in finding an *association* among the items purchased together. An example of such association is that if a customer buys bread and butter then it is likely he will buy milk. Given a set of transactions, where each transaction is a set of items, an association rule is an expression $X \Rightarrow Y$, where X and Y are sets of items. The intuitive meaning of such a rule is that transactions of database, which contain items in X, also tend to contain items in Y.

Agrawal ([3]) gives an example for using association rules. Consider a supermarket with a large collection of items (this supermarket example will follow us throughout the entire paper). Typical business decisions that the management has to make include such things as what to put on sale, how to place merchandise on the shelves in order to maximize the profit, and so on. Analysis of past transactions data, is commonly used in order to help making such decisions. Using association rules mining techniques we can look for association in a transactions database, that can help us make management decisions. Examples for such association rules are:

1. Find all rules that have "Diet Coke" as consequent. These rules may help plan what the store should do to boost the sale of Diet Coke.
2. Find all rules that have "bagels" in the antecedent. These rules may help determine what products may be impacted if the store discontinues selling bagels.
3. Find all rules that have "sausage" in the antecedent and "mustard" in the consequent. This query can be phrased alternatively as a request for the additional items that have to be sold together with sausage in order to make it highly likely that mustard will also be soled.
4. Find all rules relating items located on shelves A and B in the store. These rules may help shelf planning by determining if the sale of items on shelf A is related to the sale of items on shelf B.
5. Find the "best" *k* rules that have "bagels" in the consequent. Here, "best" can be formulated in terms of the confidence factor of the rules, or in the terms of their support, i.e., the factor of transactions satisfying the rule.

2.2. Formal Model

The following is a formal statement of the problem (as introduced in [3], but in a more general manner, in that it allows a consequent to have more than one item). Let $I=\{i_1,i_2,...,i_m\}$ be a set of attributes over the binary domain {0,1}, called items. Let T be a database of transactions, in which each transaction t is represented as a binary vector, with $t[k]=1$ if t bought the item I_k, and $t[k]=0$ otherwise. There is one tuple in the database for each transaction. Let X be a set of some items in I. We say that a transaction *t satisfies X* if for all items I_k in X, $t[k]=1$.

By an *association rules*, we mean an implication of the form $X \Rightarrow Y$, where X and Y are sets of some items in I ($X \subseteq I$, $Y \subseteq I$), and any item I_j which is in X, is not present in Y, and vice versa ($X \cap Y = \emptyset$). Negative or missing items, are not considered of interest in this approach[3]. The rule $X \Rightarrow Y$ holds in the transaction set T with *confidence factor* $0 \leq minconfidence \leq 1$ if c% of the transactions in T that contain X also contain Y.

Given a set of transactions T, the problem of mining association rules is to generate *all* association rules that have a certain user-specified constraints of two different forms:

1. *Syntactic Constraints*: These constraints involve restrictions on items that can appear in a rule. For example, we may be interested only in rules that have a specific item, I_x appearing in the consequent (or in the antecedent). A combination of some constraints is also possible.
2. *Support Constraints*: These constraints concern the number of transactions in T that support a rule. The support is defined as the percentage of the transactions that have both items of the antecedent and consequent in them. Support shouldn't be confused with confidence. While confidence defines the rules strength, support is a statistical measurement.

Considering we intend to use the *support constraints*, the problem of mining the association rules, can be divided into two sub-problems.

1. Generating all combinations of items that have a certain satisfying support. A satisfying support is a support, which is above a certain threshold called *minsupport*. Items that satisfy the support constraint are called *large* itemsets, and all the others are called *small* itemsets. We can use both syntactic and support constrains, for example, we might be interested in association rules that have I_x in the antecedent, and also have a certain support.
2. For a given large itemset $L=i_1,i_2...i_k$, $k>1$, generate all rules that use items from the set L. The antecedent of each rule will be a subset X of L, and the consequent will be a subset Y of L, such that $X,Y \neq \emptyset$ and $Y=L-X$, therefore $X \cap Y = \emptyset$ and $X \cup Y = L$. To generate a rule $X \Rightarrow Y$, with a confidence c, take the support of L and divide it by the support of X. If the ratio generated, is greater then or equal to *minconfidence* then the rule is satisfied with the confidence factor c, otherwise it is not.

2.3. Related work

Related but not directly applicable work includes the induction of classification rules, discovery of causal rules, learning of logical definitions, fitting of functions to data and clustering.

[3] The problem of finding association rules when having negative items as well, will be approached later in this paper.

The closest work in the machine learning literature is the KID3 algorithm. If used for finding all association rules, this algorithm will make as many passes over the data as the number of combinations of items in the antecedent, which is exponentially large.

Related work in the database literature is the work on inferring functional dependencies for data.

There has been work on quantifying the "usefulness" or "interestingness" of a rule. What is useful or interesting is often application-dependent. The need for a human in the loop and providing tools to allow human guidance of the rule discovery process has been articulated.

2.4. Discovering Large Itemsets

Algorithms for discovering large itemsets make multiple passes over the data. In the first pass, we count the support of each individual item and determine which items are large (which items have a support equal to or greater than minsupport). Then, in each subsequent pass, we start with a seed set of itemsets that found to be large in the previous pass. We use this seed set for generating new potentially large itemsets, called *candidate* itemsets, and count the actual support for these candidate itemsets during the pass over the data. At the end of the pass, we determine which of the candidate itemsets are actually large, and they become the seed for the next pass. This process continues until no new large itemsets are found.

2.4.1. AIS Algorithm

Figure 2 gives the AIS algorithm for finding large itemsets, as presented in [3].

```
procedure LargeItemsets
begin
   let Large set $L=\emptyset$
   let Frontier set $F=\{\emptyset\}$

   while $F\neq\emptyset$ do begin
      // make a pass over the database
      let Candidate set $C=\emptyset$
      forall database tuples $t$ do
         forall itemsets $f$ in $F$ do
            let $C_f$=candidate itemsets that are extensions of $f$ and contained in $t$
            forall itemsets $c_f$ in $C_f$ do
               if $c_f\in C$ then
                  $c_f$.count=$c_f$.count+1
               else
                  $c_f$.count=0
                  $C=C+c_f$
               end
            end
```

```
        // consolidate
        let $F=\varnothing$
        forall itemsets $c$ in $C$ do begin
           if count(c)/$dbsize>minsupport$ then
              $L=L+c$
           If $c$ should be used as frontier in the next pass then
              $F=F+c$
        end
     end
  end
```

Figure 2 – AIS Algorithm

Given a set of items I, and an itemset $X+Y$ of items in I is said to be an extension of the itemset X if $X \cap Y=\varnothing$. The parameter *dbsize* is the total number of tuples in the database.

The algorithm makes multiple passes over the database. The *frontier* set for a pass consists of those itemsets that are extended during the pass. In each pass, the support of certain itemsets is measured. These itemsets, called *candidate itemsets*, are derived from the tuples in the database and the itemsets contained in the frontier set.

A counter is associated to each itemset. The counter holds the number of transactions in which the corresponding itemset has appeared. This counter is set to zero when an itemset is created.

Initially the frontier set consists of only one element, which is an empty set. At the end of a pass, the support for a candidate is compared with *minsupport* to determine if it is a *large* itemset. At the same time, it is determined if the itemset should be added to the frontier set for the next pass. The algorithm terminates when the frontier set becomes empty. The support count for the itemset is preserved when an itemset is added to the large/frontier set.

In the most straightforward version of the algorithm, every itemset present in any of the tuples will be measured in one pass, terminating the algorithm in one pass. In the worst case, this approach will require setting up 2^m counters corresponding to all subsets of the set of items I, where m is the number of items in I. This is, of course, not only infeasible (m can easily be more than 1000 in a supermarket setting) but also unnecessary. Indeed, most likely there will be very few large itemsets containing more than l items, where l is small. Hence, a lot of those 2^m combinations will turn out to be small.

A better approach is to measure in the k^{th} pass only those itemsets that contain exactly k items. Having measured some items in the k^{th} pass, we need to measure in the $(k+1)^{\text{th}}$ pass only those itemsets that are 1-extensions (an itemset extended by exactly one item) of large itemsets. If an itemset is small, its 1-extension is also going to be small. Thus, the frontier set for the next pass is set to candidate itemsets determined large in the current pass, and only 1-extensions of a frontier itemset are generated and measured during a pass. This alternative represents another extreme – we will make to many passes over the database.

2.4.2. Apriori Algorithm

Figure 3 gives the Apriori algorithm (presented in [5]).

```
$L_1$={large 1-itemsets}
for (k=2;$L_{k-1} \neq \emptyset$;k++) do begin
   $C_k$=apriori-gen($L_{k-1}$) // New candidate
   forall transactions $t \in D$ do begin
      $C_t$=subset($C_k$,t) // candidate contained it t
      forall candidates $c \in C_t$ do
         $c$.count++
   end
   $L_k=\{c \in C_k \mid c.\text{count} \geq \text{minsupport}\}$
end
Answer=$\cup_k L_k$
```

Figure 3 – Algorithm Apriori

k-itemset	An itemset having *k* items
L_k	Set of large *k*-itemsets (those with minimum support). Each member of this set has two fields: 1. itemset and 2. support count.
C_k	Set of candidate *k*-itemsets (potentially large itemsets). Each member of this set has two fields: 1. itemset and 2. support count.

Table 1 - Notation

The first pass of the algorithm simply counts items occurrences to determine the large 1-itemsets. A subsequent pass, for example *k*, consists of two stages. First, the large itemsets L_{k-1} found in the (*k*-1)th pass, are used to generate the candidate itemsets C_k, using the apriori-gen function. Second, the database is scanned and the support of each candidate in C_k is counted. Finally, candidates that their support is below the minsupport are removed.

The apriori-gen function takes as an argument L_{k-1}, the set of all large (*k*-1)-itemsets and it returns a superset of the set of all large *k*-itemsets. First, in the *join* step, we join L_{k-1} with L_{k-1}, in order to obtain a superset if the final set of candidates C_k. The union $p \cup q$, where $p,q \in L_{k-1}$, is inserted in C_k if p and q share their first *k*-2 items. Next, in the *prune* step, we delete all itemsets $c \in C_k$ such that some (*k*-1)-subset of c is not in L_{k-1}.

As an example, let L_3 be {{1,2,3}, {1,2,4}, {1,3,4}, {1,3,5}, {2,3,4}}. After the join step, C_4 will be {{1,2,3,4}, {1,3,4,5}}. The prune step will delete the itemset

{1,3,4,5} because itemset {1,4,5} is not in L_3. We will be then left with C_4={{1,2,3,4}}.

2.5. Algorithm for Mining Association Rules

The following is an algorithm for mining association rules in a largest itemset.

1. Set n to one (n=1), create an empty illegal list (denoted by IL)
2. Create a list of sets that contain all possible permutations of items such that:
   1. The size of the set is n (the set has n elements),
   2. The set may not contain any combination from the illegal list.
3. If the list is empty then stop.
4. For each set, compute its support.
5. It the support is below the threshold T_I, put the content of the set on the illegal list.
6. If not then set n to n+1 (n=n+1), and go back to 2.
7. Create threshold α to represent the lowest confidence of an association rule.
8. Create all the association rules such that their confidence is above the threshold α.

As stated before, the problems of mining association rules can be divided into two sub-problems. The first is to find all combinations of items that have a support above the *minsupport* (in our algorithm this is done in 1 to 6). It this stage we can use the AIS algorithm or the Apriori Algorithm, which we described before. The second is generating the association rules (7 and 8).

The illegal list (IL) is a list that contains all the permutations that their support is below the threshold T_i.

The objective of the algorithm is to find the largest possible association set. In order to find the largest possible association set we need to remove those sets that their support (frequency) is *small.* We define *small* as a support, which is below the threshold. The threshold (T_n) is found in the following way.

Let *minsupport* be the smallest frequency, *maxsupport* be the largest frequency and T_n be:

1. $$n = 1 \Rightarrow T_1 = \begin{cases} \max(2, MinSupport + 1) & if \quad MaxSupport > MinSupport \\ \max(2, MinSupport) & if \quad MaxSupport = MinSupport \end{cases}$$

2. $$n > 1 \Rightarrow T_n = \begin{cases} \min(\max(2, MinSupport + 1), T_{n-1}) & if \quad MaxSupport > MinSupport \\ \max(2, MinSupport) & if \quad MaxSupport = MinSupport \end{cases}$$

Then T_n represents a frequency threshold for those items that appear at least T times in the database. Equations 1 and 2 represent a dynamic change in the threshold. Basically, it is required that the threshold will not increase as *n* increases. It is logical to assume that as *n* grows, the support of the permutation decreases. To avoid the situation when the increase in *n* will yield an increase in T, Equations 1 and 2 developed.

The stopping condition is quite simple. If we increase *n* by one, and no permutations are found, we stop the process.

2.6. Example 1

In this example will use the Native algorithm, which the simplest way of generating itemsets.

Lets take a look at the first 6 stages of the algorithm presented above.

1. Set n to one (n=1), create an empty illegal list (denoted by IL)
2. Create a list of sets that contain all possible permutations of items such that:
    1. The size of the set is n (the set has n elements),
    2. The set may not contain any combination from the illegal list.
3. If the list is empty then stop.
4. For each set, compute its support.
5. It the support is below the threshold T_I, put the content of the set on the illegal list.

The Native algorithm is a simple interpretation of line number 2. In each stage we'll generate all possible itemsets, and then compare them with the illegal list. We're not looking for a faster, more sophisticated way for generating itemsets.

Assume we have a department store where *n* customers (denoted by C_i) can buy *m* items (denoted by I_j) in the store. The following table summarizes the transactions:

	I_1	I_2	I_3	I_4	I_5	I_6
C_1	1			1		1
C_2		1			1	
C_3			1	1		
C_4	1		1			1
C_5	1	1		1	1	
C_6		1				
C_7	1			1	1	
C_8		1				
C_9	1			1		
C_{10}	1		1	1		
C_{11}	1	1		1	1	1

Table 2 - Summary of transactions

In the first stage, we have to create a list of itemsets, where each one of the sets has to contain only one element. The creation of the list should be done in the following way, for each one of the customers we'll create a set containing all the products the customer bought (we'll refer to that set as products-set), for example customer number 1 (denoted by C_1)[4] bought products 1, 4 and 6 (denoted by I_1, I_4 and I_6), therefore his products-set is $\{I_1,I_4,I_6\}$. In the same way, customer number 2's products-set is $\{I_2,I_5\}$. After we've created the products-set for all the customers, we'll take each one of the products-set and extract from it all its subsets, which contain only one element, those subsets are called "Itemset". For customer number 1, which has a products-set of $\{I_1,I_4,I_6\}$ we'll get $\{I_1\}$, $\{I_4\}$ and $\{I_6\}$ as its itemsets. For customer number, 2 we'll get $\{I_2\}$ and $\{I_5\}$ as its itemsets. As we can see some of the customers bought the same products, for example customers 1, 4, 5, 7, 9, 10 and 11 all bought product number 1, therefore all those customers have product number 1 in their products-set, hence they all have the $\{I_1\}$ set as their itemset. The following table summarizes all the itemsets extracted from all the products-sets and their frequency (which means how many customers have that sub-set in their products-set).

Itemset	Frequency
$\{I_1\}$	7
$\{I_2\}$	5
$\{I_3\}$	3
$\{I_4\}$	7
$\{I_5\}$	4
$\{I_6\}$	3

Table 3 - Summary of all itemsets of one element and their frequencies

As we can see, creating all the itemsets of one element results with a list of all products. Therefore the first stage of mining association rules can be defined as creating a list of all the products (same as creating all the itemsets of one element) and counting how many people bought each one of them (their frequency).

Assuming that minsupport is the lowest frequency and maxsupport is the highest frequency, in our example minsupport is 3 and maxsupport is 7. maxsupport and maxsupport are used while calculating the threshold. When n=1 (first stage) we're using the following formula for finding the threshold:

$$T_1 = \begin{cases} \max(2, MinSupport+1) & if \quad MaxSupport > MinSupport \\ \max(2, MinSuppor) & if \quad MaxSupport = MinSupport \end{cases}$$

Since in our example maxsupport is higher than minsupport, we'll use *max*(*2,MinSupport+1*), which in our example is *max*(*2,4*). As a result we get 4, therefore, itemsets $\{I_3\}$ and $\{I_6\}$ are put into the illegal list (IL=$\{\{I_3\}, \{I_6\}\}$).

Now we go to the second stage, in which we create all itemsets that have two elements. The creation of the itemsets with two elements is done in a similar way to the creation of the itemsets with one element. From each one of the products-set found in the first section, we'll extract all the subsets, which has two elements. For example customer number 1 has $\{I_1,I_4\}$, $\{I_1,I_6\}$ and $\{I_4,I_6\}$ as his itemsets. The following table

[4] For the ease of use customers and products will be referred to by their numbers, for example customer C_1 will be referred to as customer number 1.

summarizes all the itemsets which have two element, extracted from all the products-sets and their frequency.

Itemset	Frequency
$\{I_1,I_2\}$	2
$\{I_1,I_3\}$	2
$\{I_1,I_4\}$	6
$\{I_1,I_5\}$	3
$\{I_2,I_4\}$	2
$\{I_1,I_6\}$	3
$\{I_2,I_5\}$	3
$\{I_3,I_4\}$	2
$\{I_3,I_6\}$	1
$\{I_4,I_5\}$	3
$\{I_4,I_6\}$	2
$\{I_2,I_6\}$	1
$\{I_5,I_6\}$	1

Table 4 - Summary of all itemsets of two elements and their frequencies

As we can see from the table above some of the itemsets have sets from the illegal list as their subsets. For example, the itemset $\{I_1,I_3\}$ has $\{I_3\}$ as it subset, and $\{I_4,I_6\}$ has $\{I_6\}$ as it subset. Any itemset that has a set from the illegal list as its subset should be removed from the list. The following table is a summary of all the itemsets of two elements that don't have any subsets that are in the illegal list.

Itemset	Frequency
$\{I_1,I_2\}$	2
$\{I_1,I_4\}$	6
$\{I_1,I_5\}$	3
$\{I_2,I_4\}$	2
$\{I_2,I_5\}$	3
$\{I_4,I_5\}$	3

Table 5 - Summary of all itemsets of two elements and their frequencies, not including itemsets that have subsets that are in the illegal list

While in the process of mining the association rules, we will need to find for each one of the stages its own threshold, therefore we will need to find minsupport and maxsupport each time. The new minsupport and maxsupport for this stage are 2 and 6. The threshold for stages two and above is calculated using the following formula:

$$T_n = \begin{cases} \min(\max(2, MinSupport + 1), T_{n-1}) & if \quad MaxSupport > MinSupport \\ \max(2, MinSupport) & if \quad MaxSupport = MinSupport \end{cases}$$

In our example the threshold for the second stage is 3, therefore itemsets $\{I_1,I_2\}$ and $\{I_2,I_4\}$ are put into the illegal list, which is now IL=$\{\{I_3\}, \{I_6\}, \{I_1,I_2\}, \{I_2,I_4\}\}$.

On the third stage, we create all itemsets that have three elements. The creation of the itemsets with three elements is done in the same way we did it before, from each one of the products-set found in the first section, we'll extract all the subsets, which has three elements. The following table summarizes all the itemsets which have three element, extracted from all the products-sets and their frequency.

Itemset	Frequency
$\{I_1,I_2,I_4\}$	2
$\{I_1,I_2,I_5\}$	2
$\{I_1,I_2,I_6\}$	1
$\{I_1,I_3,I_4\}$	1
$\{I_1,I_3,I_6\}$	1
$\{I_1,I_4,I_5\}$	3
$\{I_1,I_4,I_6\}$	2
$\{I_1,I_5,I_6\}$	1
$\{I_2,I_4,I_5\}$	2
$\{I_2,I_4,I_6\}$	1
$\{I_4,I_5,I_6\}$	1

Table 6 – Summary of all itemsets of three elements and their frequencies

The illegal list contains the following itemsets, which were added to it in the first and second stages. $IL=\{\{I_3\}, \{I_6\}, \{I_1,I_2\}, \{I_2,I_4\}\}$. As we did in the second stage, each one of the itemsets found (and are listed in the table above), which contains any of the itemsets of the illegal list as its subset should be removed. We end up with the following list.

Itemset	Frequency
$\{I_1,I_4,I_5\}$	3

Table 7 – Summary of all itemsets of three elements and their frequencies, not including itemsets that have subsets that are in the illegal list

Since we found only one set, this is the last stage and the process is terminated. According to the algorithm, we should stop if we get an empty list of itemsets. Lets just continue one more stage in order to show that the next stage results in an empty list. In order to continue to the next stage, we should find the threshold and add the necessary itemsets to the illegal list. In this stage minsupport is equal to maxsupport and is 3. According to the formula for finding the threshold, if minsupport is equal to maxsupport then the threshold is the highest number of 2 and minsupport, which in our case is 3. In this case, none of the itemsets found are put into the illegal list. Now we create all itemsets of four elements, which don't have any subsets that are in the illegal list. As we said before there are no such itemsets. There are three itemsets which have four elements in our example, $\{I_1,I_2,I_4,I_5\}$ which has $\{I_2,I_4\}$ as its subsets, and $\{I_1,I_4,I_5,I_6\}$ and $\{I_2,I_4,I_5,I_6\}$, which both have $\{I_6\}$ as their subsets, both subsets are in the illegal list. Therefore stage four results in an empty list of itemsets.

Now we know which items will be in the association rules. If *l* is the itemset, and *a* is a subset of *l*, then a possible association rule is: $a \Rightarrow (l-a)$. In this way, we can generate the following association rules:

$I_1 \Rightarrow I_4I_5$, $I_4 \Rightarrow I_1I_5$, $I_5 \Rightarrow I_1I_4$, $I_1I_4 \Rightarrow I_5$, $I_1I_5 \Rightarrow I_4$ and $I_4I_5 \Rightarrow I_1$

Assuming that the confidence of each one of the possible association rules can be found by dividing the frequency of *l* by the frequency of *a*, we get the following results.

Association rule	Frequency (*l*)	Frequency (*a*)	Confidence
$I_1 \Rightarrow I_4I_5$	3	7	3/7 (0.48)
$I_4 \Rightarrow I_1I_5$	3	7	3/7 (0.48)
$I_5 \Rightarrow I_1I_4$	3	4	3/4 (0.75)
$I_1I_4 \Rightarrow I_5$	3	6	3/6 (0.5)
$I_1I_5 \Rightarrow I_4$	3	3	3/3 (1)
$I_4I_5 \Rightarrow I_1$	3	3	3/3 (1)

Table 8 – Final set of association rules

Now, after mining all possible association rules, and their confidences, we have to create a threshold α that represents the lowest confidences of the association rules we want. If we pick, for example, $\alpha=0.75$ (which means that there is at least 75% chance that the association rules really exist), then our final association rules will be $I_1I_5 \Rightarrow I_4$ and $I_4I_5 \Rightarrow I_1$, which have a confidence of 1 (all the association rules that have confidence higher than 0.75).

3. Inverse Associations

The previous example might raise the following question. Why do we have blank entries in the table ? In other words, why the customer did not purchase all the possible items ? One answer might be that the customer does not need the particular items in a given time, which is fine. However, another answer might be that some items cannot be purchased together, or in other words, if one buys item *i* he/she will not purchase item *j*. The algorithm for computing these negative (or inverse) relations is the subject of this chapter.

3.1. Formal Model

The following is a formal statement of the problem. Let $I=\{i_1,i_2,...,i_m\}$ be a set of attributes over the domain {-1,1}, called items. Let *T* be a database of transactions, in which each transaction *t* is represented as a vector, with $t[k]=1$ if *t* bought the item I_k, and $t[k]=(-1)$ otherwise. There is one tuple in the database for each transaction. Let *X* be a set of all items in *I*. We say that a transaction *t satisfies X* if for all items I_k in *X*, $t[k]=1$ or $t[k]=(-1)$.

By an *inverse* association *rules*, we mean an implication of the form $X\Rightarrow Y$, where *X* and *Y* are sets of some items in *I* ($X\subseteq I$, $Y\subseteq I$), and any item I_j which is in *X*, is not present in *Y*, and vice versa ($X\cap Y=\varnothing$). In this approach, missing items are considered as items that were not bought by the customer, whether they were really not bought or not. The rule $X\Rightarrow Y$ holds in the transaction set *T* with *confidence factor* $0\leq minconfidence\leq 1$ if c% of the transactions in *T* that contain *X* also contain *Y*.

Given a set of transactions *T*, the problem of mining inverse association rules is to generate *all* inverse association rules that have a certain user-specified constraints of two different forms:

1. *Syntactic Constraints*: These constraints involve restrictions on items that can appear in a rule. For example, we may be interested only in rules that have a specific item I_x appearing in the consequent (or in the antecedent). A combination of some constraints is also possible. A very important syntactic constraint is that an association rule cannot be of the form $I_x\Rightarrow -I_x$, which some mining algorithms (such as the Apriori algorithm) might produce during the mining procedure.
2. *Support Constraints*: These constraints concern the number of transactions in *T* that support a rule. The support is defined as the percentage of the transactions that have both items of the antecedent and consequent in them. Support shouldn't be confused with confidence. While confidence defines the rules strength, support is a statistical measurement.

Considering we intend to use the *support constraints*, the problem of mining the association rules, can be divided into two sub-problems.

1. Generating all combinations of items that have a certain satisfying support. A satisfying support is a support, which is above a certain threshold called *minsupport*. Items that satisfy the support constraint are called *large* itemsets, and

all the others are called *small* itemsets. We can use both syntactic and support constrains, for example, we might be interested in association rules that have I_x in the antecedent, and also have a certain support.

2. For a given large itemset $L=i_1,i_2...i_k$, k>1, generate all rules that use items from the set L. The antecedent of each rule will be a subset X of L, and the consequent will be a subset Y of L, such that $X,Y\neq\emptyset$ and $Y=L-X$, therefore $X\cap Y=\emptyset$ and $X\cup Y=L$. To generate a rule $X\Rightarrow Y$, with a confidence c, take the support of L and divide it by the support of X. If the ratio generated, is greater than *minconfidence* then the rule is satisfied with the confidence factor c, otherwise it is not.

3.2. Algorithm for Mining Inverse Association Rules

When mining association rules we took under consideration only attributes with values of one. Attributes with value of zero were not considered while mining the association rules. When we're mining inverse association rules, our database contain transactions that their attributes are either one or minus one. Now we should use all attributes, whether they contain one or minus one.

We can use the following general algorithm, presented earlier, for mining inverse association rules.

1. Set n to one (n=1), create an empty illegal list (denoted by IL)
2. Create a list of sets that contain all possible permutations of items such that:
   1. The size of the set is n (the set has n elements),
   2. The set may not contain any combination from the illegal list.
3. If the list is empty then stop.
4. For each set, compute its support.
5. It the support is below the threshold T_1, put the content of the set on the illegal list.
6. If not then set n to n+1 (n=n+1), and go back to 2.
7. Create threshold α to represent the lowest confidence of an association rule.
8. Create all the association rules such that their confidence is above the threshold α.

The following example illustrates the mining of the inverse association rules from the same data as in the previous example.

3.3. Example 2

We start by stating that if there is an entry somewhere in the database, we mark it as I_j and if there is no entry, we mark it as $-I_j$.

	I_1	I_2	I_3	I_4	I_5	I_6
C_1	1	-1	-1	1	-1	1
C_2	-1	1	-1	-1	1	-1
C_3	-1	-1	1	1	-1	-1
C_4	1	-1	1	-1	-1	1
C_5	1	1	-1	1	1	-1
C_6	-1	1	-1	-1	-1	-1
C_7	1	-1	-1	1	1	-1

	I_1	I_2	I_3	I_4	I_5	I_6
C_8	-1	1	-1	-1	-1	-1
C_9	1	-1	-1	1	-1	-1
C_{10}	1	-1	1	1	-1	-1
C_{11}	1	1	-1	1	1	1

Table 9 – Summary of transactions

In the first stage, we have to create a list of itemsets, where each one of the itemsets has to contain only one element, as we did when we found the regular association rules. The creation of the list should be done in the following way, for each one of the customers we'll create a set containing all the products the customer bought, and we add to that set all the products that he didn't buy, adding a minus sign ("-") before them (we will refer to that set as products-set). For example customer number 1 bought products 1, 4 and 6, and he didn't buy products 2,3, and 5, therefore his products-set is $\{I_1,I_4,I_6,-I_2,-I_3,-I_5\}$. We should notice, that when constructing the products-sets, all products-sets of all the customers must contain the same number of elements (which is the number of items in the store), unlike the products-sets found for the regular association rules. After we've created the products-set for all the customers, we'll take each one of the products-set and extract from it all its subsets, which contain only one element. For customer number 1, which has a products-set of $\{I_1,I_4,I_6,-I_2,-I_3,-I_5\}$ we'll get $\{I_1\}$, $\{I_4\}$, $\{I_6\}$, $\{-I_2\}$, $\{-I_3\}$ and $\{-I_6\}$ as its itemsets. The following table summarizes all the itemsets extracted from all the products-sets and their frequencies.

Itemset	Frequency
$\{-I_1\}$	4
$\{-I_2\}$	6
$\{-I_3\}$	8
$\{-I_4\}$	4
$\{-I_5\}$	7
$\{-I_6\}$	8
$\{I_1\}$	7
$\{I_2\}$	5
$\{I_3\}$	3
$\{I_4\}$	7
$\{I_5\}$	4
$\{I_6\}$	3

Table 10 – Summary of all itemsets of one element and their frequencies

As we did when we found the regular association rules, a threshold has to be found, and itemsets with frequency below the threshold, have to be put in the illegal list. Again, if MinSupport is the lowest frequency and MaxSupport is the highest frequency, then, in our example, MinSupport is 3 and MaxSupport is 8. We are using the same formula we used in the first stage of mining the regular association rules, to find our threshold.

$$T_1 = \begin{cases} \max(2, MinSupport+1) & if \quad MaxSupport > MinSupport \\ \max(2, MinSuppor) & if \quad MaxSupport = MinSupport \end{cases}$$

Since maxsupport is higher than minsupport, we'll use *max(2,MinSupport+1)*, which in our example is *max(2,4)*. As a result we get 4, therefore, itemsets {I_3} and {I_6} are put into the illegal list (IL={{I_3}, {I_6}}).

In the second stage, we create all itemsets that have two elements. The creation of the itemsets with two elements is done in a similar way to the creation of the itemsets with one element. From each one of the products-set found in the first section, we'll extract all the subsets, which has two elements. The following table summarizes all the itemsets which have two elements, extracted from all the products-sets and their frequencies.

Itemset	Frequency	Itemset	Frequency	Itemset	Frequency
{-I_1,-I_2}	1	{-I_1,-I_3}	3	{-I_1,-I_4}	3
{-I_1,-I_5}	3	{-I_1,-I_6}	4	{-I_2,-I_3}	3
{-I_2,-I_4}	1	{-I_2,-I_5}	5	{-I_2,-I_6}	4
{-I_3,-I_4}	3	{-I_3,-I_5}	4	{-I_3,-I_6}	6
{-I_4,-I_5}	3	{-I_4,-I_6}	3	{-I_5,-I_6}	5
{I_1,-I_2}	5	{I_1,-I_3}	5	{I_1,-I_4}	1
{I_1,-I_5}	4	{I_1,-I_6}	4	{I_1,I_2}	2
{I_1,I_3}	2	{I_1,I_4}	6	{I_1,I_5}	3
{I_1,I_6}	3	{I_2,-I_1}	3	{I_2,-I_3}	5
{I_2,-I_4}	3	{I_2,-I_5}	2	{I_2,-I_6}	4
{I_2,I_4}	2	{I_2,I_5}	3	{I_2,I_6}	1
{I_3,-I_1}	1	{I_3,-I_2}	3	{I_3,-I_4}	1
{I_3,-I_5}	3	{I_3,-I_6}	2	{I_3,I_4}	2
{I_3,I_6}	1	{I_4,-I_1}	1	{I_4,-I_2}	5
{I_4,-I_3}	5	{I_4,-I_5}	4	{I_4,-I_6}	5
{I_4,I_5}	3	{I_4,I_6}	2	{I_5,-I_1}	1
{I_5,-I_2}	1	{I_5,-I_3}	4	{I_5,-I_4}	1
{I_5,-I_6}	3	{I_5,I_6}	1	{I_6,-I_2}	2
{I_6,-I_3}	2	{I_6,-I_4}	1	{I_6,-I_5}	2

Table 11 - Summary of all itemsets of two elements and their frequencies

As with the regular association rules, any itemset found, which has an item from the illegal list as his subset should be removed, and not considered when mining the inverse association rules. The flowing tables list all itemsets that don't have any items from the illegal list as their subsets. These itemsets will be later used for mining the inverse association rules.

Itemset	Frequency	Itemset	Frequency	Itemset	Frequency
{-I_1,-I_2}	1	{-I_1,-I_3}	3	{-I_1,-I_4}	3
{-I_1,-I_5}	3	{-I_1,-I_6}	4	{-I_2,-I_3}	3
{-I_2,-I_4}	1	{-I_2,-I_5}	5	{-I_2,-I_6}	4
{-I_3,-I_4}	3	{-I_3,-I_5}	4	{-I_3,-I_6}	6

Itemset	Frequency	Itemset	Frequency	Itemset	Frequency
$\{-I_4,-I_5\}$	3	$\{-I_4,-I_6\}$	3	$\{-I_5,-I_6\}$	5
$\{I_1,-I_2\}$	5	$\{I_1,-I_3\}$	5	$\{I_1,-I_4\}$	1
$\{I_1,-I_5\}$	4	$\{I_1,-I_6\}$	4	$\{I_1,I_2\}$	2
$\{I_1,I_4\}$	6	$\{I_1,I_5\}$	3	$\{I_2,-I_1\}$	3
$\{I_2,-I_3\}$	5	$\{I_2,-I_4\}$	3	$\{I_2,-I_5\}$	2
$\{I_2,-I_6\}$	4	$\{I_2,I_4\}$	2	$\{I_2,I_5\}$	3
$\{I_4,-I_1\}$	1	$\{I_4,-I_2\}$	5	$\{I_4,-I_3\}$	5
$\{I_4,-I_5\}$	4	$\{I_4,-I_6\}$	5	$\{I_4,I_5\}$	3
$\{I_5,-I_1\}$	1	$\{I_5,-I_2\}$	1	$\{I_5,-I_3\}$	4
$\{I_5,-I_4\}$	1	$\{I_5,-I_6\}$	3		

Table 12 - Summary of all itemsets of two elements and their frequencies, not including itemsets that have subsets that are in the illegal list

Now, we have to decide which new itemsets should be put into the illegal list. For that, we should use the second formula (which from now, will be used throughout the entire process).

$$T_n = \begin{cases} \min\left(\max\left(2, MinSupport + 1\right), T_{n-1}\right) & \text{if} \quad MaxSupport > MinSupport \\ \max\left(2, MinSupport\right) & \text{if} \quad MaxSupport = MinSupport \end{cases}$$

minsupport is 1, and maxsupport is 6, therefore the threshold is 2, and $\{-I_1,-I_2\}$, $\{-I_2,-I_4\}$, $\{I_4,-I_1\}$, $\{I_5,-I_1\}$, $\{I_5,-I_4\}$, $\{I_5,-I_2\}$ and $\{I_1,-I_4\}$ are put in the illegal list. The illegal list is now: IL=$\{\{I_3\}, \{I_6\}, \{-I_1,-I_2\}, \{-I_2,-I_4\}, \{I_4,-I_1\}, \{I_5,-I_1\}, \{I_5,-I_4\}, \{I_5,-I_2\}, \{I_1,-I_4\}\}$

The third stage involves the generations of all itemsets with three elements. The following table lists all these itemsets.

Itemset	Frequency	Itemset	Frequency	Itemset	Frequency
$\{-I_1,-I_3,-I_4\}$	3	$\{-I_1,-I_3,-I_5\}$	2	$\{-I_1,-I_3,-I_6\}$	3
$\{-I_1,-I_4,-I_5\}$	2	$\{-I_1,-I_4,-I_6\}$	3	$\{-I_1,-I_5,-I_6\}$	3
$\{-I_1,I_2,-I_3\}$	3	$\{-I_1,I_2,-I_4\}$	3	$\{-I_1,I_2,-I_5\}$	2
$\{-I_1,I_2,-I_6\}$	3	$\{-I_2,-I_3,-I_5\}$	2	$\{-I_2,-I_3,-I_6\}$	2
$\{-I_2,-I_3,I_4\}$	3	$\{-I_2,-I_5,-I_6\}$	3	$\{-I_2,I_4,-I_5\}$	4
$\{-I_2,I_4,-I_6\}$	4	$\{-I_3,-I_4,-I_5\}$	2	$\{-I_3,-I_4,-I_6\}$	3
$\{-I_3,-I_5,-I_6\}$	3	$\{-I_3,I_4,-I_5\}$	2	$\{-I_3,I_4,-I_6\}$	3
$\{-I_3,I_4,I_5\}$	3	$\{-I_3,I_5,-I_6\}$	3	$\{-I_4,-I_5,-I_6\}$	2
$\{I_1,-I_2,-I_3\}$	3	$\{I_1,-I_2,-I_5\}$	4	$\{I_1,-I_2,-I_6\}$	3
$\{I_1,-I_2,I_4\}$	4	$\{I_1,-I_3,-I_5\}$	2	$\{I_1,-I_3,-I_6\}$	3
$\{I_1,-I_3,I_4\}$	5	$\{I_1,-I_3,I_5\}$	3	$\{I_1,-I_5,-I_6\}$	2
$\{I_1,I_2,-I_3\}$	2	$\{I_1,I_2,-I_6\}$	1	$\{I_1,I_2,I_4\}$	2
$\{I_1,I_2,I_5\}$	2	$\{I_1,I_4,-I_5\}$	3	$\{I_1,I_4,-I_6\}$	4
$\{I_1,I_4,I_5\}$	3	$\{I_1,I_5,-I_6\}$	2	$\{I_2,-I_3,-I_4\}$	3
$\{I_2,-I_3,-I_5\}$	2	$\{I_2,-I_3,-I_6\}$	4	$\{I_2,-I_3,I_4\}$	2
$\{I_2,-I_3,I_5\}$	3	$\{I_2,-I_4,-I_5\}$	2	$\{I_2,-I_4,-I_6\}$	3

Itemset	Frequency	Itemset	Frequency	Itemset	Frequency
$\{I_2,-I_5,-I_6\}$	2	$\{I_2,I_4,-I_6\}$	1	$\{I_2,I_4,I_5\}$	2
$\{I_2,I_5,-I_6\}$	2	$\{I_4,-I_5,-I_6\}$	3	$\{I_4,I_5,-I_6\}$	2
$\{-I_1,-I_2,-I_5\}$	1	$\{-I_1,-I_2,-I_6\}$	1	$\{-I_1,-I_2,I_3\}$	1
$\{-I_1,-I_2,I_4\}$	1	$\{-I_1,-I_3,I_5\}$	1	$\{-I_1,-I_4,I_5\}$	1
$\{-I_1,I_2,I_5\}$	1	$\{-I_1,I_3,-I_5\}$	1	$\{-I_1,I_3,-I_6\}$	1
$\{-I_1,I_3,I_4\}$	1	$\{-I_1,I_4,-I_5\}$	1	$\{-I_1,I_4,-I_6\}$	1
$\{-I_1,I_5,-I_6\}$	1	$\{-I_2,-I_3,I_5\}$	1	$\{-I_2,-I_3,I_6\}$	1
$\{-I_2,-I_4,-I_5\}$	1	$\{-I_2,-I_4,I_6\}$	1	$\{-I_2,-I_5,I_6\}$	2
$\{-I_2,I_3,-I_4\}$	1	$\{-I_2,I_3,-I_5\}$	3	$\{-I_2,I_3,-I_6\}$	2
$\{-I_2,I_3,I_4\}$	2	$\{-I_2,I_3,I_6\}$	1	$\{-I_2,I_4,I_5\}$	1
$\{-I_2,I_4,I_6\}$	1	$\{-I_2,I_5,-I_6\}$	1	$\{-I_3,-I_4,I_5\}$	1
$\{-I_3,-I_5,I_6\}$	1	$\{-I_3,I_4,I_6\}$	2	$\{-I_3,I_5,I_6\}$	1
$\{-I_4,-I_5,I_6\}$	1	$\{-I_4,I_5,-I_6\}$	1	$\{I_1,-I_2,-I_4\}$	1
$\{I_1,-I_2,I_3\}$	2	$\{I_1,-I_2,I_5\}$	1	$\{I_1,-I_2,I_6\}$	2
$\{I_1,-I_3,I_6\}$	2	$\{I_1,-I_4,-I_5\}$	1	$\{I_1,-I_4,I_6\}$	1
$\{I_1,-I_5,I_6\}$	2	$\{I_1,I_2,I_6\}$	1	$\{I_1,I_3,-I_4\}$	1
$\{I_1,I_3,-I_5\}$	2	$\{I_1,I_3,-I_6\}$	1	$\{I_1,I_3,I_4\}$	1
$\{I_1,I_3,I_6\}$	1	$\{I_1,I_4,I_6\}$	2	$\{I_1,I_5,I_6\}$	1
$\{I_2,-I_3,I_6\}$	1	$\{I_2,-I_4,I_5\}$	1	$\{I_2,I_4,I_6\}$	1
$\{I_2,I_5,I_6\}$	1	$\{I_3,-I_4,-I_5\}$	1	$\{I_3,-I_4,I_6\}$	1
$\{I_3,-I_5,-I_6\}$	2	$\{I_3,-I_5,I_6\}$	1	$\{I_3,I_4,-I_5\}$	2
$\{I_3,I_4,-I_6\}$	2	$\{I_4,-I_5,I_6\}$	1	$\{I_4,I_5,I_6\}$	1

Table 13 - Summary of all itemsets of three elements and their frequencies

Since the illegal list is now IL=$\{\{I_3\}, \{I_6\}, \{-I_1,-I_2\}, \{-I_2,-I_4\}, \{I_4,-I_1\}, \{I_5,-I_1\}, \{I_5,-I_4\}, \{I_5,-I_2\}, \{I_1,-I_4\}\}$, and we are not interested in itemsets that have any itemsets from the illegal list as their subsets, the list of itemsets with three elements will be reduced to the following:

Itemset	Frequency	Itemset	Frequency	Itemset	Frequency
$\{-I_1,-I_3,-I_4\}$	3	$\{-I_1,-I_3,-I_5\}$	2	$\{-I_1,-I_3,-I_6\}$	3
$\{-I_1,-I_4,-I_5\}$	2	$\{-I_1,-I_4,-I_6\}$	3	$\{-I_1,-I_5,-I_6\}$	3
$\{-I_1,I_2,-I_3\}$	3	$\{-I_1,I_2,-I_4\}$	3	$\{-I_1,I_2,-I_5\}$	2
$\{-I_1,I_2,-I_6\}$	3	$\{-I_2,-I_3,-I_5\}$	2	$\{-I_2,-I_3,-I_6\}$	2
$\{-I_2,-I_3,I_4\}$	3	$\{-I_2,-I_5,-I_6\}$	3	$\{-I_2,I_4,-I_5\}$	4
$\{-I_2,I_4,-I_6\}$	4	$\{-I_3,-I_4,-I_5\}$	2	$\{-I_3,-I_4,-I_6\}$	3
$\{-I_3,-I_5,-I_6\}$	3	$\{-I_3,I_4,-I_5\}$	2	$\{-I_3,I_4,-I_6\}$	3
$\{-I_3,I_4,I_5\}$	3	$\{-I_3,I_5,-I_6\}$	3	$\{-I_4,-I_5,-I_6\}$	2
$\{I_1,-I_2,-I_3\}$	3	$\{I_1,-I_2,-I_5\}$	4	$\{I_1,-I_2,-I_6\}$	3
$\{I_1,-I_2,I_4\}$	4	$\{I_1,-I_3,-I_5\}$	2	$\{I_1,-I_3,-I_6\}$	3
$\{I_1,-I_3,I_4\}$	5	$\{I_1,-I_3,I_5\}$	3	$\{I_1,-I_5,-I_6\}$	2
$\{I_1,I_2,-I_3\}$	2	$\{I_1,I_2,-I_6\}$	1	$\{I_1,I_2,I_4\}$	2
$\{I_1,I_2,I_5\}$	2	$\{I_1,I_4,-I_5\}$	3	$\{I_1,I_4,-I_6\}$	4
$\{I_1,I_4,I_5\}$	3	$\{I_1,I_5,-I_6\}$	2	$\{I_2,-I_3,-I_4\}$	3
$\{I_2,-I_3,-I_5\}$	2	$\{I_2,-I_3,-I_6\}$	4	$\{I_2,-I_3,I_4\}$	2

Itemset	Frequency	Itemset	Frequency	Itemset	Frequency
$\{I_2,-I_3,I_5\}$	3	$\{I_2,-I_4,-I_5\}$	2	$\{I_2,-I_4,-I_6\}$	3
$\{I_2,-I_5,-I_6\}$	2	$\{I_2,I_4,-I_6\}$	1	$\{I_2,I_4,I_5\}$	2
$\{I_2,I_5,-I_6\}$	2	$\{I_4,-I_5,-I_6\}$	3	$\{I_4,I_5,-I_6\}$	2

Table 14 - Summary of all itemsets of three elements and their frequencies, not including itemsets that have subsets that are in the illegal list

Now we calculate the threshold, in order to decide which itemsets we should put into the illegal list. minsupport is 1 and maxsupport is 5, calculating the threshold results with a threshold of 2. Therefore $\{I_1,I_2,-I_6\}$ and $\{I_2,I_4,-I_6\}$ are put into the illegal list, which is now: $IL=\{\{I_3\}, \{I_6\}, \{-I_1,-I_2\}, \{-I_2,-I_4\}, \{I_4,-I_1\}, \{I_5,-I_1\}, \{I_5,-I_4\}, \{I_5,-I_2\}, \{I_1,-I_4\}, \{I_1,I_2,-I_6\}, \{I_2,I_4,-I_6\}\}$

We continue to the fourth stage, and create all itemsets with four elements, as presented in the following table.

Itemset	Frequency	Itemset	Frequency	Itemset	Frequency
$\{-I_1,-I_3,-I_4,-I_5\}$	2	$\{-I_1,-I_3,-I_4,-I_6\}$	3	$\{-I_1,-I_3,-I_5,-I_6\}$	2
$\{-I_1,-I_4,-I_5,-I_6\}$	2	$\{-I_1,I_2,-I_3,-I_4\}$	3	$\{-I_1,I_2,-I_3,-I_5\}$	2
$\{-I_1,I_2,-I_3,-I_6\}$	3	$\{-I_1,I_2,-I_4,-I_5\}$	2	$\{-I_1,I_2,-I_4,-I_6\}$	3
$\{-I_1,I_2,-I_5,-I_6\}$	2	$\{-I_2,-I_3,-I_5,-I_6\}$	1	$\{-I_2,-I_3,I_4,-I_5\}$	2
$\{-I_2,-I_3,I_4,-I_6\}$	2	$\{-I_2,I_4,-I_5,-I_6\}$	3	$\{-I_3,-I_4,-I_5,-I_6\}$	2
$\{-I_3,I_4,-I_5,-I_6\}$	1	$\{-I_3,I_4,I_5,-I_6\}$	2	$\{I_1,-I_2,-I_3,-I_5\}$	2
$\{I_1,-I_2,-I_3,-I_6\}$	2	$\{I_1,-I_2,-I_3,I_4\}$	3	$\{I_1,-I_2,-I_5,-I_6\}$	2
$\{I_1,-I_2,I_4,-I_5\}$	3	$\{I_1,-I_2,I_4,-I_6\}$	3	$\{I_1,-I_3,-I_5,-I_6\}$	1
$\{I_1,-I_3,I_4,-I_5\}$	2	$\{I_1,-I_3,I_4,-I_6\}$	3	$\{I_1,-I_3,I_4,I_5\}$	3
$\{I_1,-I_3,I_5,-I_6\}$	2	$\{I_1,I_2,-I_3,I_4\}$	2	$\{I_1,I_2,-I_3,I_5\}$	2
$\{I_1,I_2,I_4,I_5\}$	2	$\{I_1,I_4,-I_5,-I_6\}$	2	$\{I_1,I_4,I_5,-I_6\}$	2
$\{I_2,-I_3,-I_4,-I_5\}$	2	$\{I_2,-I_3,-I_4,-I_6\}$	3	$\{I_2,-I_3,-I_5,-I_6\}$	2
$\{I_2,-I_3,I_4,I_5\}$	2	$\{I_2,-I_3,I_5,-I_6\}$	2	$\{I_2,-I_4,-I_5,-I_6\}$	2
$\{-I_1,-I_2,-I_5,-I_6\}$	1	$\{-I_1,-I_2,I_3,-I_5\}$	1	$\{-I_1,-I_2,I_3,-I_6\}$	1
$\{-I_1,-I_2,I_3,I_4\}$	1	$\{-I_1,-I_2,I_4,-I_5\}$	1	$\{-I_1,-I_2,I_4,-I_6\}$	1
$\{-I_1,-I_3,-I_4,I_5\}$	1	$\{-I_1,-I_3,I_5,-I_6\}$	1	$\{-I_1,-I_4,I_5,-I_6\}$	1
$\{-I_1,I_2,-I_3,I_5\}$	1	$\{-I_1,I_2,-I_4,I_5\}$	1	$\{-I_1,I_2,I_5,-I_6\}$	1
$\{-I_1,I_3,-I_5,-I_6\}$	1	$\{-I_1,I_3,I_4,-I_5\}$	1	$\{-I_1,I_3,I_4,-I_6\}$	1
$\{-I_1,I_4,-I_5,-I_6\}$	1	$\{-I_2,-I_3,-I_5,I_6\}$	1	$\{-I_2,-I_3,I_4,I_5\}$	1
$\{-I_2,-I_3,I_4,I_6\}$	1	$\{-I_2,-I_3,I_5,-I_6\}$	1	$\{-I_2,-I_4,-I_5,I_6\}$	1
$\{-I_2,I_3,-I_4,-I_5\}$	1	$\{-I_2,I_3,-I_4,I_6\}$	1	$\{-I_2,I_3,-I_5,-I_6\}$	2
$\{-I_2,I_3,-I_5,I_6\}$	1	$\{-I_2,I_3,I_4,-I_5\}$	2	$\{-I_2,I_3,I_4,-I_6\}$	2
$\{-I_2,I_4,-I_5,I_6\}$	1	$\{-I_2,I_4,I_5,-I_6\}$	1	$\{-I_3,-I_4,I_5,-I_6\}$	1
$\{-I_3,I_4,-I_5,I_6\}$	1	$\{-I_3,I_4,I_5,I_6\}$	1	$\{I_1,-I_2,-I_3,I_5\}$	1
$\{I_1,-I_2,-I_3,I_6\}$	1	$\{I_1,-I_2,-I_4,-I_5\}$	1	$\{I_1,-I_2,-I_4,I_6\}$	1
$\{I_1,-I_2,-I_5,I_5\}$	2	$\{I_1,-I_2,I_3,-I_4\}$	1	$\{I_1,-I_2,I_3,-I_5\}$	2
$\{I_1,-I_2,I_3,-I_5\}$	1	$\{I_1,-I_2,I_3,I_4\}$	1	$\{I_1,-I_2,I_3,I_5\}$	1
$\{I_1,-I_2,I_4,I_5\}$	1	$\{I_1,-I_2,I_4,I_5\}$	1	$\{I_1,-I_2,I_5,-I_5\}$	1
$\{I_1,-I_3,-I_5,I_5\}$	1	$\{I_1,-I_3,I_4,I_5\}$	2	$\{I_1,-I_3,I_5,I_5\}$	1

Itemset	Frequency	Itemset	Frequency	Itemset	Frequency
$\{I_1,-I_4,-I_5,I_5\}$	1	$\{I_1,I_2,-I_3,-I_5\}$	1	$\{I_1,I_2,-I_3,I_5\}$	1
$\{I_1,I_2,I_4,-I_5\}$	1	$\{I_1,I_2,I_4,I_5\}$	1	$\{I_1,I_2,I_5,-I_5\}$	1
$\{I_1,I_2,I_5,I_5\}$	1	$\{I_1,I_3,-I_4,-I_5\}$	1	$\{I_1,I_3,-I_4,I_5\}$	1
$\{I_1,I_3,-I_5,-I_5\}$	1	$\{I_1,I_3,-I_5,I_5\}$	1	$\{I_1,I_3,I_4,-I_5\}$	1
$\{I_1,I_3,I_4,-I_5\}$	1	$\{I_1,I_4,-I_5,I_5\}$	1	$\{I_1,I_4,I_5,I_5\}$	1
$\{I_2,-I_3,-I_4,I_5\}$	1	$\{I_2,-I_3,I_4,-I_5\}$	1	$\{I_2,-I_3,I_4,I_5\}$	1
$\{I_2,-I_3,I_5,I_5\}$	1	$\{I_2,-I_4,I_5,-I_5\}$	1	$\{I_2,I_4,I_5,-I_5\}$	1
$\{I_2,I_4,I_5,I_5\}$	1	$\{I_3,-I_4,-I_5,I_5\}$	1	$\{I_3,I_4,-I_5,-I_5\}$	2

Table 15 - Summary of all itemsets of four elements and their frequencies

Again, removing itemsets that have elements from the illegal list as their subsets, reduce the list of itemsets with four elements to the following.

Itemset	Frequency	Itemset	Frequency	Itemset	Frequency
$\{-I_1,-I_3,-I_4,-I_5\}$	2	$\{-I_1,-I_3,-I_4,-I_6\}$	3	$\{-I_1,-I_3,-I_5,-I_6\}$	2
$\{-I_1,-I_4,-I_5,-I_6\}$	2	$\{-I_1,I_2,-I_3,-I_4\}$	3	$\{-I_1,I_2,-I_3,-I_5\}$	2
$\{-I_1,I_2,-I_3,-I_6\}$	3	$\{-I_1,I_2,-I_4,-I_5\}$	2	$\{-I_1,I_2,-I_4,-I_6\}$	3
$\{-I_1,I_2,-I_5,-I_6\}$	2	$\{-I_2,-I_3,-I_5,-I_6\}$	1	$\{-I_2,-I_3,I_4,-I_5\}$	2
$\{-I_2,-I_3,I_4,-I_6\}$	2	$\{-I_2,I_4,-I_5,-I_6\}$	3	$\{-I_3,-I_4,-I_5,-I_6\}$	2
$\{-I_3,I_4,-I_5,-I_6\}$	1	$\{-I_3,I_4,I_5,-I_6\}$	2	$\{I_1,-I_2,-I_3,-I_5\}$	2
$\{I_1,-I_2,-I_3,-I_6\}$	2	$\{I_1,-I_2,-I_3,I_4\}$	3	$\{I_1,-I_2,-I_5,-I_6\}$	2
$\{I_1,-I_2,I_4,-I_5\}$	3	$\{I_1,-I_2,I_4,-I_6\}$	3	$\{I_1,-I_3,-I_5,-I_6\}$	1
$\{I_1,-I_3,I_4,-I_5\}$	2	$\{I_1,-I_3,I_4,-I_6\}$	3	$\{I_1,-I_3,I_4,I_5\}$	3
$\{I_1,-I_3,I_5,-I_6\}$	2	$\{I_1,I_2,-I_3,I_4\}$	2	$\{I_1,I_2,-I_3,I_5\}$	2
$\{I_1,I_2,I_4,I_5\}$	2	$\{I_1,I_4,-I_5,-I_6\}$	2	$\{I_1,I_4,I_5,-I_6\}$	2
$\{I_2,-I_3,-I_4,-I_5\}$	2	$\{I_2,-I_3,-I_4,-I_6\}$	3	$\{I_2,-I_3,-I_5,-I_6\}$	2
$\{I_2,-I_3,I_4,I_5\}$	2	$\{I_2,-I_3,I_5,-I_6\}$	2	$\{I_2,-I_4,-I_5,-I_6\}$	2

Table 16 - Summary of all itemsets of four elements and their frequencies, not including itemsets that have subsets that are in the illegal list

We now calculate the new threshold, to decide which of the new itemsets will be put in the illegal list. minsupport is 1 and maxsupport is 3, therefore the threshold is 2. Hence $\{-I_3,I_4,-I_5,-I_6\}$, $\{-I_2,-I_3,-I_5,-I_6\}$ and $\{I_1,-I_3,-I_5,-I_6\}$ are put into the illegal list, which is now: IL=$\{\{I_3\}$, $\{I_6\}$, $\{-I_1,-I_2\}$, {-I2,-I4}, {I4,-I1}, $\{I_5,-I_1\}$, $\{I_5,-I_4\}$, $\{I_5,-I_2\}$, $\{I_1,-I_4\}$, $\{I_1,I_2,-I_6\}$, $\{I_2,I_4,-I_6\}$, $\{-I_3,I_4,-I_5,-I_6\}$, $\{-I_2,-I_3,-I_5,-I_6\}$, $\{I_1,-I_3,-I_5,-I_6\}\}$

In the fifth stage, we extract all itemsets with five elements.

Itemset	Frequency	Itemset	Frequency
$\{-I_1,-I_2,I_3,-I_5,-I_6\}$	1	$\{-I_1,-I_2,I_3,I_4,-I_5\}$	1
$\{-I_1,-I_2,I_3,I_4,-I_6\}$	1	$\{-I,-I_2,I_4,-I_5,-I_6\}$	1
$\{-I_1,-I_3,-I_4,-I_5,-I_6\}$	2	$\{-I_1,-I_3,-I_4,I_5,-I_6\}$	1
$\{-I_1,I_2,-I_3,-I_4,-I_5\}$	2	$\{-I_1,I_2,-I_3,-I_4,-I_6\}$	3
$\{-I_1,I_2,-I_3,-I_4,I_5\}$	1	$\{-I_1,I_2,-I_3,-I_5,-I_6\}$	2
$\{-I_1,I_2,-I_3,I_5,-I_6\}$	1	$\{-I_1,I_2,-I_4,-I_5,-I_6\}$	2

Itemset	Frequency	Itemset	Frequency
$\{-I_1,I_2,-I_4,I_5,-I_6\}$	1	$\{-I_1,I_3,I_4,-I_5,-I_6\}$	1
$\{-I_2,-I_3,I_4,-I_5,-I_6\}$	1	$\{-I_2,-I_3,I_4,-I_5,I_6\}$	1
$\{-I_2,-I_3,I_4,I_5,-I_6\}$	1	$\{-I_2,I_3,-I_4,-I_5,I_6\}$	1
$\{-I_2,I_3,I_4,-I_5,-I_6\}$	2	$\{I_1,-I_2,-I_3,-I_5,-I_6\}$	1
$\{I_1,-I_2,-I_3,-I_5,I_6\}$	1	$\{I_1,-I_2,-I_3,I_4,-I_5\}$	2
$\{I_1,-I_2,-I_3,I_4,-I_6\}$	2	$\{I_1,-I_2,-I_3,I_4,I_5\}$	1
$\{I_1,-I_2,-I_3,I_4,I_6\}$	1	$\{I_1,-I_2,-I_3,I_5,-I_6\}$	1
$\{I_1,-I2,-I_4,-I_5,I_6\}$	1	$\{I_1,-I_2,I_3,-I_4,-I_5\}$	1
$\{I_1,-I2,I_3,-I_4,I_6\}$	1	$\{I_1,-I_2,I_3,-I_5,-I_6\}$	1
$\{I_1,-I2,I_3,-I_5,I_6\}$	1	$\{I_1,-I_2,I_3,I_4,-I_5\}$	1
$\{I_1,-I_2,I_3,I_4,-I_6\}$	1	$\{I_1,-I_2,I_4,-I_5,-I_6\}$	2
$\{I_1,-I_2,I_4,-I_5,I_6\}$	1	$\{I_1,-I_2,I_4,I_5,-I_6\}$	1
$\{I_1,-I_3,I_4,-I_5,-I_6\}$	1	$\{I_1,-I_3,I_4,-I_5,I_6\}$	1
$\{I_1,-I_3,I_4,I_5,-I_6\}$	2	$\{I_1,-I_3,I_4,I_5,I_6\}$	1
$\{I_1,I_2,-I_3,I_4,-I_6\}$	1	$\{I_1,I_2,-I_3,I_4,I_5\}$	2
$\{I_1,I_2,-I_3,I_4,I_6\}$	1	$\{I_1,I_2,-I_3,I_5,-I_6\}$	1
$\{I_1,I_2,-I_3,I_5,I_6\}$	1	$\{I_1,I_2,I_4,I_5,-I_6\}$	1
$\{I_1,I_2,I_4,I_5,I_6\}$	1	$\{I_1,I_3,-I_4,-I_5,I_6\}$	1
$\{I_1,I_3,I_4,-I_5,-I_6\}$	1	$\{I_2,-I_3,-I_4,-I_5,-I_6\}$	2
$\{I_2,-I_3,-I_4,I_5,-I_6\}$	1	$\{I_2,-I_3,I_4,I_5,-I_6\}$	1
$\{I_2,-I_3,I_4,I_5,I_6\}$	1		

Table 17 - Summary of all itemsets of five elements and their frequencies

As with the previous stages, we remove itemsets with elements of the illegal list as their subsets, resulting with the following list.

Itemset	Frequency	Itemset	Frequency
$\{-I_1,-I_3,-I_4,-I_5,-I_6\}$	2	$\{-I_1,I_2,-I_3,-I_4,-I_5\}$	2
$\{-I_1,I_2,-I_3,-I_4,-I_6\}$	3	$\{-I_1,I_2,-I_3,-I_5,-I_6\}$	2
$\{-I_1,I_2,-I_4,-I_5,-I_6\}$	2	$\{I_1,-I_2,-I_3,I_4,-I_5\}$	2
$\{I_1,-I_2,-I_3,I_4,-I_6\}$	2	$\{I_1,-I_2,I_4,-I_5,-I_6\}$	2
$\{I_1,-I_3,I_4,I_5,-I_6\}$	2	$\{I_1,I_2,-I_3,I_4,I_5\}$	2
$\{I_2,-I_3,-I_4,-I_5,-I_6\}$	2		

Table 18 - Summary of all itemsets of five elements and their frequencies, not including itemsets that have subsets that are in the illegal list

A new threshold has to found in order to decide which itemsets will be put into the illegal list. minsupport is 2 and maxsupport is 3, therefore the threshold is 2. As a result we don't put new itemsets into the illegal list (since we don't have any itemset with a frequency of 1), which remains: IL=$\{\{I_3\}, \{I_6\}, \{-I_1,-I_2\}, \{-I2,-I4\}, \{I4,-I1\}, \{I_5,-I_1\}, \{I_5,-I_4\}, \{I_5,-I_2\}, \{I_1,-I_4\}, \{I_1,I_2,-I_6\}, \{I_2,I_4,-I_6\}, \{-I_3,I_4,-I_5,-I_6\}, \{-I_2,-I_3,-I_5,-I_6\}, \{I_1,-I_3,-I_5,-I_6\}\}$

We continue to the sixth stage, and extract all itemsets with six elements.

Itemset	Frequency
{$-I_1,-I_2,I_3,I_4,-I_5,-I_6$}	1
{$I_1,I_2,-I_3,I_4,I_5,I_6$}	1
{$I_1,I_2,-I_3,I_4,I_5,-I_6$}	1
{$-I_1,I_2,-I_3,-I_4,I_5,-I_6$}	1
{$-I_1,I_2,-I_3,-I_4,-I_5,-I_6$}	2
{$I_1,-I_2,-I_3,I_4,-I_5,I_6$}	1
{$I_1,-I_2,-I_3,I_4,I_5,-I_6$}	1
{$I_1,-I_2,-I_3,I_4,-I_5,-I_6$}	1
{$I_1,-I_2,I_3,-I_4,-I_5,I_6$}	1
{$I_1,-I_2,I_3,I_4,-I_5,-I_6$}	1

Table 19 - Summary of all itemsets of six elements and their frequencies

We then remove all the itemsets that have as a subset an element from the illegal list, resulting with the following list.

Itemset	Frequency
{$-I_1,I_2,-I_3,-I_4,-I_5,-I_6$}	2

Table 20 - Summary of all itemsets of six elements and their frequencies, not including itemsets that have subsets that are in the illegal list

We now have to decide which itemsets should be put into the illegal list. minsupport is 2 and maxsupport is 2, therefore the threshold is 2, and no new itemsets are being put into the illegal list, which remains: IL={{I_3}, {I_6}, {$-I_1,-I_2$}, {$-I_2,-I_4$}, {I4,-I1}, {$I_5,-I_1$}, {$I_5,-I_4$}, {$I_5,-I_2$}, {$I_1,-I_4$}, {$I_1,I_2,-I_6$}, {$I_2,I_4,-I_6$}, {$-I_3,I_4,-I_5,-I_6$}, {$-I_2,-I_3,-I_5,-I_6$}, {$I_1,-I_3,-I_5,-I_6$}}.

While mining inverse association rules, we use all the possible products that in the store to create the product-sets, therefore, in our example the product-sets contain six elements. Since in this stage, we found itemsets with six elements, the next stage will result no itemsets, since we have no product-sets with seven elements. In that case, this will be the last stage, before the creation of the inverse association rules.

Now we know which items will be in the inverse association rules. Again, as with the regular association rules, if *l* is the itemset, and *a* is a subset of *l*, then a possible association rule is: *a*⇒(*l*-*a*). Assuming that the confidence of each one of the possible association rules can be found by dividing the frequency of *l* by the frequency of *a*, we get the following results.

Association rule	Frequency (*l*)	Frequency (*a*)	Confidence
$-I_1 \Rightarrow I_2,-I_3,-I_4,-I_5,-I_6$	2	4	2/4 (0.5)
$I_2 \Rightarrow -I_1,-I_3,-I_4,-I_5,-I_6$	2	6	2/6 (0.34)
$-I_3 \Rightarrow -I_1,I_2,-I_4,-I_5,-I_6$	2	8	2/8 (0.25)
$-I_4 \Rightarrow -I_1,I_2,-I_3,-I_5,-I_6$	2	4	2/4 (0.5)
$-I_5 \Rightarrow -I_1,I_2,-I_3,-I_4,-I_6$	2	7	2/7 (0.29)
$-I_6 \Rightarrow -I_1,I_2,-I_3,-I_4,-I_5$	2	8	2/8 (0.25)

Association rule	Frequency (*l*)	Frequency (*a*)	Confidence
$-I_1, I_2 \Rightarrow -I_3, -I_4, -I_5, -I_6$	2	3	2/3 (0.67)
$-I_1, -I_3 \Rightarrow I_2, -I_4, -I_5, -I_6$	2	3	2/3 (0.67)
$-I_1, -I_4 \Rightarrow I_2, -I_3, -I_5, -I_6$	2	3	2/3 (0.67)
$-I_1, -I_5 \Rightarrow I_2, -I_3, -I_4, -I_6$	2	3	2/3 (0.67)
$-I_1, -I_6 \Rightarrow I_2, -I_3, -I_4, -I_5$	2	4	2/4 (0.5)
$I_2, -I_3 \Rightarrow -I_1, -I_4, -I_5, -I_6$	2	5	2/5 (0.4)
$I_2, -I_4 \Rightarrow -I_1, -I_3, -I_5, -I_6$	2	3	2/3 (0.67)
$I_2, -I_5 \Rightarrow -I_1, -I_3, -I_4, -I_6$	2	2	2/2 (1)
$I_2, -I_6 \Rightarrow -I_1, -I_3, -I_4, -I_5$	2	4	2/4 (0.5)
$-I_3, -I_4 \Rightarrow -I_1, I_2, -I_5, -I_6$	2	3	2/3 (0.67)
$-I_3, -I_5 \Rightarrow -I_1, I_2, -I_4, -I_6$	2	4	2/4 (0.5)
$-I_3, -I_6 \Rightarrow -I_1, I_2, -I_4, -I_5$	2	6	2/6 (0.34)
$-I_4, -I_5 \Rightarrow -I_1, I_2, -I_3, -I_6$	2	3	2/3 (0.67)
$-I_4, -I_6 \Rightarrow -I_1, I_2, -I_3, -I_5$	2	3	2/3 (0.67)
$-I_5, -I_6 \Rightarrow -I_1, I_2, -I_3, -I_4$	2	5	2/5 (0.4)
$-I_1, I_2, -I_3 \Rightarrow -I_4, -I_5, -I_6$	2	3	2/3 (0.67)
$-I_1, I_2, -I_4 \Rightarrow -I_3, -I_5, -I_6$	2	3	2/3 (0.67)
$-I_1, I_2, -I_5 \Rightarrow -I_3, -I_4, -I_6$	2	2	2/2 (1)
$-I_1, I_2, -I_6 \Rightarrow -I_3, -I_4, -I_5$	2	3	2/3 (0.67)
$-I_1, -I_3, -I_4 \Rightarrow I_2, -I_5, -I_6$	2	3	2/3 (0.67)
$-I_1, -I_3, -I_5 \Rightarrow I_2, -I_4, -I_6$	2	2	2/2 (1)
$-I_1, -I_3, -I_6 \Rightarrow I_2, -I_4, -I_5$	2	3	2/3 (0.67)
$-I_1, -I_4, -I_5 \Rightarrow I_2, -I_3, -I_6$	2	2	2/2 (1)
$-I_1, -I_4, -I_6 \Rightarrow I_2, -I_3, -I_5$	2	3	2/3 (0.67)
$-I_1, -I_5, -I_6 \Rightarrow I_2, -I_3, -I_4$	2	3	2/3 (0.67)
$I_2, -I_3, -I_4 \Rightarrow -I_1, -I_5, -I_6$	2	3	2/3 (0.67)
$I_2, -I_3, -I_5 \Rightarrow -I_1, -I_4, -I_6$	2	2	2/2 (1)
$I_2, -I_3, -I_6 \Rightarrow -I_1, -I_4, -I_5$	2	4	2/4 (0.5)
$I_2, -I_4, -I_5 \Rightarrow -I_1, -I_3, -I_6$	2	2	2/2 (1)
$I_2, -I_4, -I_6 \Rightarrow -I_1, -I_3, -I_5$	2	3	2/3 (0.67)
$I_2, -I_5, -I_6 \Rightarrow -I_1, -I_3, -I_4$	2	2	2/2 (1)
$-I_3, -I_4, -I_5 \Rightarrow -I_1, I_2, -I_6$	2	2	2/2 (1)
$-I_3, -I_4, -I_6 \Rightarrow -I_1, I_2, -I_5$	2	3	2/3 (0.67)
$-I_3, -I_5, -I_6 \Rightarrow -I_1, I_2, -I_4$	2	3	2/3 (0.67)
$-I_4, -I_5, -I_6 \Rightarrow -I_1, I_2, -I_3$	2	2	2/2 (1)
$-I_1, I_2, -I_3, -I_4 \Rightarrow -I_5, -I_6$	2	3	2/3 (0.67)
$-I_1, I_2, -I_3, -I_5 \Rightarrow -I_4, -I_6$	2	2	2/2 (1)
$-I_1, I_2, -I_3, -I_6 \Rightarrow -I_4, -I_5$	2	3	2/3 (0.67)
$-I_1, I_2, -I_4, -I_5 \Rightarrow -I_3, -I_6$	2	2	2/2 (1)
$-I_1, I_2, -I_4, -I_6 \Rightarrow -I_3, -I_5$	2	3	2/3 (0.67)
$-I_1, I_2, -I_5, -I_6 \Rightarrow -I_3, -I_4$	2	2	2/2 (1)
$-I_1, -I_3, -I_4, -I_5 \Rightarrow I_2, -I_6$	2	2	2/2 (1)
$-I_1, -I_3, -I_4, -I_6 \Rightarrow I_2, -I_5$	2	3	2/3 (0.67)

Association rule	Frequency (*l*)	Frequency (*a*)	Confidence
$-I_1,-I_3,-I_5,-I_6 \Rightarrow I_2,-I_4$	2	2	2/2 (1)
$-I_1,-I_4,-I_5,-I_6 \Rightarrow I_2,-I_3$	2	2	2/2 (1)
$I_2,-I_3,-I_4,-I_5 \Rightarrow -I_1,-I_6$	2	2	2/2 (1)
$I_2,-I_3,-I_4,-I_6 \Rightarrow -I_1,-I_5$	2	3	2/3 (0.67)
$I_2,-I_3,-I_5,-I_6 \Rightarrow -I_1,-I_4$	2	2	2/2 (1)
$I_2,-I_4,-I_5,-I_6 \Rightarrow -I_1,-I_3$	2	2	2/2 (1)
$-I_3,-I_4,-I_5,-I_6 \Rightarrow -I_1,I_2$	2	2	2/2 (1)
$-I_1,I_2,-I_3,-I_4,-I_5 \Rightarrow -I_6$	2	2	2/2 (1)
$-I_1,I_2,-I_3,-I_4,-I_6 \Rightarrow -I_5$	2	3	2/3 (0.67)
$-I_1,I_2,-I_3,-I_5,-I_6 \Rightarrow -I_4$	2	2	2/2 (1)
$-I_1,I_2,-I_4,-I_5,-I_6 \Rightarrow -I_3$	2	2	2/2 (1)
$-I_1,-I_3,-I_4,-I_5,-I_6 \Rightarrow I_2$	2	2	2/2 (1)
$I_2,-I_3,-I_4,-I_5,-I_6 \Rightarrow -I_1$	2	2	2/2 (1)

Table 21 – Final set of inverse association rules

Now, after mining all possible association rules, and their confidences, we have to create a threshold α that represents the lowest confidences of the association rules we want. If we pick, for example, α=0.75 (which means that there is at least 75% chance that the association rules really exist), then our final association rules will be: $I_2,-I_5 \Rightarrow -I_1,-I_3,-I_4,-I_6$, $-I_1,I_2,-I_5 \Rightarrow -I_3,-I_4,-I_6$, $-I_1,-I_3,-I_5 \Rightarrow I_2,-I_4,-I_6$, $-I_1,-I_4,-I_5 \Rightarrow I_2,-I_3,-I_6$, $I_2,-I_3,-I_5 \Rightarrow -I_1,-I_4,-I_6$, $I_2,-I_4,-I_5 \Rightarrow -I_1,-I_3,-I_6$, $I_2,-I_5,-I_6 \Rightarrow -I_1,-I_3,-I_4$, $-I_3,-I_4,-I_5 \Rightarrow -I_1,I_2,-I_6$, $-I_4,-I_5,-I_6 \Rightarrow -I_1,I_2,-I_3$, $-I_1,I_2,-I_3,-I_5 \Rightarrow -I_4,-I_6$, $-I_1,I_2,-I_4,-I_5 \Rightarrow -I_3,-I_6$, $-I_1,I_2,-I_5,-I_6 \Rightarrow -I_3,-I_4$, $-I_1,-I_3,-I_4,-I_5 \Rightarrow I_2,-I_6$, $-I_1,-I_3,-I_5,-I_6 \Rightarrow I_2,-I_4$, $-I_1,-I_4,-I_5,-I_6 \Rightarrow I_2,-I_3$, $I_2,-I_3,-I_4,-I_5 \Rightarrow -I_1,-I_6$, $I_2,-I_3,-I_5,-I_6 \Rightarrow -I_1,-I_4$, $I_2,-I_4,-I_5,-I_6 \Rightarrow -I_1,-I_3$, $-I_3,-I_4,-I_5,-I_6 \Rightarrow -I_1,I_2$, $-I_1,I_2,-I_3,-I_4,-I_5 \Rightarrow -I_6$, $-I_1,I_2,-I_3,-I_5,-I_6 \Rightarrow -I_4$, $-I_1,I_2,-I_4,-I_5,-I_6 \Rightarrow -I_3$, $-I_1,-I_3,-I_4,-I_5,-I_6 \Rightarrow I_2$ and $I_2,-I_3,-I_4,-I_5,-I_6 \Rightarrow -I_1$, which have a confidence of 1 (all the association rules that have confidence equal to and higher than 0.75).

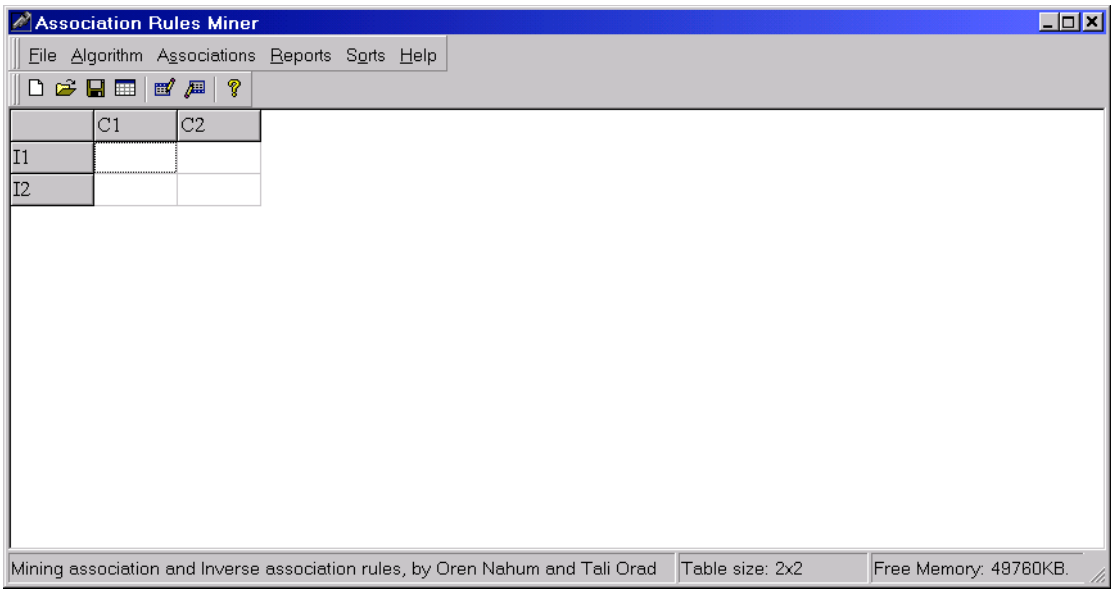
4. Implementation of the Algorithms

This section describes the “Association Rules Miner” program, in terms of screens, and the way the user can and should operate the program. We can refer to this section as the “Association Rules Miner” user guide.

While showing and explaining the “Association Rules Miner” program, we’ll use as an example the same data as presented in the department store example (examples 1 and 2).

4.1. Data Insertion

When starting the program, we’ll see the following screen.



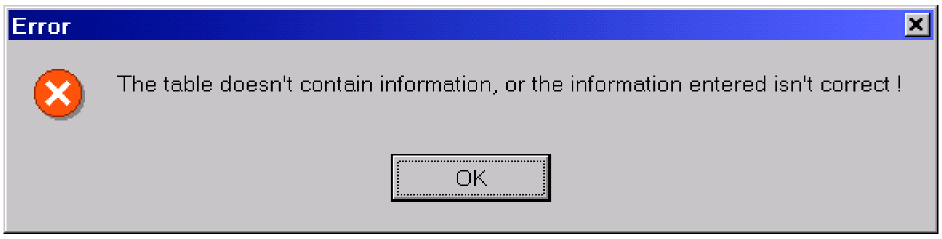
Mining association and Inverse association rules, by Oren Nahum and Tali Orad | Table size: 2x2 | Free Memory: 49760KB.

Figure 4 – The main form of the program, the *Data Table*

This is the main screen of the program in which the data is stored.

Before starting mining association rules, it is essential to enter the *correct data* into the table located on the main screen. Each one of the table’s cells *must* contain data, with values of either 1 or 0.

When starting the program the table contains no data, and the entire table’s cells are empty. Trying to start mining in this stage will bring the following error message: “*The table doesn’t contain information, or the information entered isn’t correct*”, as shown in Figure 5.


Error

The table doesn't contain information, or the information entered isn't correct !

Figure 5 – Main screen error message

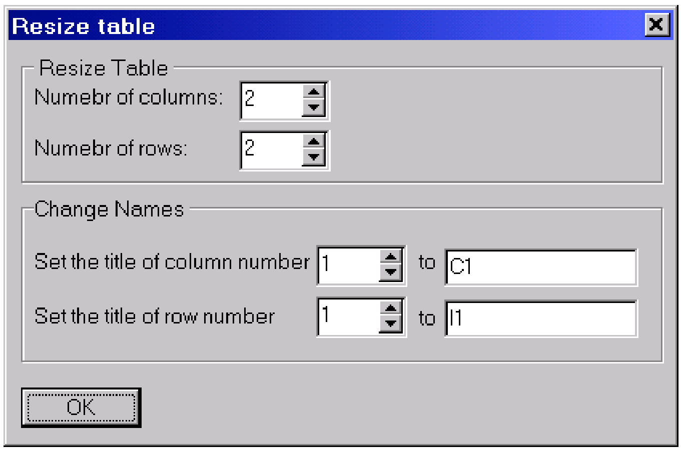
Filling the table with data can be done in several ways.

1. Using the mouse or keyboard to navigate to the desired cell, and then filling it with the value of 0 or 1, by pressing the “0” key or the “1” key, pressing any other keys results with no cell change. This way it is possible to fill every one of the table’s cells with values of 0 or 1.
2. The data table can be saved by clicking the “Save” button, or choosing the menu File|Save command, and supplying a name by which the table will be saved. We can fill our data table, by loading such a saved file. This can be done by clicking the “Load” button, or choosing the File|Load menu command and then entering the name of the saved table (the file name).
3. A third way of filling our data table is to choose the File|Fill with random values menu command. This fills the table with random values of 0 and 1. We added this option in order to make it possible to test the application more easily with different data tables.

4.2. Resizing the Data Table

When starting the application, by default we get an empty 2 by 2 data table. Such data table, obviously can’t represent any real-life information. Therefore, it is possible to change the number of columns and rows of the data table by using the “Resize table” dialogue.

Clicking the “Resize table” button, or using the File|Resize table menu command, brings the following dialogue.


Resize table

Resize Table

Numebr of columns: 2

Numebr of rows: 2

Change Names

Set the title of column number 1 to C1

Set the title of row number 1 to I1

Figure 6 – The "Resize table" dialogue

As we can see, the "Resize table" dialogue is divided into two sections. The first section (the upper one), is in charge of the table's size. The second one (the lower one) is in charge of the columns' and rows' names.

By default, when adding new rows or columns, the new columns will be named "I" adding the number of the column, and the new rows will be named "C" adding the number of the row. We can change the name of the columns and rows by choosing each column and row we want to rename and entering the new name. After such a renaming, we must press the "OK" button, otherwise the changes we made will be lost.

The number of columns can vary from 2 to 30, and the number of rows can vary from 2 to 1,000,000, which can form a data table of $3 \cdot 10^7$ cells.

4.3. Algorithms

The "Association Rules Minder" program implements two mining algorithms, both of them are able to mine regular and inverse association rules. The first algorithm is the "Native" algorithm. The "Native" algorithm uses the straightforward approach for generating the itemsets. In each stage, it simply generates all possible itemsets, and then compares them to the Illegal List. The second algorithm is the Apriori algorithm. When choosing the Apriori algorithm, the "Association Rules Miner" program uses the Apriori algorithms for generating the itemsets.

Choosing the mining algorithm is done by selecting either the Algorithm|Native menu for the "Native" algorithm. Alternatively, selecting the Algorithm|Apriori menu causes the "Association Rules Miner" to use the Apriori algorithm.

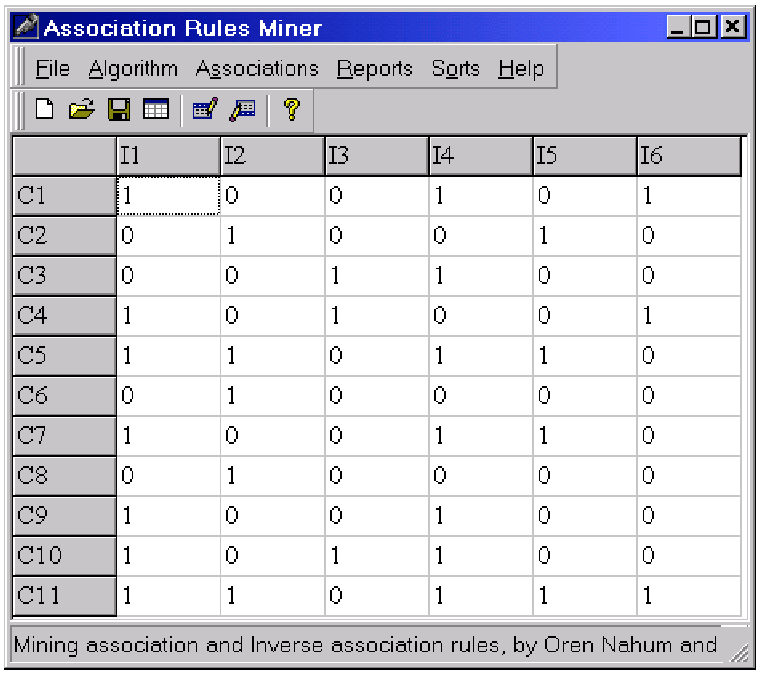
By default, the "Association Rules Miner" uses the Apriori algorithm. We can see which algorithm is currently being used by the program by clicking on the Algorithm menu. Clicking on the Algorithm menu pops up the algorithm menu, on that menu the currently used algorithm is marked with "v", which appears beside it.

4.4. The Mining Procedure

After we have chosen the algorithm we want to use, and filled the data table with the appropriate data, we can start mining association and inverse association rules.

There are two optional ways for the mining procedure. The first one is the manual procedure, which means that the program shows every stage of the mining algorithm, waiting for the user to choose his action, which can be one of the following: Stop – in order to stop the mining procedure, Next – in order to proceed to the next stage or Previous – in order to return to the previous stage. An alternative way is using the automatic mining option. Using the automatic mining option causes the program to automatically proceed to the next stage whenever the manual procedure was suppose to wait for the users reaction.

Suppose we want to use the "Association Rules Miner" program to look for the association rules we found in example 1. The first stage will be filling the data table with the information. The main screen, then, should look like this:



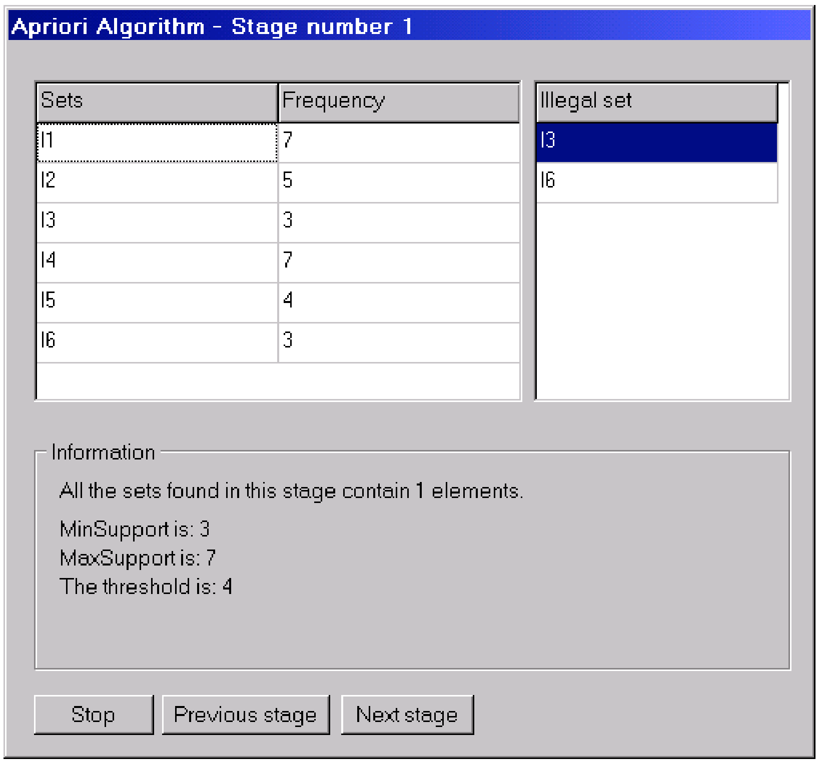
	I1	I2	I3	I4	I5	I6
C1	1	0	0	1	0	1
C2	0	1	0	0	1	0
C3	0	0	1	1	0	0
C4	1	0	1	0	0	1
C5	1	1	0	1	1	0
C6	0	1	0	0	0	0
C7	1	0	0	1	1	0
C8	0	1	0	0	0	0
C9	1	0	0	1	0	0
C10	1	0	1	1	0	0
C11	1	1	0	1	1	1

Mining association and Inverse association rules, by Oren Nahum and

Figure 7 - The main screen when the data table contain the information used in example 1

After filling the data table, we can proceed with the mining procedure. As we said we can choose between manual and automatic procedures for both the regular and inverse association rules mining procedures. The manual operation can be started by pressing the "Association rules" button, or choosing the Association|Find association rules menu command for regular association rules mining. In addition, by pressing the "Inverse association rules" button, or choosing the Association|Find inverse association rules menu command for inverse association rules mining. The automatic mining operation can be started only by choosing the Association|Find association rules automatically menu command for regular association rules mining, or by choosing the Association|Find inverse association rules automatically menu command for inverse association rules mining. Automatic operation is not available via buttons.

Suppose we choose to use the manual mining operation. After pressing the "Association rules" button, or choosing the Association|Find association rules menu command, the program starts mining association rules. After it has finished the first stage of the mining operation, the program would show the following screen:


Apriori Algorithm - Stage number 1

Sets	Frequency
I1	7
I2	5
I3	3
I4	7
I5	4
I6	3

Illegal set
I3
I6

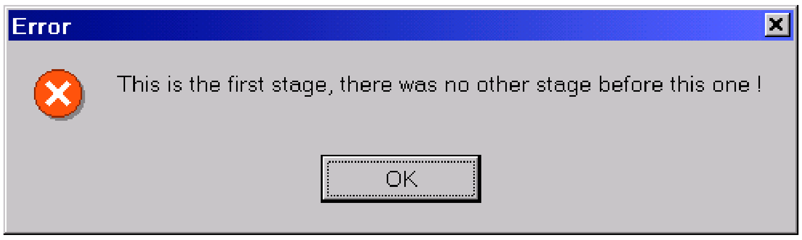
Information

All the sets found in this stage contain 1 elements.

MinSupport is: 3
MaxSupport is: 7
The threshold is: 4

Figure 8 - The first stage of the mining operation

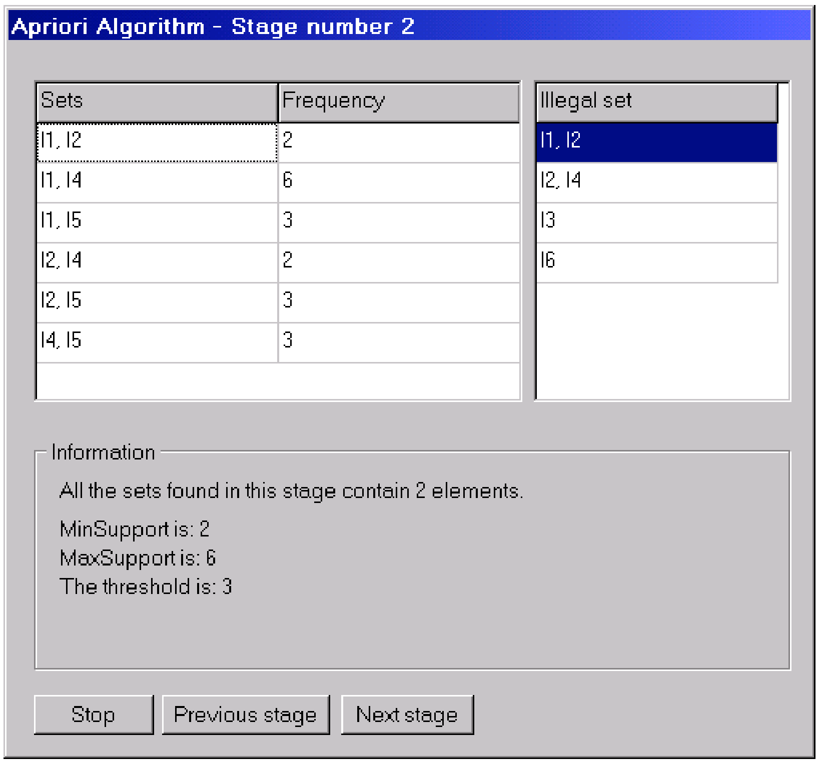
This screen shows all the itemsets of one element found during the mining operation. It also shows all the itemsets that the illegal list contains. Since this screen will follow us during the entire mining operation, it provides the following options. Stop – in order to stop the mining operation and return to the main screen. Next – in order to proceed to the next stage, and Previous – in order to return to the previous stage. In this stage, the first stage, pressing the "Previous stage" button results with the following message: *This is the first stage, there was on other stage before this one !*


Error

This is the first stage, there was no other stage before this one !

Figure 9 - First stage Previous option error message

Pressing the "Next stage" button, the program will continue to the next stage. Eventually the program will show the following screen.



Sets	Frequency
I1, I2	2
I1, I4	6
I1, I5	3
I2, I4	2
I2, I5	3
I4, I5	3

Illegal set
I1, I2
I2, I4
I3
I6

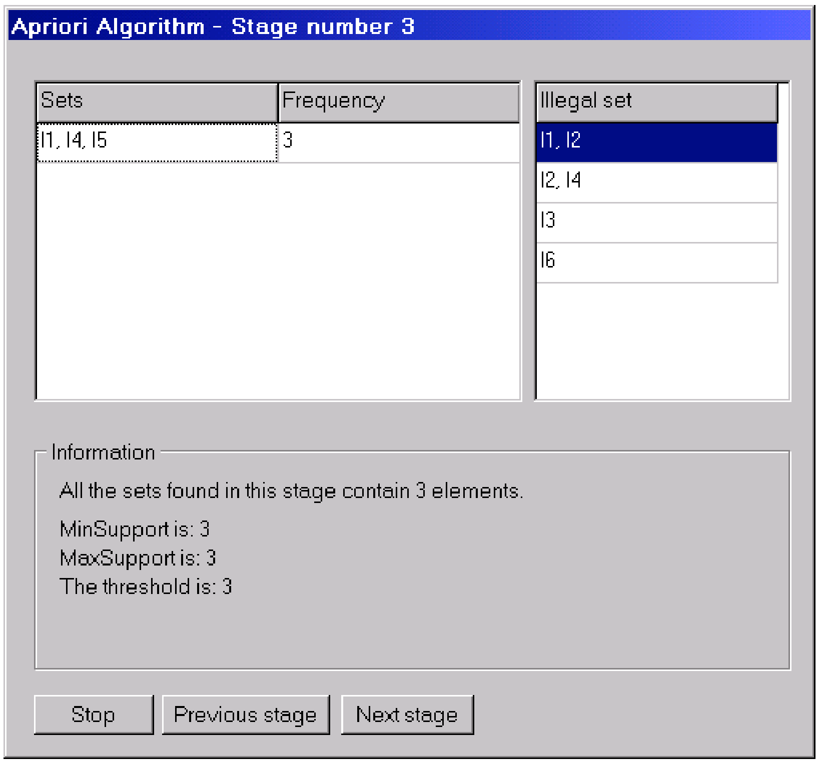
Information

All the sets found in this stage contain 2 elements.

MinSupport is: 2
MaxSupport is: 6
The threshold is: 3

Figure 10 - The second stage of the mining operation

Again, we can choose from the Stop, Previous stage and Next stage options. Pressing the "Next stage" button, will produce the next screen, which is the last stage of our example.



Sets	Frequency
I1, I4, I5	3

Illegal set
I1, I2
I2, I4
I3
I6

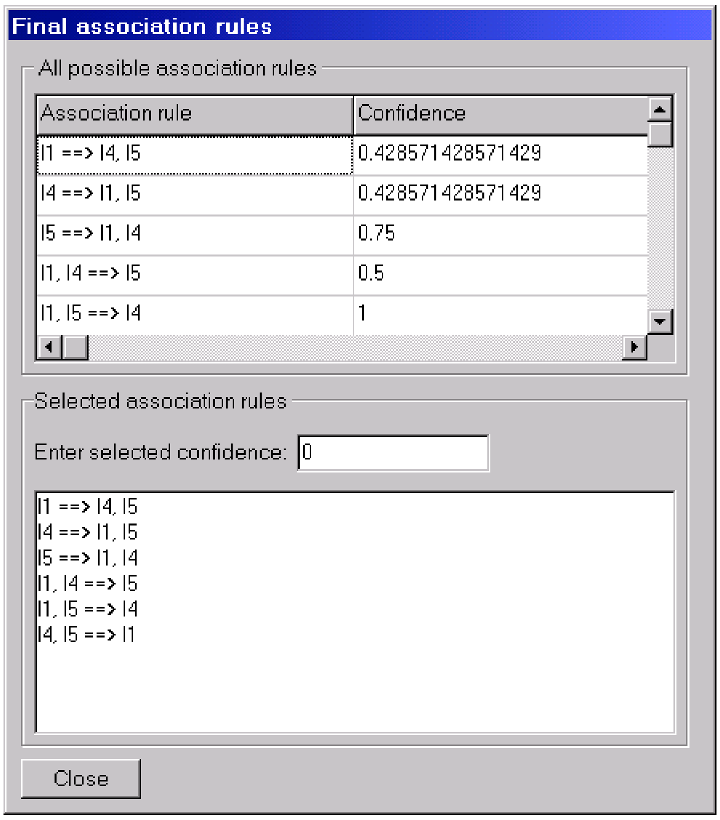
Information

All the sets found in this stage contain 3 elements.

MinSupport is: 3
MaxSupport is: 3
The threshold is: 3

Figure 11 - The Third stage of the mining operation

Since this was the last stage of our example, pressing the “Next stage” button again should generate all the association rules found by the program, as the next figure shows.



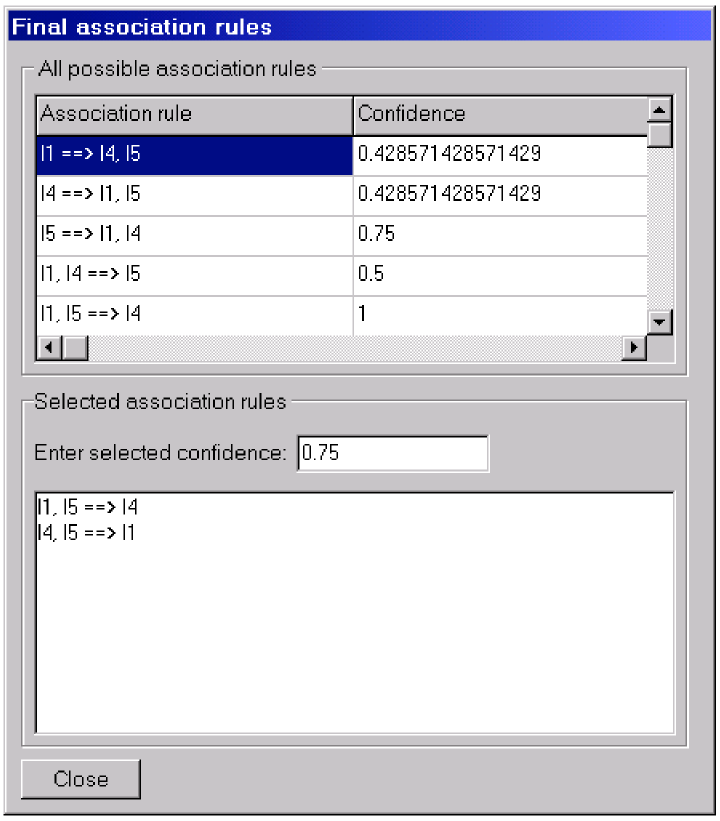
Association rule	Confidence
I1 ==> I4, I5	0.428571428571429
I4 ==> I1, I5	0.428571428571429
I5 ==> I1, I4	0.75
I1, I4 ==> I5	0.5
I1, I5 ==> I4	1

Enter selected confidence: 0

```
I1 ==> I4, I5
I4 ==> I1, I5
I5 ==> I1, I4
I1, I4 ==> I5
I1, I5 ==> I4
I4, I5 ==> I1
```

Figure 12 - Final association rules found by the "Association Rules Miner"

After mining all the possible association rules, we can let the program know what is our desired confidence, by entering the confidence in the edit box. Automatically, as we type, the program displays all association rules with a confidence higher than or equal to the one we entered.



Association rule	Confidence
I1 ==> I4, I5	0.428571428571429
I4 ==> I1, I5	0.428571428571429
I5 ==> I1, I4	0.75
I1, I4 ==> I5	0.5
I1, I5 ==> I4	1

Enter selected confidence: 0.75

```
I1, I5 ==> I4
I4, I5 ==> I1
```

Figure 13 - Specifying a desired confidence

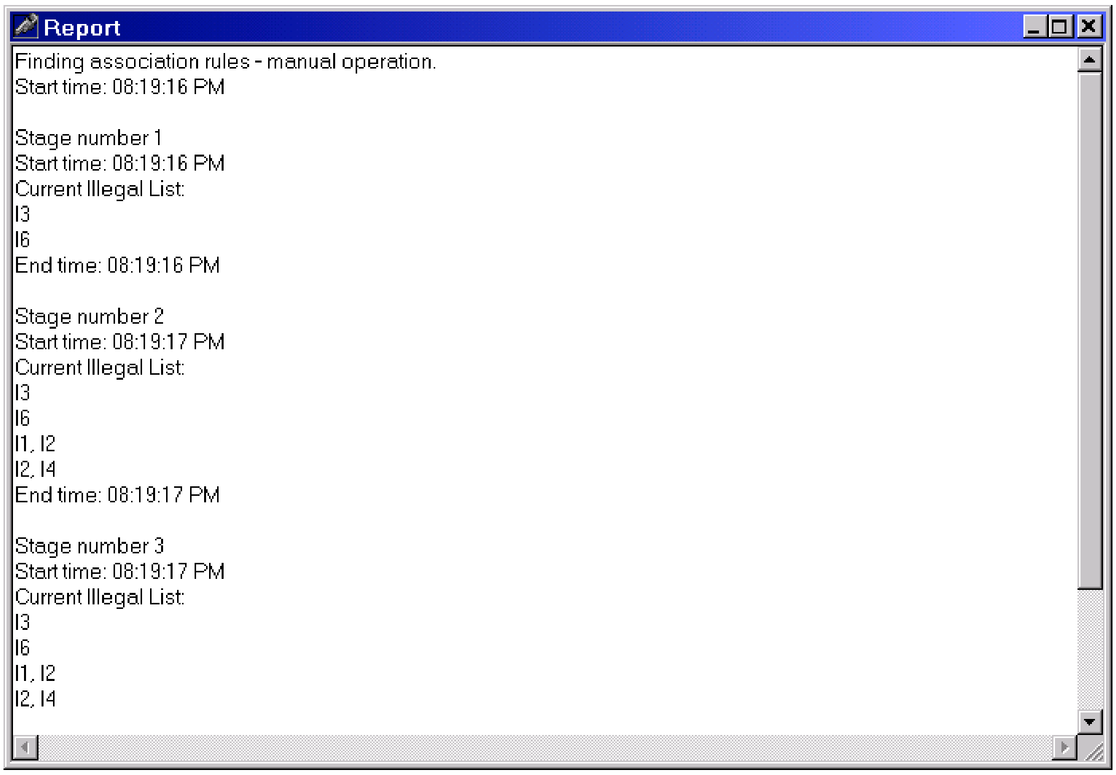
Some options are available for the mining procedure. By default the program sorts the Illegal List and the Itemsets list in every stage of the mining procedure. It is possible to disable (and enable) each one of those sorts in order to increase the mining procedure (sorting simply takes time, and it is not an essential part of the algorithms). From the Sorts menu, it is possible to disable or enable the sort of the Itemsets, by choosing the Sort Itemsets Table commands. This way, we can disable or enable the sort of the Illegal List, by choosing the Sort Illegal List Table command.

Sorting is also available for the association rules table. It is also possible to disable and enable the sort of the association rules table, by choosing the Sorts|Sort Associations Table menu command.

4.5. Reports

The “Association Rules Miner” program can generate a report of the mining operation. By default a report is generated for each mining operation, but the report can be disabled or enabled by using the Report|Generate reports menu command.

In order to view a report, the user should use the Reports|Show last report menu command, which will bring a screen similar to the one shown in Figure 14.



```
Finding association rules - manual operation.
Start time: 08:19:16 PM

Stage number 1
Start time: 08:19:16 PM
Current Illegal List:
I3
I6
End time: 08:19:16 PM

Stage number 2
Start time: 08:19:17 PM
Current Illegal List:
I3
I6
I1, I2
I2, I4
End time: 08:19:17 PM

Stage number 3
Start time: 08:19:17 PM
Current Illegal List:
I3
I6
I1, I2
I2, I4
```

Figure 14 - The report generated while mining association rules

It is possible to print the report using the Reports|Print last report menu command.

4.6. The About Box, and More

As with other program, our program has a help menu.

The help menu provide among other things an About command, which shows the About dialogue box as shown in Figure 15.

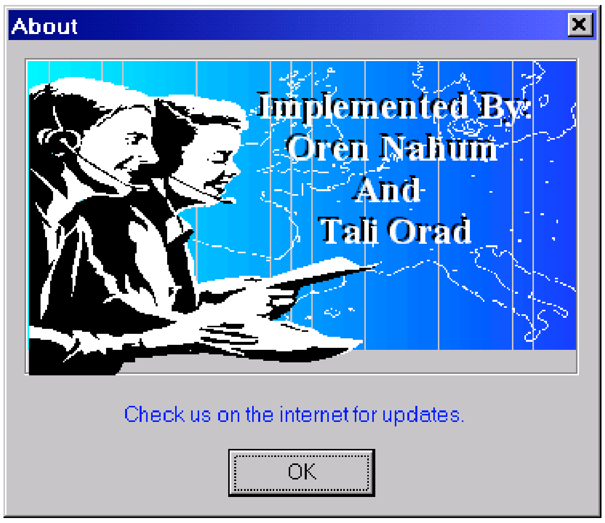


Figure 15 – The about dialogue

Other options available from the Help menu is a link for the association rules miner home page, which is available from the Help|Association Rules Miner Homepage menu command. Using this command launches the default browser, and sets it to show the association rules miner homepage.

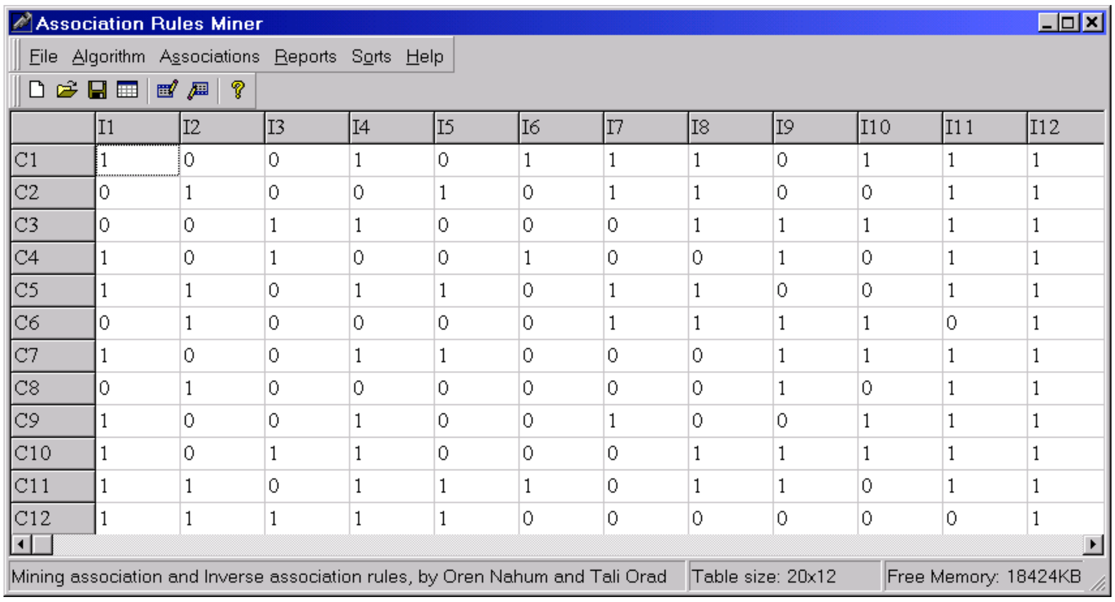
Another option is to send an e-mail to the author of the association rules miner. This is available by using the Help|Send us feedback vial e-mail menu command. This command launches the default e-mail client setting to send e-mail to us, the authors.

5. The Source Code

We choose *Delphi* as our development language. Delphi, currently in its 5th version, is a rapid application development (RAD) tool from Borland International. Delphi uses Pascal, actually if we want to be correct it uses Object-Pascal as its language. Object-Pascal is a new version of the well-known programming language Pascal, which adds the functionality of object oriented programming to Pascal. Object Pascal's implementation of the object oriented programming, and the compiler distributed with Delphi, results in very small and fast applications, which can easily compete with any C++ applications.

Applications developed with Delphi are small in size and also do not require any special DLLs (that means that we can distribute our application as a single executable file).

5.1. DataInuptUnit Unit



	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12
C1	1	0	0	1	0	1	1	1	0	1	1	1
C2	0	1	0	0	1	0	1	1	0	0	1	1
C3	0	0	1	1	0	0	0	1	1	1	1	1
C4	1	0	1	0	0	1	0	0	1	0	1	1
C5	1	1	0	1	1	0	1	1	0	0	1	1
C6	0	1	0	0	0	0	1	1	1	1	0	1
C7	1	0	0	1	1	0	0	0	1	1	1	1
C8	0	1	0	0	0	0	0	0	1	0	1	1
C9	1	0	0	1	0	0	1	0	0	1	1	1
C10	1	0	1	1	0	0	0	1	1	1	1	1
C11	1	1	0	1	1	1	0	1	1	0	1	1
C12	1	1	1	1	1	0	0	0	0	0	0	1

Mining association and Inverse association rules, by Oren Nahum and Tali Orad | Table size: 20x12 | Free Memory: 18424KB

```
unit DataInputUnit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Grids, ComCtrls, ToolWin, ExtCtrls, ImgList, Menus, Sets,ShellAPI,
  LinkedList,HashTable;

type
  TDataInputForm = class(TForm)
    Data: TStringGrid;
    MainMenu: TMainMenu;
    File1: TMenuItem;
    NewMenu: TMenuItem;
    Open1: TMenuItem;
    Save1: TMenuItem;
    N1: TMenuItem;
    Exit1: TMenuItem;
    About1: TMenuItem;
    OpenDialog1: TOpenDialog;
    SaveDialog1: TSaveDialog;
    Associations: TMenuItem;
    Findassociationrules1: TMenuItem;
    Findinverseassociationrules1: TMenuItem;
    StatusBar1: TStatusBar;
    ImageList1: TImageList;
    N2: TMenuItem;
    Resizetable1: TMenuItem;
    Findassociationrulesautomaticly1: TMenuItem;
    Findinverseassociationrulesautomaticly1: TMenuItem;
    Sorts1: TMenuItem;
    SortItemsetsTable1: TMenuItem;
    SortIllegalListTable1: TMenuItem;
    SortAssociationsTable1: TMenuItem;
    AssociationRulesMinerHomepage1: TMenuItem;
    SendusfeadbackviaEMail1: TMenuItem;
    N3: TMenuItem;
    About2: TMenuItem;
    Reports1: TMenuItem;
    Showlastreport1: TMenuItem;
    Generatereports1: TMenuItem;
    Printlastreport1: TMenuItem;
    ImageList2: TImageList;
    Algorithm1: TMenuItem;
    Native1: TMenuItem;
    Apriori1: TMenuItem;
    Timer1: TTimer;
    ControlBar: TControlBar;
    ToolBarMenu: TToolBar;
    ToolBarButtons: TToolBar;
    NewTableButton: TToolButton;
    LoadTableButton: TToolButton;
    SaveTableButton: TToolButton;
    ResizeTableButton: TToolButton;
    ToolButton7: TToolButton;
    AssociationRulesButton: TToolButton;
    InverseAssociationRulesButton: TToolButton;
    ToolButton10: TToolButton;
    AboutBoxButton: TToolButton;
    ControlBarPopupMenu: TPopupMenu;
    MainMenu2: TMenuItem;
    ToolBar1: TMenuItem;
    N4: TMenuItem;
    Fillwithrandomvalues1: TMenuItem;
    procedure LoadTableClick(Sender: TObject);
    procedure SaveTableClick(Sender: TObject);
    procedure AssociationRulesButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Exit1Click(Sender: TObject);
    procedure ResizeTableClick(Sender: TObject);
    procedure InverseAssociationRulesButtonClick(Sender: TObject);
    procedure Findassociationrulesautomaticly1Click(Sender: TObject);
    procedure Findinverseassociationrulesautomaticly1Click(
      Sender: TObject);
    procedure NewTableButtonClick(Sender: TObject);
    procedure SortItemsetsTable1Click(Sender: TObject);
    procedure SortIllegalListTable1Click(Sender: TObject);
    procedure SortAssociationsTable1Click(Sender: TObject);
    procedure DataKeyPress(Sender: TObject; var Key: Char);
    procedure AboutBoxButtonClick(Sender: TObject);
    procedure AssociationRulesMinerHomepage1Click(Sender: TObject);
    procedure SendusfeadbackviaEMail1Click(Sender: TObject);
    procedure Showlastreport1Click(Sender: TObject);
    procedure Generatereports1Click(Sender: TObject);
    procedure Apriori1Click(Sender: TObject);
    procedure Native1Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure Timer1Timer(Sender: TObject);
    procedure MainMenu2Click(Sender: TObject);
    procedure ControlBarPopupMenuPopup(Sender: TObject);
    procedure ToolBar1Click(Sender: TObject);
    procedure Fillwithrandomvalues1Click(Sender: TObject);
  private
```

```
    { Private declarations }
  public
    { Public declarations }
  end;

var
  DataInputForm : TDataInputForm;
  Threshold     : integer;
  n             : Integer;
  FindInverse   : Boolean;
  FindAuto      : Boolean;
  IllegalList   : TLinkedSet;
  ItemSets      : THashTable;

implementation

uses AprioriUnit, AboutUnit, ResizeTableUnit, ReportUnit,NativeUnit;

{$R *.DFM}
```

Clicking the "Load" button, or choosing the File|Load menu, brings the open file dialogue, from which we choose which data file we want to load. After choosing the file, the file is being loaded to the main memory. The `LoadTableClick` is handling the load operation. The data file is a text file, in which the first two line are storing the data size, as it is represented as a two dimensional table.

```
procedure TDataInputForm.LoadTableClick(Sender: TObject); // Loads a saved data table
var
  F   : TextFile;
  i,j : Integer;
  s   : String;
begin
  Application.ProcessMessages;
  if OpenDialog1.Execute then
  begin
    AssignFile (F,OpenDialog1.FileName);
    reset (F);
    readln (F,S);
    Data.ColCount:=StrToInt(S);
    readln (F,S);
    Data.RowCount:=StrToInt(S);
    for i:=0 to Data.ColCount-1 do
      for j:=0 to Data.RowCount-1 do
      begin
        readln (F,S);
        Data.Cells[i,j]:=S;
      end;
    CloseFile (F);
  end;
  StatusBar1.Panels[1].Text:='Table size:
'+IntToStr(Data.ColCount-1)+'x'+IntToStr(Data.RowCount-1);
end;
```

Clicking the "Save" button, or choosing the File|Save menu, brings the save file dialogue, in which we specify the name of the data file. Then, the data is being saved as a text file. The `SaveTableClick` is handling the save operation.

```
procedure TDataInputForm.SaveTableClick(Sender: TObject); // Saves a data table
var
  F   : TextFile;
  i,j : Integer;
begin
  Application.ProcessMessages;
  if SaveDialog1.Execute then
  begin
    if Pos('.',SaveDialog1.FileName)=0 then
      SaveDialog1.FileName:=SaveDialog1.FileName+'.ARM';
    AssignFile (F,SaveDialog1.FileName);
    rewrite (F);
    writeln (F,Data.ColCount);
    writeln (F,Data.RowCount);
    for i:=0 to Data.ColCount-1 do
      for j:=0 to Data.RowCount-1 do
        writeln (F,Data.Cells[i,j]);
    CloseFile (F);
  end;
end;
```

Clicking the “Association rules” button, or choosing the Associations|Find association rules menu, starts the mining operation. We are first checking if our data table is containing a valid data, if it isn’t, we are showing an error message. If the data is valid, we are initializing the report, checking which algorithm to execute, and executing the algorithm.

```
procedure TDataInputForm.AssociationRulesButtonClick(Sender: TObject);
var
  x,y : integer;
  Error : boolean;
begin
  Error:=False;
  FindInverse:=False;
  FindAuto:=False;
  for x:=1 to Data.ColCount-1 do
    for y:=1 to Data.RowCount-1 do
      if (Data.Cells[x,y]<>'0') and (Data.Cells[x,y]<>'1') then Error:=True;
  if Not(Error) then
  begin
    ReportForm.ReportMemo.Clear;
    ReportForm.ReportMemo.Lines.Add('Finding association rules - manual operation.');
    ReportForm.ReportMemo.Lines.Add('Start time: '+TimeToStr(Time));
    n:=1;
    FreeLinkedList(IllegalList);
    // Choose the right algorithm
    if Native1.Checked=True then
      NativeForm.Show
    else if Apriori1.Checked=True then
      AprioriForm.Show;
  end
  else
  begin
    MessageDlg('The table doesn''t contain information, or the information entered
isn’’t correct !', mtError    ,[mbOk], 0);
  end;
end;
```

Every form in Delphi as an event called Create. This event is fired when the form is created. The procedure `FormCreate` is being executed, whenever this event is fired, whenever the form is created. We are using this procedure to initialize some variables in our program, such as the ItemSets’ hash table, and the Illegal List’s linked list. We are also building the office 97 look alike menu in this procedure.

```
procedure TDataInputForm.FormCreate(Sender: TObject);
var
  I,ToolSize : Integer;
  tb         : TToolButton;
begin
  n:=0;
  InitHashTable(ItemSets);
  InitLinkedList(IllegalList);

  ToolSize := 0;
  for I := MainMenu.Items.Count - 1 downto 0 do
  begin
    tb := TToolButton.Create (ToolBarMenu);
    tb.Parent := ToolBarMenu;
    tb.AutoSize := True;
    tb.Grouped := True;
    tb.Caption := MainMenu.Items[I].Caption;
    tb.MenuItem := MainMenu.Items[I];
    Inc (ToolSize, tb.Width);
  end;
  // size the menu toolbar
  ToolBarMenu.Width := ToolSize;
  // hide the standard menu, using the form's Menu property
  Menu := nil;
end;
```

The `Exit1Click` procedure implements the File|Exit menu operation.

```
procedure TDataInputForm.Exit1Click(Sender: TObject);
begin
  Application.Terminate;
end;

procedure TDataInputForm.ResizeTableClick(Sender: TObject);
begin
  DataInputForm.Enabled:=False;
  ResizeTableForm.Show;
end;
```

Clicking the “Inverse association rules” button, or choosing the Associations|Find inverse association rules menu, starts the mining operation. We are first checking if our data table is containing a valid data, if it isn’t, we are showing an error message. If the data is valid, we are initializing the report, checking which algorithm to execute, and executing the algorithm. We’re also initializing the illegal list adding to it itemsets, which have both positive and negative attribute.

```
procedure TDataInputForm.InverseAssociationRulesButtonClick(Sender: TObject);
var
  x,y,j : Integer;
  Error : Boolean;
  S : TIntSet;
begin
  FindInverse:=True;
  FindAuto:=False;
  Error:=False;
  for x:=1 to Data.ColCount-1 do
    for y:=1 to Data.RowCount-1 do
      if (Data.Cells[x,y]<>'0') and (Data.Cells[x,y]<>'1') then Error:=True;
  if Not(Error) then
  begin
    ReportForm.ReportMemo.Clear;
    ReportForm.ReportMemo.Lines.Add('Finding inverse association rules - manual
operation.');
    ReportForm.ReportMemo.Lines.Add('Start time: '+TimeToStr(Time));
    n:=1;
    FreeLinkedList(IllegalList);
    for j:=1 to Data.ColCount-1 do
    begin
      S:=[];
      S:=S+[j];
      S:=S+[j+NEG_POS_ELEM];
      AddLinkedList(IllegalList,S);
    end;
    // Choose the right algorithm
    if Native1.Checked=True then
      NativeForm.Show
    else if Apriori1.Checked=True then
      AprioriForm.Show;
  end
  else
  begin
    MessageDlg('The table doesn''t contain information, or the information entered
isn’’t correct !', mtError    ,[mbOk], 0);
  end;
end;
```

Choosing the Associations|Find association rules automatically menu, starts the automatic mining operation. We are first checking if our data table is containing a valid data, if it isn't, we are showing an error message. If the data is valid, we are initializing the report, checking which algorithm to execute, and executing the algorithm.

```
procedure TDataInputForm.Findassociationrulesautomaticly1Click(Sender: TObject);
var
  x,y : integer;
  Error : boolean;
begin
  Error:=False;
  FindInverse:=False;
  FindAuto:=True;
  for x:=1 to Data.ColCount-1 do
    for y:=1 to Data.RowCount-1 do
      if (Data.Cells[x,y]<>'0') and (Data.Cells[x,y]<>'1') then Error:=True;
  if Not(Error) then
  begin
    ReportForm.ReportMemo.Clear;
    ReportForm.ReportMemo.Lines.Add('Finding association rules - automatic 
operation.');
    ReportForm.ReportMemo.Lines.Add('Start time: '+TimeToStr(Time));
    n:=1;
    FreeLinkedList(IllegalList);
    if Native1.Checked=True then
      NativeForm.Show
    else if Apriori1.Checked=True then
      AprioriForm.Show;
  end
  else
  begin
    MessageDlg('The table doesn''t contain information, or the information entered 
isn’’t correct !', mtError   ,[mbOk], 0);
  end;
end;
```

Choosing the Associations|Find inverse association rules automatically menu, starts the automatic mining operation. We are first checking if our data table is containing a valid data, if it isn't, we are showing an error message. If the data is valid, we are initializing the report, checking which algorithm to execute, and executing the algorithm. We're also initializing the illegal list adding to it itemsets, which have both positive and negative attribute.

```
procedure TDataInputForm.Findinverseassociationrulesautomaticly1Click(
  Sender: TObject);
var
  x,y,j : Integer;
  Error : Boolean;
  S : TIntSet;
begin
  FindInverse:=True;
  FindAuto:=True;
  Error:=False;
  for x:=1 to Data.ColCount-1 do
    for y:=1 to Data.RowCount-1 do
      if (Data.Cells[x,y]<>'0') and (Data.Cells[x,y]<>'1') then Error:=True;
  if Not(Error) then
  begin
    ReportForm.ReportMemo.Clear;
    ReportForm.ReportMemo.Lines.Add('Finding inverse association rules - automatic 
operation.');
    ReportForm.ReportMemo.Lines.Add('Start time: '+TimeToStr(Time));
    n:=1;
    FreeLinkedList(IllegalList);
    for j:=1 to Data.ColCount-1 do
    begin
      S:=[];
      S:=S+[j];
      S:=S+[j+NEG_POS_ELEM];
      AddLinkedList(IllegalList,S);
    end;
    if Native1.Checked=True then
      NativeForm.Show
    else if Apriori1.Checked=True then
      AprioriForm.Show;
  end
  else
  begin
    MessageDlg('The table doesn''t contain information, or the information entered
isn’’t correct !', mtError    ,[mbOk], 0);
  end;
end;
```

Clicking the “New” button, or choosing the File|New menu, will execute the `NewTableButtonClick` procedure. This procedure resizes the data table to 2 by 2, and erases any old data stored in the table.

```
procedure TDataInputForm.NewTableButtonClick(Sender: TObject);
begin
  Data.ColCount:=3;
  Data.RowCount:=3;
  Data.Cells[0,0]:='';
  Data.Cells[1,0]:='C1';
  Data.Cells[2,0]:='C2';
  Data.Cells[0,1]:='I1';
  Data.Cells[0,2]:='I2';
  Data.Cells[1,1]:='';
  Data.Cells[1,2]:='';
  Data.Cells[2,1]:='';
  Data.Cells[2,2]:='';
  Data.FixedCols:=1;
  Data.FixedRows:=1;
  StatusBar1.Panels[1].Text:='Table size: 2x2';
end;
```

The `SortItemsetsTable1Click` turns the SortItemsetsTable flag on and off. It is executed by choosing the Sorts|Sort Itemsets table menu.

```
procedure TDataInputForm.SortItemsetsTable1Click(Sender: TObject);
begin
  SortItemsetsTable1.Checked:=Not(SortItemsetsTable1.Checked);
end;
```

The `SortIllegalListTable1Click` turns the SortIllegalListTable flag on and off. It is executed by choosing the Sorts|Sort Illegal List table menu.

```
procedure TDataInputForm.SortIllegalListTable1Click(Sender: TObject);
begin
  SortIllegalListTable1.Checked:=Not(SortIllegalListTable1.Checked);
end;
```

The `SortAssociationsTable1Click` turns the SortAssociationsTable flag on and off. It is executed by choosing the Sorts|Sort Associations table menu.

```
procedure TDataInputForm.SortAssociationsTable1Click(Sender: TObject);
begin
  SortAssociationsTable1.Checked:=Not(SortAssociationsTable1.Checked);
end;
```

The `DataKeyPress` is executed upon pressing a key while trying to change the data in the data table. The purpose of this procedure is to ensure that the data table contains only vales of zeros or ones.

```
procedure TDataInputForm.DataKeyPress(Sender: TObject; var Key: Char);
begin
```

```
  if Key='0' then
    Data.Cells[Data.Col,Data.Row]:='0'
  else if Key='1' then
    Data.Cells[Data.Col,Data.Row]:='1'
  else
    Key:=#0;
end;
```

Choosing the Help|About menu, or clicking the "About" button, will execute the `AboutBoxButtonClick` procedure. This procedure shows the about box.

```
procedure TDataInputForm.AboutBoxButtonClick(Sender: TObject);
begin
  DataInputForm.Enabled:=False;
  AboutBoxForm.Show;
end;
```

The `AssociationRulesMinerHomepage1Click` procedure opens your default web browser, showing you the association rules miner homepage. This is done by choosing the Help|Association Rules Miner Homepage menu.

```
procedure TDataInputForm.AssociationRulesMinerHomepage1Click(Sender: TObject);
begin
  ShellExecute(GetDesktopWindow(), 'open',
PChar('http://users.surfree.net.il/orennahum/assoc.htm'), nil, nil, SW_SHOWNORMAL);
end;
```

The `SendusfeadbackviaEMail1Click` procedure opens your default e-mail program, setting it to send an e-mail to the author of the "association rules miner" program. This is done by choosing the Help|Send us feedback via e-mail menu.

```
procedure TDataInputForm.SendusfeadbackviaEMail1Click(Sender: TObject);
begin
ShellExecute(GetDesktopWindow(), 'open', PChar('mailto:orennahum@yahoo.com'), nil,
nil, SW_SHOWNORMAL);
end;
```

The `Showlastreport1Click` procedure opens the report form. The report form usually contains the last report, generated during the last mining operation. This procedure is executed when choosing the Reports|Show last report menu.

```
procedure TDataInputForm.Showlastreport1Click(Sender: TObject);
begin
  ReportForm.Show;
end;
```

The `Generatereports1Click` procedure turns on and off the generate reports flag. This procedure is executed upon choosing the Reports|Generate reports menu.

```
procedure TDataInputForm.Generatereports1Click(Sender: TObject);
begin
  Generatereports1.Checked:=Not(Generatereports1.Checked);
end;
```

The `Apriori1Click` procedure sets the mining algorithm to be used as the Apriori algorithm. This procedure is executed upon choosing the Algorithm|Apriori menu.

```
procedure TDataInputForm.Apriori1Click(Sender: TObject);
begin
  if Apriori1.Checked=False then
  begin
    Native1.Checked:=False;
    Apriori1.Checked:=True;
  end;
end;
```

The `Native1Click` procedure sets the mining algorithm to be used as the Native algorithm. This procedure is executed when choosing the Algorithm|Native menu.

```
procedure TDataInputForm.Native1Click(Sender: TObject);
begin
  if Native1.Checked=False then
  begin
    Apriori1.Checked:=False;
    Native1.Checked:=True;
  end;
end;
```

The `FormClose` procedure is executed when the form is closed, when the program is ended. We are using this procedure as an opportunity to free dynamically allocated memory.

```
procedure TDataInputForm.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
  Data.Cells[0,0]:='';
  Data.Cells[1,0]:='C1';
  Data.Cells[2,0]:='C2';
  Data.Cells[0,1]:='I1';
  Data.Cells[0,2]:='I2';
  Data.Cells[1,1]:='';
  Data.Cells[1,2]:='';
  Data.Cells[2,1]:='';
  Data.Cells[2,2]:='';
  FreeHashTable(ItemSets);
  FreeLinkedList(IllegalList);
end;
```

The `Timer1Timer` procedure updated the memory available for the program as reported by windows. This procedure is executed every 1 second.

```
procedure TDataInputForm.Timer1Timer(Sender: TObject);
var
  MemBuf : _MEMORYSTATUS;
begin
  GlobalMemoryStatus(MemBuf);
  StatusBar1.Panels[2].Text:='Free Memory: '+IntToStr(MemBuf.dwAvailPhys div
1024)+'KB.';
end;
```

Since the menu can be dragged and even hidden (as in office), the `MainMenu2Click` is used whenever the user wants to hide or show the menu. This procedure is executed from a pop-up menu.

```
procedure TDataInputForm.MainMenu2Click(Sender: TObject);
begin
  ToolBarMenu.Visible:=Not(ToolBarMenu.Visible);
end;
```

A pop-up menu is used to show or hide the menu and the tool-bar. This procedure is executed when the pop-up menu appears, and updates the status (visible or hidden) of the menu and the tool-bar.

```
procedure TDataInputForm.ControlBarPopupMenuPopup(Sender: TObject);
begin
   MainMenu2.Checked:=ToolBarMenu.Visible;
   ToolBar1.Checked:=ToolBarButtons.Visible;
end;
```

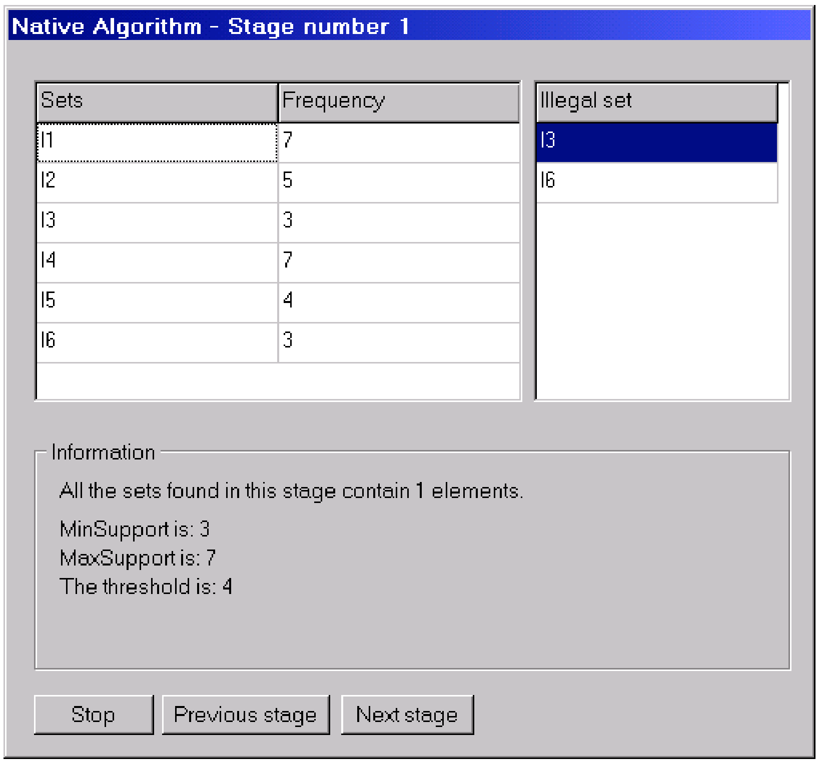
As with the menu, the tool bar can also be dragged and hidden, so the `ControlBarPopupMenuPopup` is used whenever the user wants to hide or show the tool-bar. This procedure is executed from a pop-up menu.

```
procedure TDataInputForm.ToolBar1Click(Sender: TObject);
begin
  ToolBarButtons.Visible:=Not(ToolBarButtons.Visible);
end;
```

The `FillWithRandomValues1Click` procedure is executed upon choosing the File|Fill with random values menu. It fills the data table with random values of 1 and 0. This is useful for testing purposes.

```
procedure TDataInputForm.FillWithRandomValues1Click(Sender: TObject);
var
  x,y : Integer;
begin
  for x:=1 to Data.ColCount-1 do
    for y:=1 to Data.RowCount-1 do
      Data.Cells[x,y]:=char(random(2)+ord('0'));
end;

end.
```

5.2. NativeUnit Unit



Sets	Frequency
I1	7
I2	5
I3	3
I4	7
I5	4
I6	3

Illegal set
I3
I6

Information

All the sets found in this stage contain 1 elements.

MinSupport is: 3
MaxSupport is: 7
The threshold is: 4

```
unit NativeUnit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Grids,DataInputUnit, StdCtrls,Sets,Math, ExtCtrls, LinkedList,HashTable;

type
  TNativeForm = class(TForm)
    SetsTable: TStringGrid;
    NextButton: TButton;
    StopButton: TButton;
    GroupBox1: TGroupBox;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    PreviousButton: TButton;
    StringGrid1: TStringGrid;
    AutoAssociationTimer: TTimer;
    StringGrid2: TStringGrid;
    StringGrid3: TStringGrid;
    procedure FormShow(Sender: TObject);
    procedure FindTheshold(var MinSupport,MaxSupport,Threshold:Integer);
    procedure NextButtonClick(Sender: TObject);
    procedure PreviousButtonClick(Sender: TObject);
    procedure StopButtonClick(Sender: TObject);
    procedure SortTable;
    procedure SortILTable;
    procedure AutoAssociationTimerTimer(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  NativeForm: TNativeForm;

implementation

uses FinalAssociationUnit, ProgressUnit, ReportUnit;

{$R *.DFM}
```

The `FindTheshold` procedure calculates the threshold using the minsuport, maxsuport and the previous threshold.

```
procedure TNativeForm.FindTheshold(var MinSupport,MaxSupport,Threshold:Integer);
var
  i:integer;

  function Min(x,y:Integer):Integer; // Returns the smaller number between two numbers
  begin
    if x<y then Min:=x else Min:=y;
  end;

  function Max(x,y:Integer):Integer; // Returns the higher number between two numbers
  begin
    if x>y then Max:=x else Max:=y;
  end;

begin
  MaxSupport:=StrToInt(SetsTable.cells[1,1]);
  MinSupport:=MaxSupport;
  for i:=2 to SetsTable.RowCount-1 do
  begin
    Application.ProcessMessages;
    if StrToInt(SetsTable.Cells[1,i])>MaxSupport then
      MaxSupport:=StrToInt(SetsTable.Cells[1,i])
    else if StrToInt(SetsTable.Cells[1,i])<MinSupport then
      MinSupport:=StrToInt(SetsTable.Cells[1,i])
  end;

  if n=1 then
  begin
    if MaxSupport>MinSupport then
      Threshold:=Max(2,MinSupport+1)
    else if MaxSupport=MinSupport then
      Threshold:=Max(2,MinSupport);
  end
  else
  begin
    if MaxSupport>MinSupport then
      Threshold:=Min(Max(2,MinSupport+1),Threshold)
    else if MaxSupport=MinSupport then
      Threshold:=Max(2,MinSupport);
  end;
end;
```

The `FormShow` procedure is executed whenever the form becomes visible. It is the main procedure of this form, because is encapsulates the entire Native algorithm. It creates all Itemsets with *n* elements. Then it add all itemsets which have no subsets in the Illegal list to the Itemsets table, calculates the threshold and add Itemsets to the Illegal list according to the threshold.

```
procedure TNativeForm.FormShow(Sender: TObject);
var
  x   : Integer;
  i,j : Integer;
  Se  : TIntSet;
  count : integer;
  min,max:integer;
  S : TIntSet;
  ISA   : TLinkedSetsPtr;
begin
  SetsTable.Visible:=False;
  StringGrid1.Visible:=False;
  AutoAssociationTimer.Enabled:=False;
  NextButton.Enabled:=False; // Disabling all buttons
  PreviousButton.Enabled:=False;
  StopButton.Enabled:=False;
  NativeForm.Caption:='Native Algorithm - Stage number '+IntToStr(n);
  Label1.caption:='All the sets found in this stage contain '+IntToStr(n)+'
elements.';
  Label2.Caption:='MinSupport is: ';
  Label3.Caption:='MaxSupport is: ';
  Label4.Caption:='The threshold is: ';
  ReportForm.ReportMemo.Lines.Add('');
  ReportForm.ReportMemo.Lines.Add('Stage number '+IntToStr(n));
  ReportForm.ReportMemo.Lines.Add('Start time: '+TimeToStr(Time));

  //FreeLinkedList(ItemSets);
  FreeHashTable(ItemSets);

  for i:=1 to DataInputForm.Data.RowCount-1 do // Scan all rows
  begin
    S:=[];
    for j:=1 to DataInputForm.Data.ColCount-1 do
    begin
      if DataInputForm.Data.Cells[j,i]='1' then // Creates a set
        S:=S+[j]
      else
        if FindInverse then
          S:=S+[j+NEG_POS_ELEM];
    end;
    SerIndex:=0;
    PowerSet (S,n,i); // Gets all subsets length n
  end;

  StringGrid1.Cells[0,0]:='Illegal set';
  StringGrid1.RowCount:=1;
  SetsTable.Cells[0,0]:='Sets';
  SetsTable.Cells[1,0]:='Frequency';
  SetsTable.RowCount:=1;

  NativeForm.Enabled:=False;
  ProgressForm.Show;
  ProgressForm.Label1.Caption:='Updating Itemsets table...';
  //ISA:=ItemSets.Head;
  HashTableHead(ItemSets);
  ISA:=ItemSets.DataPtr;
  //for x:=0 to ItemSets.Count-1 do
  for x:=0 to HashTableCount(ItemSets)-1 do
  begin
    //if ItemSets.Count>1 then
    if HashTableCount(ItemSets)>1 then
      //ProgressForm.ProgressBar1.Position:=Trunc(100*x/(ItemSets.Count-1))
      ProgressForm.ProgressBar1.Position:=Trunc(100*x/(HashTableCount(ItemSets)-1))
    else
      ProgressForm.ProgressBar1.Position:=100;
    ProgressForm.Process;
    if ISA^.Data<>[] then
    begin
      count:=0;
      for i:=1 to DataInputForm.Data.RowCount-1 do
      begin
        Application.ProcessMessages;
        Se:=[];
        for j:=1 to DataInputForm.Data.ColCount-1 do
        begin
          Application.ProcessMessages;
          if DataInputForm.Data.Cells[j,i]='1' then
            Se:=Se+[j]
          else
            if FindInverse then
              Se:=Se+[j+NEG_POS_ELEM]; // Creates a set from a line
        end;
        if ISA^.Data<=Se then count:=count+1;
      end;
      if not(HasSubSetInIL(ISA^.Data)) then // This Itemset has no subset which is in
the illegal sets
      begin
        SetsTable.RowCount:=SetsTable.RowCount+1;
        SetsTable.Cells[1,SetsTable.RowCount-1]:=IntToStr(count);
        SetsTable.Cells[0,SetsTable.RowCount-1]:=SetToStr(ISA^.Data);
      end;
    end;
    //ISA:=ISA^.Next;
    HashTableNext(ItemSets);
    ISA:=ItemSets.DataPtr;
  end;
  ProgressForm.Hide;
  NativeForm.Enabled:=True;

  if SetsTable.RowCount>1 then
  begin
    SetsTable.FixedRows:=1;

    FindTheshold(min,max,threshold); // Finds MinSupport, MaxSupport and the Threshold
    Label2.Caption:='MinSupport is: '+IntToStr(Min);
    Label3.Caption:='MaxSupport is: '+IntToStr(Max);
    Label4.Caption:='The threshold is: '+IntToStr(Threshold);

    // Adds Itemsets which their supports are lower than the Threshold
    NativeForm.Enabled:=False;
    ProgressForm.Show;
    ProgressForm.Label1.Caption:='Adding Itemsets to the Illegal List table...';
    //ISA:=ItemSets.Head;
    HashTableHead(ItemSets);
    ISA:=ItemSets.DataPtr;
    //for x:=0 to ItemSets.Count-1 do
    for x:=0 to HashTableCount(ItemSets)-1 do
    begin
      //if ItemSets.Count>1 then
      if HashTableCount(ItemSets)>1 then
        //ProgressForm.ProgressBar1.Position:=Trunc(100*x/(ItemSets.Count-1))
        ProgressForm.ProgressBar1.Position:=Trunc(100*x/(HashTableCount(ItemSets)-1))
      else
        ProgressForm.ProgressBar1.Position:=100;
      ProgressForm.Process;
      if ISA^.Data<>[] then
      begin
        count:=0;
        for i:=1 to DataInputForm.Data.RowCount-1 do // Checks each row
        begin
          Se:=[];
          for j:=1 to DataInputForm.Data.ColCount-1 do // Create the set from the row
          begin
            Application.ProcessMessages;
            if DataInputForm.Data.Cells[j,i]='1' then
              Se:=Se+[j]
            else
            if FindInverse then
              Se:=Se+[j+NEG_POS_ELEM]; // Creates a set from a line
          end;
          if ISA^.Data<=Se then count:=count+1;
        end;
        if (not(HasSubSetInIL(ISA^.Data))) and (count<Threshold) then
        begin
          AddLinkedList(IllegalList,ISA^.Data);
        end;
      end;
      //ISA:=ISA^.Next;
      HashTableNext(ItemSets);
      ISA:=ItemSets.DataPtr;
    end;
    ProgressForm.Hide;
    NativeForm.Enabled:=True;

    //Show Illegal list
    ReportForm.ReportMemo.Lines.Add('Current Illegal List:');
    ISA:=IllegalList.Head;
    for x:=1 to IllegalList.Count do
    begin
      if NumOfElements(ISA^.Data)<=n then
      begin
        Application.ProcessMessages;
        StringGrid1.RowCount:=StringGrid1.RowCount+1;
        StringGrid1.Cells[0,StringGrid1.RowCount-1]:=SetToStr(ISA^.Data);
        ReportForm.ReportMemo.Lines.Add(SetToStr(ISA^.Data));
      end;
      ISA:=ISA^.Next;
    end;
    StringGrid1.FixedRows:=1;
    if DataInputForm.SortIllegalListTable1.Checked then
      SortILTable;
    if DataInputForm.SortItemsetsTable1.Checked then
      SortTable;
    AutoAssociationTimer.Enabled:=True;
  end
  else
  begin
    Hide;
    FinalAssociationForm.Show;
  end;
  NextButton.Enabled:=True;
  PreviousButton.Enabled:=True;
  StopButton.Enabled:=True;
  SetsTable.Visible:=True;
  StringGrid1.Visible:=True;
  ReportForm.ReportMemo.Lines.Add('End time: '+TimeToStr(Time));
end;
```

The `NextButtonClick` procedure is executed whenever the user clicks the “next” button. It shows the next stage of the Apriori algorithm.

```
procedure TNativeForm.NextButtonClick(Sender: TObject);
begin
  n:=n+1; // Increase stage counter
  FormShow(Sender);
end;
```

The `PreviousButtonClick` procedure is executed whenever the user clicks the “previous” button. It shows the previous stage of the Apriori algorithm, or shows a warning massage if this is the first stage.

```
procedure TNativeForm.PreviousButtonClick(Sender: TObject);
```

```
begin
  if n>1 then
  begin
    n:=n-1; // Decrease stage counter
    FormShow(Sender);
  end
  else
  begin
    MessageDlg('This is the first stage, there was no other stage before this one !',
mtError     ,[mbOk], 0);
  end;
end;
```

The `StopButtonClick` procedure is executed whenever the user clicks the “stop” button. It stops the Apriori algorithm, and returns the program to the main screen.

```
procedure TNativeForm.StopButtonClick(Sender: TObject);
begin
  Hide;
end;
```

The `SortTable` procedure sorts the Itemsets table, in every stage of the algorithm, if the sorting was enabled (from the main screen from). This is an implementation of the heap-sort algorithm.

```
procedure TNativeForm.SortTable;
var
  x    : LongWord;
  Temp : String;

  Procedure Heapify (i,Size:LongWord); { "Fixs" a branch in the heap array }
  var
    Largest : LongWord;
    Temp    : String;
  begin
    repeat
      Largest:=i;
      if (2*i<=Size) then
        if (SetsTable.Cells[0,2*i]>SetsTable.Cells[0,i]) then Largest:=2*i;
      if (2*i+1<=Size) then
        if (SetsTable.Cells[0,2*i+1]>SetsTable.Cells[0,Largest]) then Largest:=2*i+1;
      if Largest<>i then
      begin
        Temp:=SetsTable.Cells[0,i];
        SetsTable.Cells[0,i]:=SetsTable.Cells[0,Largest]; { We have to swap the two
variables }
        SetsTable.Cells[0,Largest]:=Temp;
        Temp:=SetsTable.Cells[1,i];
        SetsTable.Cells[1,i]:=SetsTable.Cells[1,Largest]; { We have to swap the two
variables }
        SetsTable.Cells[1,Largest]:=Temp;
        i:=Largest;
      end
      else
        i:=Size+1;
    until i>=Size;
  end;

begin
  NativeForm.Enabled:=False;
  ProgressForm.Show;
  ProgressForm.Label1.Caption:='Sorting Itemsets table...';
  for x:=(SetsTable.RowCount-1 div 2) downto 1 do { "Fixs" the array to a heap form,
from the father variable to the begining of the array }
  begin
    ProgressForm.ProgressBar1.Position:=Trunc(100*((SetsTable.RowCount-1 div
2)-x)/(SetsTable.RowCount-1 div 2 + SetsTable.RowCount-2));
    ProgressForm.Process;
    Heapify (x,SetsTable.RowCount-1);
  end;
  for x:=SetsTable.RowCount-1 downto 2 do
  begin
    ProgressForm.ProgressBar1.Position:=Trunc(100*((SetsTable.RowCount-1 div 2) +
SetsTable.RowCount-2 - x)/(SetsTable.RowCount-1 div 2 + SetsTable.RowCount-2));
    ProgressForm.Process;
    Temp:=SetsTable.Cells[0,x];
    SetsTable.Cells[0,x]:=SetsTable.Cells[0,1]; { We have to swap the two variables }
    SetsTable.Cells[0,1]:=Temp;
    Temp:=SetsTable.Cells[1,x];
    SetsTable.Cells[1,x]:=SetsTable.Cells[1,1]; { We have to swap the two variables }
    SetsTable.Cells[1,1]:=Temp;
    Heapify(1,SetsTable.RowCount-1-(SetsTable.RowCount-x));
  end;


ProgressForm.ProgressBar1.Position:=Trunc(100*(SetsTable.RowCount-1)/(SetsTable.RowCou
nt-1));
    ProgressForm.Process;
  ProgressForm.Hide;
  NativeForm.Enabled:=True;
end;
```

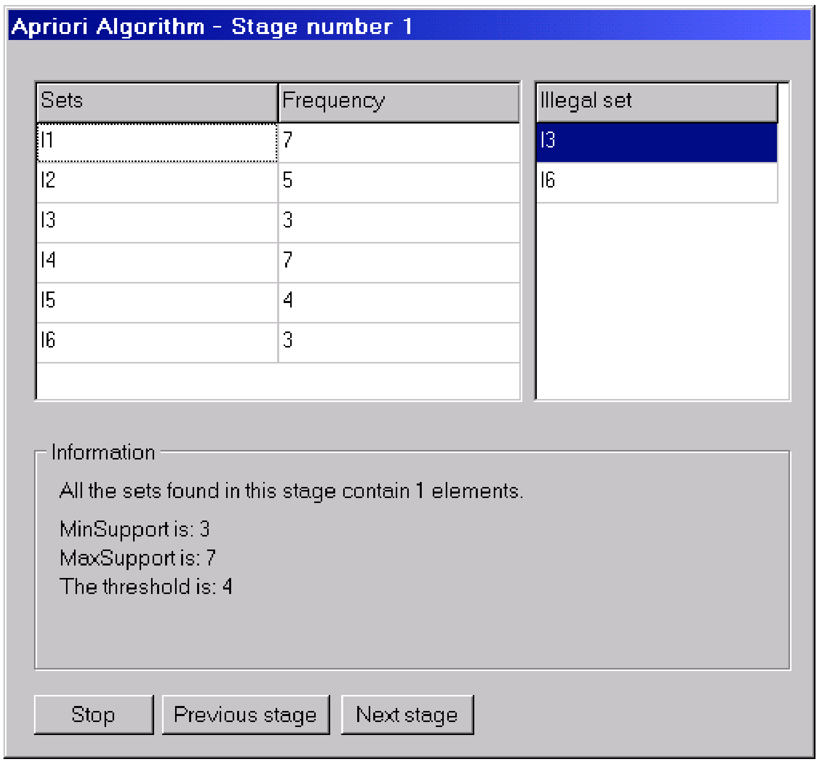
The `SortILTable` procedure sorts the Illegal List table, in every stage of the algorithm, if the sorting was enabled (from the main screen from). This is an implementation of the heap-sort algorithm.

```
procedure TNativeForm.SortILTable;
var
  i,j : integer; { Variables used for array scanning }
  flg : boolean; { A flag }
  Temp: String;
begin
  NativeForm.Enabled:=False;
  ProgressForm.Show;
  ProgressForm.Label1.Caption:='Sorting Illega List table...';
  i:=StringGrid1.RowCount-1; { i gets the size of the array }
  flg:=true; { If the flag is set to true, then the array is not sorted }
  while (i>1) and (flg) do { If i is bigger than 1 and the flag is set to true (the
array isn't sorted) }
  begin

ProgressForm.ProgressBar1.Position:=Trunc(100*(StringGrid1.RowCount-1-i)/(StringGrid1.
RowCount-1));
    ProgressForm.Process;
    flg:=false; { Let's assume the the array is already sorted }
    for j:=1 to i-1 do { We are scanning variables 1 to i-1 of the array }
      if StringGrid1.Cells[0,j]>StringGrid1.Cells[0,j+1] then { If variable j is
bigger than variable j+1 }
      begin
        flg:=true; { The array isn't sorted, another chack is needed }
        Temp:=StringGrid1.Cells[0,j];
        StringGrid1.Cells[0,j]:=StringGrid1.Cells[0,j+1]; { We have to swap the two
variables }
        StringGrid1.Cells[0,j+1]:=Temp;
      end;
    i:=i-1; { Variable i is "sorted", we should chack again from 1 to i-1 }
  end; { The array is sorted }
  ProgressForm.Hide;
  NativeForm.Enabled:=True;
end;

procedure TNativeForm.AutoAssociationTimerTimer(Sender: TObject);
begin
  AutoAssociationTimer.Enabled:=False;
  if FindAuto then
    NextButtonClick(Sender);
end;
```

The `SortTable` procedure sorts the Itemsets table, in every stage of the algorithm, if the sorting was enabled (from the main screen from). This is an implementation of the heap-sort algorithm.

```
procedure TNativeForm.FormCreate(Sender: TObject);
begin
  StringGrid2.Cells[0,0]:='Illegal set';
  StringGrid3.Cells[0,0]:='Sets';
  StringGrid3.Cells[1,0]:='Frequency';
end;

end.
```

5.3. AprioriUnit Unit



Sets	Frequency
I1	7
I2	5
I3	3
I4	7
I5	4
I6	3

Illegal set
I3
I6

Information

All the sets found in this stage contain 1 elements.

MinSupport is: 3
MaxSupport is: 7
The threshold is: 4

```
unit AprioriUnit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Grids,DataInputUnit, StdCtrls,Sets,Math, ExtCtrls, LinkedList,HashTable;

type
  TAprioriForm = class(TForm)
    SetsTable: TStringGrid;
    NextButton: TButton;
    StopButton: TButton;
    GroupBox1: TGroupBox;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    PreviousButton: TButton;
    AprioriILGrid: TStringGrid;
    AutoAssociationTimer: TTimer;
    StringGrid2: TStringGrid;
    StringGrid3: TStringGrid;
    procedure FormShow(Sender: TObject);
    procedure FindTheshold(var MinSupport,MaxSupport,Threshold:Integer);
    procedure NextButtonClick(Sender: TObject);
    procedure PreviousButtonClick(Sender: TObject);
    procedure StopButtonClick(Sender: TObject);
    procedure SortTable;
    procedure SortILTable;
    procedure AutoAssociationTimerTimer(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Create_1_Itemsets;
    procedure Create_n_Itemsets;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  AprioriForm: TAprioriForm;

implementation

uses FinalAssociationUnit, ProgressUnit, ReportUnit;

{$R *.DFM}
```

The `FindTheshold` procedure calculates the threshold using the minsuport, maxsuport and the previous threshold.

```
procedure TAprioriForm.FindTheshold(var MinSupport,MaxSupport,Threshold:Integer);
var
  i:integer;

  function Min(x,y:Integer):Integer; // Returns the smaller number between two numbers
  begin
    if x<y then Min:=x else Min:=y;
  end;

  function Max(x,y:Integer):Integer; // Returns the higher number between two numbers
  begin
    if x>y then Max:=x else Max:=y;
  end;

begin
  MaxSupport:=StrToInt(SetsTable.cells[1,1]);
  MinSupport:=MaxSupport;
  for i:=2 to SetsTable.RowCount-1 do
  begin
    Application.ProcessMessages;
    if StrToInt(SetsTable.Cells[1,i])>MaxSupport then
      MaxSupport:=StrToInt(SetsTable.Cells[1,i])
    else if StrToInt(SetsTable.Cells[1,i])<MinSupport then
      MinSupport:=StrToInt(SetsTable.Cells[1,i])
  end;

  if n=1 then
  begin
    if MaxSupport>MinSupport then
      Threshold:=Max(2,MinSupport+1)
    else if MaxSupport=MinSupport then
      Threshold:=Max(2,MinSupport);
  end
  else
  begin
    if MaxSupport>MinSupport then
      Threshold:=Min(Max(2,MinSupport+1),Threshold)
    else if MaxSupport=MinSupport then
      Threshold:=Max(2,MinSupport);
  end;
end;
```

The `Create_1_Itemsets` procedure is uses to create all Itemsets with 1 element. This is needed for the first stage of the Apriori algorithm.

```
procedure TAprioriForm.Create_1_Itemsets;
var
  x : integer;
begin
```

```
  FreeHashTable(ItemSets);
  for x:=1 to DataInputForm.Data.ColCount-1 do
  begin
    AddHashTable(ItemSets,[x]);
    if FindInverse then
    begin
      AddHashTable(ItemSets,[x+NEG_POS_ELEM]);
    end;
    ProgressForm.Process;
  end;
end;
```

The `Create_n_Itemsets` procedure is uses to create all Itemsets with *n* elements. This is needed for the Apriori algorithm.

```
procedure TAprioriForm.Create_n_Itemsets;
var
  SubsCount : LongWord;
  i,j,x,y : integer;
  Se1 : array[1..MAX_SET_SIZE] of byte;
  Se2 : array[1..MAX_SET_SIZE] of byte;
  Add : Boolean;
  TempISArrayP,NewItemSet : TLinkedSet;
  H1,H2    : TLinkedSetsPtr;
  DataPtr  : TLinkedSetsPtr;
  DataLoc  : Integer;
begin
  SubsCount:=0;
  InitLinkedList(TempISArrayP);
  InitLinkedList(NewItemSet);
  ProgressForm.Show;
  ProgressForm.Label1.Caption:='Creating subsets...';

  if HashTableCount(ItemSets)>1 then
  begin
    // Scans all sets-last
    HashTableHead(ItemSets);
    H1:=ItemSets.DataPtr;
    for i:=0 to HashTableCount(ItemSets)-2 do
    begin
      // Translates the set in H1^.Data to an array Se1[]
      y:=1;
      for x:=1 to MAX_SET_SIZE do
      begin
        Se1[x]:=0;
        if x in H1^.Data then
        begin
          Se1[y]:=x;
          Inc(y);
        end;
        ProgressForm.Process;
      end;
      // Compares set with the rest of the sets
      DataPtr:=ItemSets.DataPtr;
      DataLoc:=ItemSets.DataLoc;
      HashTableNext(ItemSets);
      H2:=ItemSets.DataPtr;
      for j:=i+1 to HashTableCount(ItemSets)-1 do
      begin
        // Translates the set in H2^.Data to an array Se2[]
        y:=1;
        for x:=1 to MAX_SET_SIZE do
        begin
          Se2[x]:=0;
          if x in H2^.Data then
          begin
            Se2[y]:=x;
            Inc(y);
          end;
          ProgressForm.Process;
        end;
        // Checks whether we should add the new combined set
        Add:=True;
        // If we have first n-2 equal elements then add
        for x:=1 to n-2 do
        begin
          if Se1[x]<>Se2[x] then
          begin
            Add:=False;
            break;
          end;
          ProgressForm.Process;
        end;
        // Now do the adding
        if Add then
        begin
          if NumOfElements(H1^.Data+H2^.Data)=n then
            AddLinkedList(TempISArrayP,H1^.Data+H2^.Data);
        end;
        Inc(SubsCount);

ProgressForm.ProgressBar1.Position:=Trunc(SubsCount/(HashTableCount(ItemSets)*(HashTab
leCount(ItemSets)-1)/2)*100);
        ProgressForm.Process;
        HashTableNext(ItemSets);
        H2:=ItemSets.DataPtr;
      end;
      ItemSets.DataPtr:=DataPtr;
      ItemSets.DataLoc:=DataLoc;
      HashTableNext(ItemSets);
      H1:=ItemSets.DataPtr;
    end;
  end;

  if TempISArrayP.Count>0 then
  begin
    Add:=True;
    H1:=TempISArrayP.Head;
    for x:=0 to TempISArrayP.Count-1 do
    begin
      Add:=False;
      for y:=1 to MAX_SET_SIZE do
      begin
        if y in H1^.Data then
        begin
          HashTableHead(ItemSets);
          H2:=ItemSets.DataPtr;
          for i:=0 to HashTableCount(ItemSets)-1 do
          begin
            if H2^.Data=(H1^.Data-[y]) then
            begin
              Add:=True;
              break;
            end;
            HashTableNext(Itemsets);
            H2:=ItemSets.DataPtr;
            ProgressForm.Process;
          end;
          if Add then
            Break;
        end;
        ProgressForm.Process;
      end;
      if Add then
        AddLinkedList(NewItemSet,H1^.Data);
      H1:=H1^.Next;
    end;
  end;
  FreeLinkedList(TempISArrayP);
  FreeHashTable(ItemSets);
  H1:=NewItemSet.Head;
  for x:=1 to NewItemSet.Count do
  begin
    AddHashTable(ItemSets,H1^.Data);
    H1:=H1^.Next;
  end;
  ProgressForm.Hide;
```

```
end;
```

The `FormShow` procedure is executed whenever the form becomes visible. It is the main procedure of this form, because is encapsulates the entire Apriori algorithm. It creates all Itemsets with *n* elements. Then it add all itemsets which have no subsets in the Illegal list to the Itemsets table, calculates the threshold and add Itemsets to the Illegal list according to the threshold.

```
procedure TAprioriForm.FormShow(Sender: TObject);
var
  x   : Integer;
  i,j : Integer;
  Se  : TIntSet;
  count : integer;
  min,max:integer;
  ISA : TLinkedSetsPtr;
begin
  SetsTable.Visible:=False;
  AprioriILGrid.Visible:=False;
  AutoAssociationTimer.Enabled:=False;
  NextButton.Enabled:=False; // Disabling all buttons
  PreviousButton.Enabled:=False;
  StopButton.Enabled:=False;
  AprioriForm.Caption:='Apriori Algorithm - Stage number '+IntToStr(n);
  Label1.caption:='All the sets found in this stage contain '+IntToStr(n)+'
elements.';
  Label2.Caption:='MinSupport is: ';
  Label3.Caption:='MaxSupport is: ';
  Label4.Caption:='The threshold is: ';
  ReportForm.ReportMemo.Lines.Add('');
  ReportForm.ReportMemo.Lines.Add('Stage number '+IntToStr(n));
  ReportForm.ReportMemo.Lines.Add('Start time: '+TimeToStr(Time));

  if n=1 then
    Create_1_Itemsets
  else
    Create_n_Itemsets;

  AprioriILGrid.Cells[0,0]:='Illegal set';
  AprioriILGrid.RowCount:=1;
  SetsTable.Cells[0,0]:='Sets';
  SetsTable.Cells[1,0]:='Frequency';
  SetsTable.RowCount:=1;

  AprioriForm.Enabled:=False;
  ProgressForm.Show;
  ProgressForm.Label1.Caption:='Updating Itemsets table...';
  HashTableHead(ItemSets);
  ISA:=ItemSets.DataPtr;
  for x:=0 to HashTableCount(ItemSets)-1 do
  begin
    if HashTableCount(ItemSets)>1 then
      ProgressForm.ProgressBar1.Position:=Trunc(100*x/(HashTableCount(ItemSets)-1))
    else
      ProgressForm.ProgressBar1.Position:=100;
    ProgressForm.Process;
    if ISA^.Data<>[] then
    begin
      count:=0;
      for i:=1 to DataInputForm.Data.RowCount-1 do
      begin
        Application.ProcessMessages;
        Se:=[];
        for j:=1 to DataInputForm.Data.ColCount-1 do
        begin
          Application.ProcessMessages;
          if DataInputForm.Data.Cells[j,i]='1' then
            Se:=Se+[j]
          else
            if FindInverse then
              Se:=Se+[j+NEG_POS_ELEM]; // Creates a set from a line
        end;
        if ISA^.Data<=Se then count:=count+1;
      end;
      if not(HasSubSetInIL(ISA^.Data)) then // This Itemset has no subset which is in
the illegal sets
      begin
        SetsTable.RowCount:=SetsTable.RowCount+1;
        SetsTable.Cells[1,SetsTable.RowCount-1]:=IntToStr(count);
        SetsTable.Cells[0,SetsTable.RowCount-1]:=SetToStr(ISA^.Data);
      end;
    end;
    HashTableNext(ItemSets);
    ISA:=ItemSets.DataPtr;
  end;
  ProgressForm.Hide;
  AprioriForm.Enabled:=True;

  if SetsTable.RowCount>1 then
  begin
    SetsTable.FixedRows:=1;

    FindTheshold(min,max,threshold); // Finds MinSupport, MaxSupport and the Threshold
    Label2.Caption:='MinSupport is: '+IntToStr(Min);
    Label3.Caption:='MaxSupport is: '+IntToStr(Max);
    Label4.Caption:='The threshold is: '+IntToStr(Threshold);

    // Adds Itemsets which their supports is lower than the Threshold
    AprioriForm.Enabled:=False;
    ProgressForm.Show;
    ProgressForm.Label1.Caption:='Adding Itemsets to the Illegal List table...';
    HashTableHead(ItemSets);
    ISA:=ItemSets.DataPtr;
    for x:=0 to HashTableCount(ItemSets)-1 do
    begin
      if HashTableCount(ItemSets)>1 then
        ProgressForm.ProgressBar1.Position:=Trunc(100*x/(HashTableCount(ItemSets)-1))
      else
        ProgressForm.ProgressBar1.Position:=100;
      ProgressForm.Process;
      if ISA^.Data<>[] then
      begin
        count:=0;
        for i:=1 to DataInputForm.Data.RowCount-1 do // Checks each row
        begin
          Se:=[];
          for j:=1 to DataInputForm.Data.ColCount-1 do // Create the set from the row
          begin
            Application.ProcessMessages;
            if DataInputForm.Data.Cells[j,i]='1' then
              Se:=Se+[j]
            else
            if FindInverse then
              Se:=Se+[j+NEG_POS_ELEM]; // Creates a set from a line
          end;
          if ISA^.Data<=Se then count:=count+1;
        end;
        if (not(HasSubSetInIL(ISA^.Data))) and (count<Threshold) then
        begin
          AddLinkedList(IllegalList,ISA^.Data);
          ISA^.Data:=[];
        end;
      end;
      HashTableNext(ItemSets);
      ISA:=ItemSets.DataPtr;
    end;
    ProgressForm.Hide;
    AprioriForm.Enabled:=True;

    //Show Illegal list
    ReportForm.ReportMemo.Lines.Add('Current Illegal List:');
    ISA:=IllegalList.Head;
    for x:=1 to IllegalList.Count do
    begin
      if NumOfElements(ISA^.Data)<=n then
      begin
        Application.ProcessMessages;
        AprioriILGrid.RowCount:=AprioriILGrid.RowCount+1;
        AprioriILGrid.Cells[0,AprioriILGrid.RowCount-1]:=SetToStr(ISA^.Data);
        ReportForm.ReportMemo.Lines.Add(SetToStr(ISA^.Data));
      end;
      ISA:=ISA^.Next;
    end;
    AprioriILGrid.FixedRows:=1;
    if DataInputForm.SortIllegalListTable1.Checked then
      SortILTable;
    if DataInputForm.SortItemsetsTable1.Checked then
      SortTable;
    AutoAssociationTimer.Enabled:=True;
  end
  else
  begin
    Hide;
    FinalAssociationForm.Show;
  end;
  NextButton.Enabled:=True;
  PreviousButton.Enabled:=True;
  StopButton.Enabled:=True;
  SetsTable.Visible:=True;
  AprioriILGrid.Visible:=True;
  ReportForm.ReportMemo.Lines.Add('End time: '+TimeToStr(Time));
end;
```

The `NextButtonClick` procedure is executed whenever the user clicks the "next" button. It shows the next stage of the Apriori algorithm.

```
procedure TAprioriForm.NextButtonClick(Sender: TObject);
begin
  n:=n+1; // Increase stage counter
  FormShow(Sender);
end;
```

The `PreviousButtonClick` procedure is executed whenever the user clicks the "previous" button. It shows the previous stage of the Apriori algorithm, or shows a warning massage if this is the first stage.

```
procedure TAprioriForm.PreviousButtonClick(Sender: TObject);
begin
  if n>1 then
  begin
    n:=n-1; // Decrease stage counter
    FormShow(Sender);
  end
  else
  begin
    MessageDlg('This is the first stage, there was no other stage before this one !',
mtError     ,[mbOk], 0);
  end;
end;
```

The `StopButtonClick` procedure is executed whenever the user clicks the "stop" button. It stops the Apriori algorithm, and returns the program to the main screen.

```
procedure TAprioriForm.StopButtonClick(Sender: TObject);
begin
  Hide;
end;
```

The `SortTable` procedure sorts the Itemsets table, in every stage of the algorithm, if the sorting was enabled (from the main screen from). This is an implementation of the heap-sort algorithm.

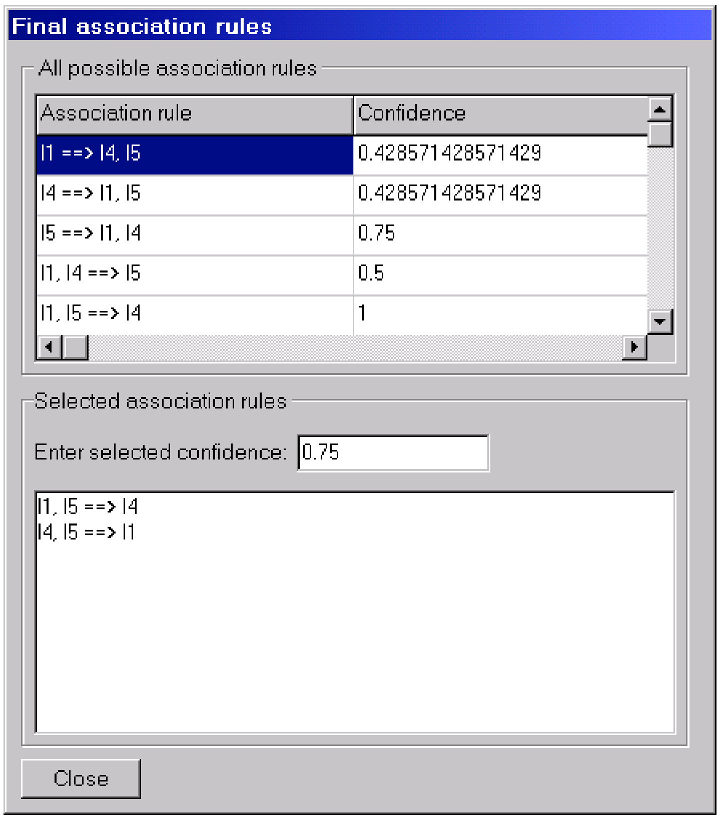
```
procedure TAprioriForm.SortTable;
var
```

```
  x    : LongWord;
  Temp : String;

  Procedure Heapify (i,Size:LongWord); { "Fixs" a branch in the heap array }
  var
    Largest : LongWord;
    Temp    : String;
  begin
    repeat
      Largest:=i;
      if (2*i<=Size) then
        if (SetsTable.Cells[0,2*i]>SetsTable.Cells[0,i]) then Largest:=2*i;
      if (2*i+1<=Size) then
        if (SetsTable.Cells[0,2*i+1]>SetsTable.Cells[0,Largest]) then Largest:=2*i+1;
      if Largest<>i then
      begin
        Temp:=SetsTable.Cells[0,i];
        SetsTable.Cells[0,i]:=SetsTable.Cells[0,Largest]; { We have to swap the two
variables }
        SetsTable.Cells[0,Largest]:=Temp;
        Temp:=SetsTable.Cells[1,i];
        SetsTable.Cells[1,i]:=SetsTable.Cells[1,Largest]; { We have to swap the two
variables }
        SetsTable.Cells[1,Largest]:=Temp;
        i:=Largest;
      end
      else
        i:=Size+1;
    until i>=Size;
  end;

begin
  AprioriForm.Enabled:=False;
  ProgressForm.Show;
  ProgressForm.Label1.Caption:='Sorting Itemsets table...';
  for x:=(SetsTable.RowCount-1 div 2) downto 1 do { "Fixs" the array to a heap form,
from the father variable to the begining of the array }
  begin
    ProgressForm.ProgressBar1.Position:=Trunc(100*((SetsTable.RowCount-1 div
2)-x)/(SetsTable.RowCount-1 div 2 + SetsTable.RowCount-2));
    ProgressForm.Process;
    Heapify (x,SetsTable.RowCount-1);
  end;
  for x:=SetsTable.RowCount-1 downto 2 do
  begin
    ProgressForm.ProgressBar1.Position:=Trunc(100*((SetsTable.RowCount-1 div 2) +
SetsTable.RowCount-2 - x)/(SetsTable.RowCount-1 div 2 + SetsTable.RowCount-2));
    ProgressForm.Process;
    Temp:=SetsTable.Cells[0,x];
    SetsTable.Cells[0,x]:=SetsTable.Cells[0,1]; { We have to swap the two variables }
    SetsTable.Cells[0,1]:=Temp;
    Temp:=SetsTable.Cells[1,x];
    SetsTable.Cells[1,x]:=SetsTable.Cells[1,1]; { We have to swap the two variables }
    SetsTable.Cells[1,1]:=Temp;
    Heapify(1,SetsTable.RowCount-1-(SetsTable.RowCount-x));
  end;


ProgressForm.ProgressBar1.Position:=Trunc(100*(SetsTable.RowCount-1)/(SetsTable.RowCou
nt-1));
    ProgressForm.Process;
  ProgressForm.Hide;
  AprioriForm.Enabled:=True;
end;
```

The `SortILTable` procedure sorts the Illegal List table, in every stage of the algorithm, if the sorting was enabled (from the main screen from). This is an implementation of the heap-sort algorithm.

```
procedure TAprioriForm.SortILTable;
var
  i,j : integer; { Variables used for array scanning }
  flg : boolean; { A flag }
  Temp: String;
begin
  AprioriForm.Enabled:=False;
  ProgressForm.Show;
  ProgressForm.Label1.Caption:='Sorting Illega List table...';
  i:=AprioriILGrid.RowCount-1; { i gets the size of the array }
  flg:=true; { If the flag is set to true, then the array is not sorted }
  while (i>1) and (flg) do { If i is bigger than 1 and the flag is set to true (the
array isn't sorted) }
  begin

ProgressForm.ProgressBar1.Position:=Trunc(100*(AprioriILGrid.RowCount-1-i)/(AprioriILG
rid.RowCount-1));
    ProgressForm.Process;
    flg:=false; { Let's assume the the array is already sorted }
    for j:=1 to i-1 do { We are scanning variables 1 to i-1 of the array }
      if AprioriILGrid.Cells[0,j]>AprioriILGrid.Cells[0,j+1] then { If variable j is
bigger than variable j+1 }
      begin
        flg:=true; { The array isn't sorted, another chack is needed }
        Temp:=AprioriILGrid.Cells[0,j];
        AprioriILGrid.Cells[0,j]:=AprioriILGrid.Cells[0,j+1]; { We have to swap the
two variables }
        AprioriILGrid.Cells[0,j+1]:=Temp;
      end;
    i:=i-1; { Variable i is "sorted", we should chack again from 1 to i-1 }
  end; { The array is sorted }
  ProgressForm.Hide;
  AprioriForm.Enabled:=True;
end;

procedure TAprioriForm.AutoAssociationTimerTimer(Sender: TObject);
begin
  AutoAssociationTimer.Enabled:=False;
  if FindAuto then
    NextButtonClick(Sender);
end;

procedure TAprioriForm.FormCreate(Sender: TObject);
begin
  StringGrid2.Cells[0,0]:='Illegal set';
  StringGrid3.Cells[0,0]:='Sets';
  StringGrid3.Cells[1,0]:='Frequency';
end;

end.
```

5.4. FinalAssociationUnit unit



Association rule	Confidence
I1 ==> I4, I5	0.428571428571429
I4 ==> I1, I5	0.428571428571429
I5 ==> I1, I4	0.75
I1, I4 ==> I5	0.5
I1, I5 ==> I4	1

Enter selected confidence: 0.75

I1, I5 ==> I4
I4, I5 ==> I1

```
unit FinalAssociationUnit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls,Sets ,DataInputUnit, Grids,Math, Mask, HashTable,LinkedList;

Type
  TFinalAssociationForm = class(TForm)
    GroupBox1: TGroupBox;
    StringGrid1: TStringGrid;
    GroupBox2: TGroupBox;
    ListBox1: TListBox;
    Label1: TLabel;
    Button1: TButton;
    Edit1: TMaskEdit;
    procedure FormShow(Sender: TObject);
    procedure Edit1Change(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure SortAssoc;
    procedure StringGrid1DblClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FinalAssociationForm: TFinalAssociationForm;

implementation

uses ProgressUnit;
```

```
var
  TempSetsArray : THashTable;

{$R *.DFM}
```

The `TempPowerSet` procedure generates all sub-sets of size *n* and stores them in a temporary hash table.

```
Procedure TempPowerSet(S:TIntSet);
var
  TempNum : LongWord;
  xx : LongWord;
  i  : byte;
  Se : array[1..MAX_SET_SIZE] of byte;
  TS : TIntSet;
  NOE : Byte;
Begin
  FreeHashTable(TempSetsArray);
  if S<>[] then // There are no subsets for an empty set
  begin
    // Save the Set values in the array;
    i:=1;
    for xx:=1 to MAX_SET_SIZE do
    begin
      Se[xx]:=0;
      if xx in S then
      begin
        Se[i]:=xx;
        Inc(i);
      end;
    end;
    NOE:=Trunc(Power(2,NumOfElements(S)))-2; // The number of sets in the Power-set,
without the empty set and the original set
    for xx:=1 to NOE do
    begin
      TS:=[];
      TempNum:=xx;
      for i:=1 to xx do
      begin
        if (TempNum mod 2)=1 then
            TS:=TS+[Se[i]];
        TempNum:=TempNum div 2;
      end;
      if TS<>[] then
        AddHashTable(TempSetsArray,TS);
    end;
  end;
end;

//------------------------------------------------------------------------------------
----
```

The `FormShow` procedure generates the association rules from the Itemsets found in the last stage of the mining operation (with either the Native or Apriori algorithms). Displays them, and let you select a desired confidence, and displays all association rules with the desired confidence and higher.

```
procedure TFinalAssociationForm.FormShow(Sender: TObject);
var
  x,i,j,y : integer;
  S : TIntSet;
  SetCount : integer;
  SubSetCount : integer;
  ISA,TISA   : TLinkedSetsPtr;
Begin
  Edit1.Text:='0';
  FreeHashTable(TempSetsArray);
  FreeHashTable(ItemSets);
  ListBox1.Clear;
  for i:=1 to DataInputForm.Data.RowCount-1 do // Scan all rows
  begin
    Application.ProcessMessages;
    S:=[];
    for j:=1 to DataInputForm.Data.ColCount-1 do
    begin
      Application.ProcessMessages;
      if DataInputForm.Data.Cells[j,i]='1' then // Creates a set
        S:=S+[j]
      Else
        if FindInverse then
          S:=S+[j+NEG_POS_ELEM];
    end;
    SerIndex:=0;
    PowerSet (S,n-1,i); // Gets all subsets length n
  end;

  StringGrid1.RowCount:=1;
  StringGrid1.Cells[0,0]:='Association rule';
  StringGrid1.Cells[1,0]:='Confidence';

  ProgressForm.Show;
  ProgressForm.Label1.Caption:='Generating association rules...';
  HashTableHead(ItemSets);
  ISA:=ItemSets.DataPtr;
  for x:=0 to HashTableCount(ItemSets)-1 do
  begin
    ProgressForm.ProgressBar1.Position:=Trunc(100*x/(HashTableCount(ItemSets)-1));
    ProgressForm.Process;
    if ISA^.Data<>[] then
    begin
      SetCount:=0;
      if not(HasSubSetInIL(ISA^.Data)) then
      begin
        for i:=1 to DataInputForm.Data.RowCount-1 do
        begin
          S:=[];
          for j:=1 to DataInputForm.Data.ColCount-1 do
          begin
            Application.ProcessMessages;
            if DataInputForm.Data.Cells[j,i]='1' then
              S:=S+[j]
            Else
              if FindInverse then
                S:=S+[j+NEG_POS_ELEM]; // Creates a set from a line
          end;
          if s>=ISA^.Data then
            SetCount:=SetCount+1;
        end;
        TempPowerSet(ISA^.Data);
        HashTableHead(TempSetsArray);
        TISA:=TempSetsArray.DataPtr;
        for j:=0 to HashTableCount(TempSetsArray)-1 do // here is the set
        begin
          SubSetCount:=0;
          for i:=1 to DataInputForm.Data.RowCount-1 do
          begin
            S:=[];
            for y:=1 to DataInputForm.Data.ColCount-1 do
              if DataInputForm.Data.Cells[y,i]='1' then // Creates a set
                S:=S+[y]
              Else
                if FindInverse then
                  S:=S+[y+NEG_POS_ELEM];
            if s>=TISA^.Data then
              SubSetCount:=SubSetCount+1;
          end;
          StringGrid1.RowCount:=StringGrid1.RowCount+1;
          StringGrid1.Cells[0,StringGrid1.RowCount-1]:=SetToStr(TISA^.Data)+' ==>
'+SetToStr(ISA^.Data-TISA^.Data);
        StringGrid1.Cells[1,StringGrid1.RowCount-1]:=FloatToStr(SetCount/SubSetCount);
          ListBox1.Items.Add(SetToStr(TISA^.Data)+' ==>
'+SetToStr(ISA^.Data-TISA^.Data));
          HashTableNext(TempSetsArray);
          TISA:=TempSetsArray.DataPtr;
        end;
      end;
    end;
    HashTableNext(ItemSets);
    ISA:=ItemSets.DataPtr;
  end;
  ProgressForm.Hide;

  if StringGrid1.RowCount=1 then
  begin
    Beep;
    MessageDlg('Association or Inverse Accosiation Rules cannot be found !',
mtInformation     ,[mbOk], 0);
    StringGrid1.RowCount:=2;
  end;
  StringGrid1.FixedRows:=1;
  if DataInputForm.SortAssociationsTable1.Checked then
    SortAssoc;
end;

procedure TFinalAssociationForm.Edit1Change(Sender: TObject);
var
  x : integer;
  S : String;
begin
  ListBox1.Clear;
  S:=Edit1.Text;
  While Pos(' ',S)<>0 do
    S[Pos(' ',S)]:='0';
  if StrToFloat(S)>0 then
    if S[1]='.' then
      S:='0'+S;
  Edit1.Text:=S;
  if StrToFloat(S)>1 then
  begin
    Edit1.Text:='1';
    S:='1';
  end;
  try
    for x:=1 to StringGrid1.RowCount-1 do
    begin
      if StrToFloat(S)<=StrToFloat(StringGrid1.Cells[1,x]) then
        ListBox1.Items.Add(StringGrid1.Cells[0,x]);
    end;
  except
  end;
end;
```

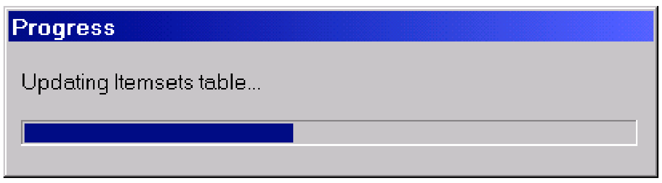
Pressing the “Close” button, return the user to the main screen. This is done by the `Button1Click` procedure.

```
procedure TFinalAssociationForm.Button1Click(Sender: TObject);
begin
  Hide;
end;
```

The `SortAssoc` procedure sorts the association rules table, if the sorting was enabled (from the main screen from). This is an implementation of the bubble-sort algorithm.

```
procedure TFinalAssociationForm.SortAssoc;
var
  i,j : integer; { Variables used for array scanning }
  flg : boolean; { A flag }
  Temp: String;
begin
  ProgressForm.Show;
  ProgressForm.Label1.Caption:='Sorting association rules table...';
  i:=StringGrid1.RowCount-1; { i gets the size of the array }
  flg:=true; { If the flag is set to true, then the array is not sorted }
  while (i>1) and (flg) do { If i is bigger than 1 and the flag is set to true (the
array isn't sorted) }
  begin

ProgressForm.ProgressBar1.Position:=Trunc(100*(StringGrid1.RowCount-1-i)/(StringGrid1.
RowCount-1));
    ProgressForm.Process;
    flg:=false; { Let's assume the the array is already sorted }
    for j:=1 to i-1 do { We are scanning variables 1 to i-1 of the array }
      if StringGrid1.Cells[1,j]>StringGrid1.Cells[1,j+1] then { If variable j is
bigger than variable j+1 }
      begin
        flg:=true; { The array isn't sorted, another chack is needed }
        Temp:=StringGrid1.Cells[0,j];
        StringGrid1.Cells[0,j]:=StringGrid1.Cells[0,j+1]; { We have to swap the two
variables }
        StringGrid1.Cells[0,j+1]:=Temp;
        Temp:=StringGrid1.Cells[1,j];
        StringGrid1.Cells[1,j]:=StringGrid1.Cells[1,j+1]; { We have to swap the two
variables }
        StringGrid1.Cells[1,j+1]:=Temp;
      end;
    i:=i-1; { Variable i is "sorted", we should chack again from 1 to i-1 }
  end; { The array is sorted }
  ProgressForm.Hide;
end;

procedure TFinalAssociationForm.StringGrid1DblClick(Sender: TObject);
var
  x : Integer;
  TempW : Integer;
begin
  TempW:=0;
  for x:=0 to StringGrid1.RowCount-1 do
  begin
    if StringGrid1.Canvas.TextWidth(StringGrid1.Cells[0,x])>TempW then
      TempW:=StringGrid1.Canvas.TextWidth(StringGrid1.Cells[0,x]);
  end;
  StringGrid1.ColWidths[0]:=TempW+5;
end;

procedure TFinalAssociationForm.FormCreate(Sender: TObject);
begin
  InitHashTable(TempSetsArray);
end;

procedure TFinalAssociationForm.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
  FreeHashTable(TempSetsArray);
end;

end.
```

5.5. ProgressUnit Unit



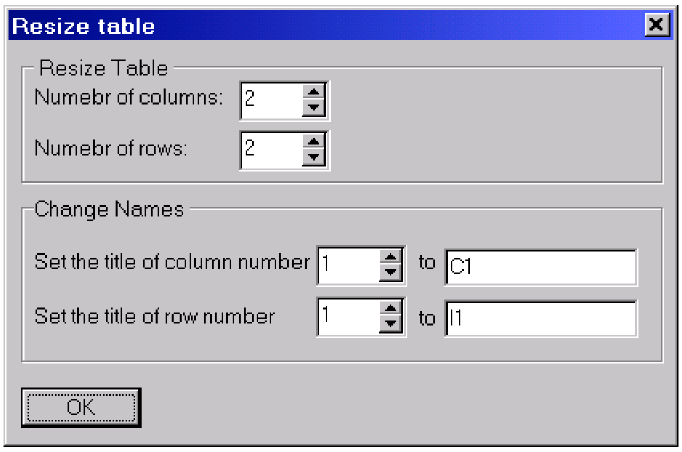
```
unit ProgressUnit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ComCtrls, StdCtrls;
```

```
type
  TProgressForm = class(TForm)
    ProgressBar1: TProgressBar;
    Label1: TLabel;
    procedure Process;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  ProgressForm: TProgressForm;

implementation

{$R *.DFM}
```

The `Prcess` procedure is used whenever the program is using a loop, in order to make the program look working and not stack.

```
procedure TProgressForm.Process;
begin
  Application.ProcessMessages;
end;

end.
```

5.6. ResizeTableUnit Unit



```
unit ResizeTableUnit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Spin;

type
  TResizeTableForm = class(TForm)
    Button1: TButton;
    GroupBox1: TGroupBox;
    Label1: TLabel;
    SpinEdit1: TSpinEdit;
    Label2: TLabel;
    SpinEdit2: TSpinEdit;
    GroupBox2: TGroupBox;
    Label3: TLabel;
    SpinEdit3: TSpinEdit;
    Edit4: TEdit;
    Edit1: TEdit;
    SpinEdit4: TSpinEdit;
    Label5: TLabel;
    Label4: TLabel;
    Label6: TLabel;
    procedure FormShow(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure SpinEdit3Change(Sender: TObject);
    procedure SpinEdit4Change(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  ResizeTableForm: TResizeTableForm;

implementation

uses DataInputUnit;

{$R *.DFM}
```

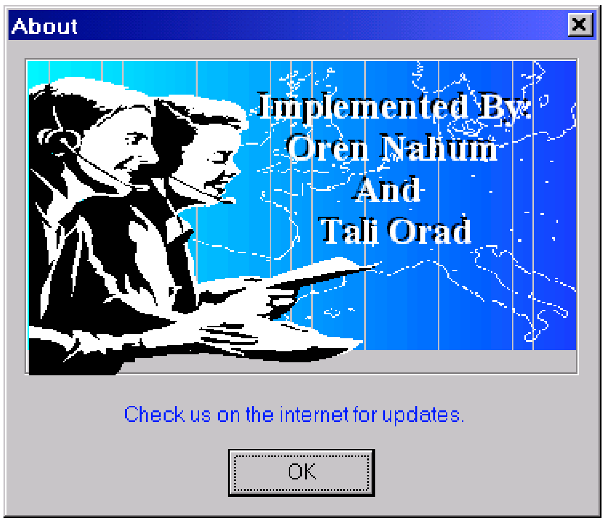
The `FormShow` procedure is used to update the information that the resize form dialogue shows.

```
procedure TResizeTableForm.FormShow(Sender: TObject);
begin
  SpinEdit1.Value:=DataInputForm.Data.ColCount-1;
  SpinEdit2.Value:=DataInputForm.Data.RowCount-1;
  Edit4.Text:=DataInputForm.Data.Cells[SpinEdit3.Value,0];
  Edit1.Text:=DataInputForm.Data.Cells[0,SpinEdit4.Value];
end;
```

The `Button1Click` procedure is used to update the data table with the information entered in the resize form dialogue.

```
procedure TResizeTableForm.Button1Click(Sender: TObject);
var
  x : Integer;
begin
  DataInputForm.Enabled:=True;
  DataInputForm.Data.ColCount:=SpinEdit1.Value+1;
  DataInputForm.Data.RowCount:=SpinEdit2.Value+1;
  DataInputForm.Data.Cells[SpinEdit3.Value,0]:=Edit4.Text;
  DataInputForm.Data.Cells[0,SpinEdit4.Value]:=Edit1.Text;
  DataInputForm.StatusBar1.Panels[1].Text:='Table size:
'+IntToStr(DataInputForm.Data.ColCount-1)+'x'+IntToStr(DataInputForm.Data.RowCount-1);
  for x:=1 to DataInputForm.Data.ColCount-1 do
  begin
    if DataInputForm.Data.Cells[x,0]='' then
      DataInputForm.Data.Cells[x,0]:='C'+IntToStr(x);
  end;
  for x:=1 to DataInputForm.Data.RowCount-1 do
  begin
    if DataInputForm.Data.Cells[0,x]='' then
      DataInputForm.Data.Cells[0,x]:='I'+IntToStr(x);
  end;
  Hide;
end;

procedure TResizeTableForm.SpinEdit3Change(Sender: TObject);
begin
  Edit4.Text:=DataInputForm.Data.Cells[SpinEdit3.Value,0];
end;

procedure TResizeTableForm.SpinEdit4Change(Sender: TObject);
begin
  Edit1.Text:=DataInputForm.Data.Cells[0,SpinEdit4.Value];
end;

end.
```

5.7. AboutUnit Unit



```
unit AboutUnit;

interface

uses Windows, SysUtils, Classes, Graphics, Forms, Controls, StdCtrls,
  Buttons, ExtCtrls, ShellAPI;

type
  TAboutBoxForm = class(TForm)
    Panel1: TPanel;
    ProgramIcon: TImage;
    OKButton: TButton;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    procedure OKButtonClick(Sender: TObject);
    procedure Label1MouseMove(Sender: TObject; Shift: TShiftState; X,
      Y: Integer);
    procedure FormMouseMove(Sender: TObject; Shift: TShiftState; X,
      Y: Integer);
    procedure Label1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  AboutBoxForm: TAboutBoxForm;

implementation

uses DataInputUnit;

{$R *.DFM}
```

The `OKButtonClick` procedure closes the about box, and brings the user back to the main screen.

```
procedure TAboutBoxForm.OKButtonClick(Sender: TObject);
begin
  Hide;
  DataInputForm.Enabled:=True;
end;

procedure TAboutBoxForm.Label1MouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
  Label1.Font.Color:=clRed;
end;

procedure TAboutBoxForm.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
    Label1.Font.Color:=clBlue;
end;
```

The `Label1Click` procedure opens your default web browser, showing you the association rules miner homepage.

```
procedure TAboutBoxForm.Label1Click(Sender: TObject);
begin
 ShellExecute(GetDesktopWindow(), 'open',
PChar('http://users.surfree.net.il/orennahum/assoc.htm'), nil, nil, SW_SHOWNORMAL);
end;

end.
```

5.8. Sets Unit

```
unit Sets;

interface

uses Sysutils,Math;

Const
  MAX_SET_SIZE = 30*2;
  NEG_POS_ELEM = MAX_SET_SIZE div 2;

Type
  TIntSet = set of 1..MAX_SET_SIZE; // A set can have at most 256 elements

var
  SerIndex    : LongWord;
  SMax        : LongWord;

procedure PowerSet(S:TIntSet; L,LineNum:Integer);
function  HasSubSetInIL(S:TIntSet):boolean;
function  SetToStr(S:TIntSet):String;
function  NumOfElements(S:TIntSet):Integer;
function  NumOfSubSets(SerSize:Integer):LongWord;
function  Factorial(n:integer):Extended;

implementation

uses DataInputUnit,ProgressUnit,Dialogs,LinkedList,HashTable;
```

The `Factorial` function returns the factorial of *n*.

```
function Factorial(n:integer):Extended;
var
  x : integer;
  r : Extended;
begin
  r:=1;
  Factorial:=1;
  if n>0 then
  begin
    for x:=1 to n do
      r:=r*x;
    Factorial:=r;
  end;
```

```
end;
```

The `NumOfSubSets` function returns the maximum number of possible sub-set of a given size, *n*.

```
function NumOfSubSets(SerSize:Integer):LongWord;
var
  Nu : Integer;
begin
  if FindInverse then
    Nu:=2*(DataInputForm.Data.ColCount-1)
  else
    Nu:=(DataInputForm.Data.ColCount-1);

  NumOfSubSets:=Trunc(Factorial(Nu) / (Factorial(SerSize)*Factorial(Nu-SerSize)));
end;

// ------------------ General Sets functions and procedures -------------------
```

For a given set, the `SetToStr` function returns a string that contains the elements of the set. The returned string can be printed, for example.

```
function SetToStr(S:TIntSet):String; // Convers a Set to String
var
  x  : Integer;
  St : String;
begin
  St:='';
  for x:=1 to NEG_POS_ELEM do
  begin
    if (x in S) then
    begin
      if St='' then // is this the first element in the set
        St:=DataInputForm.Data.Cells[x,0]
      else
        St:=St+', '+DataInputForm.Data.Cells[x,0];
    end;
    if ((x+NEG_POS_ELEM) in S) then
    begin
      if St='' then // is this the first element in the set
        St:='-'+DataInputForm.Data.Cells[x,0]
      else
        St:=St+', -'+DataInputForm.Data.Cells[x,0];
    end;
  end;
  SetToStr:=St;
end;
```

The `NumOfElements` function, gets a set, and returns the number of elements in the set.

```
function NumOfElements(S:TIntSet):Integer; // Returns the number of elements in the
set
var
  x : integer;
  count : integer;
begin
  Count:=0;
  for x:=1 to MAX_SET_SIZE do
    if x in S then Count:=Count+1;
  NumOfElements:=Count;
end;
```

The `PowerSet` procedure gets a set, and returns all its sub-sets of a given size.

```
Procedure PowerSet(S:TIntSet;L,LineNum:Integer);
var
  xx : LongWord;
  i  : byte;
  Se : array[1..MAX_SET_SIZE] of byte;
  TS : TIntSet;
  N  : Integer;
  x : array[1..MAX_SET_SIZE] of byte;
  r : array[1..MAX_SET_SIZE] of byte;
  NOE : Byte;
  NOS : LongWord;
  NOSC : LongWord;
begin
  if (S<>[]) and (NumOfElements(S)>=L) then // There are no subsets for an empty set
  begin
    // Save the Set values in the array;
    i:=1;
    N:=NumOfElements(S);
    for xx:=1 to MAX_SET_SIZE do
    begin
      r[xx]:=0;
      x[xx]:=0;
      Se[xx]:=0;
      if xx in S then
      begin
        Se[i]:=xx;
        Inc(i);
      end;
    end;
    // Now finding all subsets with size L
    NOE:=NumOfElements(S);
    if NOE>=L then
      NOS:=Trunc(Factorial(NOE)/(Factorial(L)*Factorial(NOE-L)))
    else
      NOS:=1;
    NOSC:=0;
    ProgressForm.Show;
    ProgressForm.Label1.Caption:='Creating subsets for line number
'+IntToStr(LineNum)+' out of '+IntToStr(DataInputForm.Data.RowCount-1)+'...';
    if L>29 then
      x[30]:=1
    else
      x[30]:=0;
    repeat
      if x[30]<>0 then r[x[30]]:=1;
      if L>28 then
        x[29]:=x[30]+1
      else
        x[29]:=0;
      repeat
        if x[29]<>0 then r[x[29]]:=1;
        if L>27 then
          x[28]:=x[29]+1
        else
          x[28]:=0;
        repeat
          if x[28]<>0 then r[x[28]]:=1;
          if L>26 then
            x[27]:=x[28]+1
          else
            x[27]:=0;
          repeat
            if x[27]<>0 then r[x[27]]:=1;
            if L>25 then
              x[26]:=x[27]+1
            else
              x[26]:=0;
            repeat
              if x[26]<>0 then r[x[26]]:=1;
              if L>24 then
                x[25]:=x[26]+1
              else
                x[25]:=0;
              repeat
                if x[25]<>0 then r[x[25]]:=1;
                if L>23 then
                  x[24]:=x[25]+1
                else
                  x[24]:=0;
```

```
repeat
  if x[24]<>0 then r[x[24]]:=1;
  if L>22 then
    x[23]:=x[24]+1
  else
    x[23]:=0;
  repeat
    if x[23]<>0 then r[x[23]]:=1;
    if L>21 then
      x[22]:=x[23]+1
    else
      x[22]:=0;
    repeat
      if x[22]<>0 then r[x[22]]:=1;
      if L>20 then
        x[21]:=x[22]+1
      else
        x[21]:=0;
      repeat
        if x[21]<>0 then r[x[21]]:=1;
        if L>19 then
          x[20]:=x[21]+1
        else
          x[20]:=0;
        repeat
          if x[20]<>0 then r[x[20]]:=1;
          if L>18 then
            x[19]:=x[20]+1
          else
            x[19]:=0;
          repeat
            if x[19]<>0 then r[x[19]]:=1;
            if L>17 then
              x[18]:=x[19]+1
            else
              x[18]:=0;
            repeat
              if x[18]<>0 then r[x[18]]:=1;
              if L>16 then
                x[17]:=x[18]+1
              else
                x[17]:=0;
              repeat
                if x[17]<>0 then r[x[17]]:=1;
                if L>15 then
                  x[16]:=x[17]+1
                else
                  x[16]:=0;
                repeat
                  if x[16]<>0 then r[x[16]]:=1;
                  if L>14 then
                    x[15]:=x[16]+1
                  else
                    x[15]:=0;
                  repeat
                    if x[15]<>0 then r[x[15]]:=1;
                    if L>13 then
                      x[14]:=x[15]+1
                    else
                      x[14]:=0;
                    repeat
                      if x[14]<>0 then r[x[14]]:=1;
                      if L>12 then
                        x[13]:=x[14]+1
                      else
                        x[13]:=0;
                      repeat
                        if x[13]<>0 then r[x[13]]:=1;
                        if L>11 then
                          x[12]:=x[13]+1
                        else
                          x[12]:=0;
                        repeat
                          if x[12]<>0 then r[x[12]]:=1;
```

```
                                          if L>10 then
                                            x[11]:=x[12]+1
                                          else
                                            x[11]:=0;
                                          repeat
                                            if x[11]<>0 then r[x[11]]:=1;
                                            if L>9 then
                                              x[10]:=x[11]+1
                                            else
                                              x[10]:=0;
                                            repeat
                                              if x[10]<>0 then r[x[10]]:=1;
                                              if L>8 then
                                                x[9]:=x[10]+1
                                              else
                                                x[9]:=0;
                                              repeat
                                                if x[9]<>0 then r[x[9]]:=1;
                                                if L>7 then
                                                  x[8]:=x[9]+1
                                                else
                                                  x[8]:=0;
                                                repeat
                                                  if x[8]<>0 then r[x[8]]:=1;
                                                  if L>6 then
                                                    x[7]:=x[8]+1
                                                  else
                                                    x[7]:=0;
                                                  repeat
                                                    if x[7]<>0 then r[x[7]]:=1;
                                                    if L>5 then
                                                      x[6]:=x[7]+1
                                                    else
                                                      x[6]:=0;
                                                    repeat
                                                      if x[6]<>0 then r[x[6]]:=1;
                                                      if L>4 then
                                                        x[5]:=x[6]+1
                                                      else
                                                        x[5]:=0;
                                                      repeat
                                                        if x[5]<>0 then r[x[5]]:=1;
                                                        if L>3 then
                                                          x[4]:=x[5]+1
                                                        else
                                                          x[4]:=0;
                                                        repeat
                                                          if x[4]<>0 then r[x[4]]:=1;
                                                          if L>2 then
                                                            x[3]:=x[4]+1
                                                          else
                                                            x[3]:=0;
                                                          repeat
                                                            if x[3]<>0 then
r[x[3]]:=1;

                                                            if L>1 then
                                                              x[2]:=x[3]+1
                                                            else
                                                              x[2]:=0;
                                                            repeat
                                                              x[1]:=x[2]+1;
                                                              if x[2]<>0 then
r[x[2]]:=1;

                                                              repeat
                                                                Inc(NOSC);

ProgressForm.ProgressBar1.Position:=Trunc(NOSC/NOS*100);
                                                                ProgressForm.Process;
                                                                r[x[1]]:=1;
                                                                TS:=[];
                                                                for i:=1 to N do
                                                                begin
                                                                  if r[i]=1 then
                                                                    TS:=TS+[Se[i]];
                                                                end;
                                                                if NumOfElements(TS)=L
then

AddHashTable(ItemSets,TS);
                                                                r[x[1]]:=0;
                                                                Inc(x[1]);
                                                              until x[1]>NOE;
                                                              r[x[2]]:=0;
                                                              Inc(x[2]);
                                                            until (x[2]>NOE-1) or
(L<2);

                                                            r[x[3]]:=0;
                                                            Inc(x[3]);
                                                          until (x[3]>NOE-2) or (L<3);
                                                          r[x[4]]:=0;
                                                          Inc(x[4]);
                                                        until (x[4]>NOE-3) or (L<4);
                                                        r[x[5]]:=0;
                                                        Inc(x[5]);
                                                      until (x[5]>NOE-4) or (L<5);
                                                      r[x[6]]:=0;
                                                      Inc(x[6]);
                                                    until (x[6]>NOE-5) or (L<6);
                                                    r[x[7]]:=0;
                                                    Inc(x[7]);
                                                  until (x[7]>NOE-6) or (L<7);
                                                  r[x[8]]:=0;
                                                  Inc(x[8]);
                                                until (x[8]>NOE-7) or (L<8);
                                                r[x[9]]:=0;
                                                Inc(x[9]);
                                              until (x[9]>NOE-8) or (L<9);
                                              r[x[10]]:=0;
                                              Inc(x[10]);
                                            until (x[10]>NOE-9) or (L<10);
                                            r[x[11]]:=0;
                                            Inc(x[11]);
                                          until (x[11]>NOE-10) or (L<11);
                                          r[x[12]]:=0;
                                          Inc(x[12]);
                                        until (x[12]>NOE-11) or (L<12);
                                        r[x[13]]:=0;
                                        Inc(x[13]);
                                      until (x[13]>NOE-12) or (L<13);
                                      r[x[14]]:=0;
                                      Inc(x[14]);
                                    until (x[14]>NOE-13) or (L<14);
                                    r[x[15]]:=0;
                                    Inc(x[15]);
                                  until (x[15]>NOE-14) or (L<15);
                                  r[x[16]]:=0;
                                  Inc(x[16]);
                                until (x[16]>NOE-15) or (L<16);
                                r[x[17]]:=0;
                                Inc(x[17]);
                              until (x[17]>NOE-16) or (L<17);
                              r[x[18]]:=0;
                              Inc(x[18]);
                            until (x[18]>NOE-17) or (L<18);
                            r[x[19]]:=0;
                            Inc(x[19]);
                          until (x[19]>NOE-18) or (L<19);
                          r[x[20]]:=0;
                          Inc(x[20]);
                        until (x[20]>NOE-19) or (L<20);
                        r[x[21]]:=0;
                        Inc(x[21]);
                      until (x[21]>NOE-20) or (L<21);
                      r[x[22]]:=0;
                      Inc(x[22]);
                    until (x[22]>NOE-21) or (L<22);
                    r[x[23]]:=0;
                    Inc(x[23]);
```

```
                  until (x[23]>NOE-22) or (L<23);
                  r[x[24]]:=0;
                  Inc(x[24]);
                until (x[24]>NOE-23) or (L<24);
                r[x[25]]:=0;
                Inc(x[25]);
              until (x[25]>NOE-24) or (L<25);
              r[x[26]]:=0;
              Inc(x[26]);
            until (x[26]>NOE-25) or (L<26);
            r[x[27]]:=0;
            Inc(x[27]);
          until (x[27]>NOE-26) or (L<27);
          r[x[28]]:=0;
          Inc(x[28]);
        until (x[28]>NOE-27) or (L<28);
        r[x[29]]:=0;
        Inc(x[29]);
      until (x[29]>NOE-28) or (L<29);
      r[x[30]]:=0;
      Inc(x[30]);
    until (x[30]>NOE-29) or (L<30);
  end;
  ProgressForm.Hide;
end;


// ------------------ Illegal List functions and procedures -------------------
```

The `HasSubSetInIL` function gets a set, and checks whether the Illegal List contains any sets that are sub-sets to the checked set.

```
function HasSubSetInIL(S:TIntSet):boolean;
var
  Temp : TLinkedSetsPtr;
begin
  HasSubSetInIL:=False;
  Temp:=IllegalList.Head;
  while Temp<>Nil do
  begin
    if (Temp^.Data<=S) and (NumOfElements(S)>=NumOfElements(Temp^.Data)) then // has a
sub-set
    begin
      HasSubSetInIL:=true;
      exit;
    end;
    Temp:=Temp^.Next;
  end;
end;

end.
```

5.9. LinkedList Unit

```
unit LinkedList;

interface

uses Sets;

Type
  TLinkedSetsPtr = ^TLinkedSets;
  TLinkedSets = Record
    Data : TIntSet;
    Next : TLinkedSetsPtr;
  end;
  TLinkedSet = Record
    Head  : TLinkedSetsPtr;
    Count : LongWord;
  end;

procedure InitLinkedList(var LS:TLinkedSet);
procedure FreeLinkedList (var LS:TLinkedSet);
procedure AddLinkedList (var LS:TLinkedSet;S:TIntSet);
procedure CleanLinkedList (var LS:TLinkedSet);

implementation

uses ProgressUnit;
```

The `InitLinkedList` procedure initializes the linked list structure.

```
procedure InitLinkedList(var LS:TLinkedSet);
begin
  LS.Head:=Nil;
  LS.Count:=0;
end;
```

The `FreeLinkedList` procedure remover all the elements of the linked list from the linked list, and frees all dynamically allocated memory used by the linked list elements.

```
procedure FreeLinkedList (var LS:TLinkedSet);
var
  Temp : TLinkedSetsPtr;
begin
  While LS.Head<>Nil do
  begin
    Temp:=LS.Head^.Next;
    Dispose(LS.Head);
    LS.Head:=Temp;
    ProgressForm.Process;
  end;
  LS.Head:=Nil;
  LS.Count:=0;
end;
```

The `CleanLinkedList` procedure removes all elements of the linked list, which have an empty set as their data.

```
procedure CleanLinkedList (var LS:TLinkedSet);
var
  Temp : TLinkedSetsPtr;
  Prev : TLinkedSetsPtr;
begin
  While (LS.Head<>Nil) and (LS.Head^.Data=[]) do
  begin
    Temp:=LS.Head^.Next;
    Dispose(LS.Head);
    LS.Head:=Temp;
    Dec(LS.Count);
    ProgressForm.Process;
  end;
  if LS.Head<>Nil then
  begin
    Prev:=LS.Head;
    Temp:=Prev^.Next;
    While Temp<>Nil do
    begin
      if Temp.Data=[] then
      begin
        Prev^.Next:=Temp^.Next;
        Dispose(Temp);
        Dec(LS.Count);
        Temp:=Prev;
      end;
      Prev:=Temp;
      Temp:=Temp^.Next;
    end;
  end;
end;
```

The `AddLinkedList` procedure adds an element to the linked list, if this element is not already in the list.

```
procedure AddLinkedList (var LS:TLinkedSet;S:TIntSet);
var
  Temp  : TLinkedSetsPtr;
  Found : Boolean;
begin
  if LS.Head=Nil then
  begin
    New(LS.Head);
    LS.Head^.Data:=S;
    LS.Head^.Next:=Nil;
    Inc(LS.Count);
  end
  else
  begin
    Temp:=LS.Head;
    Found:=False;
    while (Not(Found)) and (Temp^.Next<>Nil) do
    begin
      if Temp^.Data=S then
        Found:=True;
      Temp:=Temp^.Next;
      ProgressForm.Process;
    end;
    if Not(Found) and (Temp^.Data<>S) Then
    begin
      New(Temp^.Next);
      Temp^.Next^.Data:=S;
      Temp^.Next^.Next:=Nil;
      Inc(LS.Count);
    end;
  end;
end;

end.
```

5.10. HashTable Unit

```
unit HashTable;

interface

uses Sets,LinkedList;

Const
  HASH_ARRAY_SIZE = 500;

Type
  THashTable = Record
    Data     : Array[0..HASH_ARRAY_SIZE-1] of TLinkedSet;
    DataPtr  : TLinkedSetsPtr;
    DataLoc  : Integer;
  end;

function HashFunction(S:TIntSet):Integer;
procedure InitHashTable(var HT:THashTable);
procedure FreeHashTable(var HT:THashTable);
function InHashTable(HT:THashTable;S:TIntSet):Boolean;
procedure AddHashTable(var HT:THashTable;S:TIntSet);
Procedure HashTableNext(var HT:THashTable);
Procedure HashTableHead(var HT:THashTable);
procedure CleanHashTable(var HT:THashTable);
function HashTableCount(HT:THashTable):LongWord;

implementation

uses ProgressUnit,Windows;
```

The `HashFunction` function maps every set to an entry in the hash table.

```
function HashFunction(S:TIntSet):Integer;
var
  x   : Integer;
  r,v : Int64;
begin
  r:=1;
  v:=0;
  for x:=1 to MAX_SET_SIZE do
  begin
    r:=r*2;
    if x in S then
      v:=v+r;
  end;
  Result:=v mod HASH_ARRAY_SIZE;
end;
```

The hash table is an array of linked lists. The `InitHashTable` procedure initializes every linked list of the hash table.

```
procedure InitHashTable(var HT:THashTable);
var
  x : Integer;
begin
  HT.DataPtr:=Nil;
  HT.DataLoc:=0;
  for x:=0 to HASH_ARRAY_SIZE-1 do
    InitLinkedList(HT.Data[x]);
end;
```

The hash table is an array of linked lists. The Free`HashTable` procedure frees every linked list of the hash table.

```
procedure FreeHashTable(var HT:THashTable);
var
  x : Integer;
begin
  HT.DataPtr:=Nil;
  HT.DataLoc:=0;
  for x:=0 to HASH_ARRAY_SIZE-1 do
    FreeLinkedList(HT.Data[x]);
end;
```

The `InHashTable` function gets a set and checks whether this set is already in the hash table.

```
function InHashTable(HT:THashTable;S:TIntSet):Boolean;
var
  Temp:TLinkedSetsPtr;
begin
  if HT.Data[HashFunction(S)].Head=Nil then
  begin
     Result:=False;
     exit;
  end
  else
  Begin
    Temp:=HT.Data[HashFunction(S)].Head;
    While (Temp^.Next=Nil) and (Temp^.Data<>S) do
    begin
      if Temp^.Data=S then
      begin
        Result:=True;
        break;
      end;
      Temp:=Temp^.Next;
      ProgressForm.Process;
    end;
    Result:=False;
  end;
end;
```

The `AddHashTable` procedure gets a set, and adds it to the linked list. It maps the set to an entry of the hash table using the hash function, and then adds it to the linked list associated to the mapped entry.

```
procedure AddHashTable(var HT:THashTable;S:TIntSet);
begin
  AddLinkedList(HT.Data[HashFunction(S)],S);
end;
```

The `HashTableHead` procedure sets the hash table’s element pointer to the first element of the hash table.

```
Procedure HashTableHead(var HT:THashTable);
var
  x : Integer;
begin
  HT.DataPtr:=Nil;
  HT.DataLoc:=0;
  for x:=0 to HASH_ARRAY_SIZE do
    if HT.Data[x].Head<>Nil then
    begin
      HT.DataLoc:=x;
      HT.DataPtr:=HT.Data[x].Head;
      break;
    end;
end;
```

The `HashTableNext` procedure sets the hash table’s element pointer to the next element of the hash table.

```
procedure HashTableNext(var HT:THashTable);
var
  x : Integer;
begin
  if HT.DataPtr^.Next<>Nil then
  begin
    HT.DataPtr:=HT.DataPtr^.Next;
  end
  else
  begin
    x:=HT.DataLoc+1;
    HT.DataPtr:=Nil;
    HT.DataLoc:=0;
    While x<HASH_ARRAY_SIZE do
    begin
      if HT.Data[x].Head<>Nil then
      begin
        HT.DataLoc:=x;
        HT.DataPtr:=HT.Data[x].Head;
        break;
      end;
      x:=x+1;
    end;
  end;
end;
```

The `HashTableCount` function returns the number of elements in the hash table.

```
function HashTableCount(HT:THashTable):LongWord;
var
  x : Integer;
  s : LongWord;
begin
  s:=0;
  for x:=0 to HASH_ARRAY_SIZE-1 do
  begin
    s:=s+HT.Data[x].Count;
  end;
  HashTableCount:=s;
end;
```

The `CleanHashTable` procedure cleans every linked list associated with the hash table.

```
procedure CleanHashTable(var HT:THashTable);
var
  x : Integer;
begin
  for x:=0 to HASH_ARRAY_SIZE-1 do
  begin
    if x<HASH_ARRAY_SIZE then
      CleanLinkedList(HT.Data[x]);
  end;
end;

end.
```

6. Summary

Knowledge discovery is the most desirable-end product in computing. Finding new phenomena or enhancing our knowledge about them has a greater long-range value than optimizing production processes or inventories, and as second only to tasks that help preserve our world and our environment. It is not surprising that it is also one of the most difficult computing challenges to do well.

Using data mining techniques such as classification, regression, clustering, summarization and so on, is a step in the process of knowledge discovery. Other data mining methods are decision trees and rules, example based methods, probabilistic graphical dependency methods and relational learning models are used to extract knowledge from data.

One of the many end products of the data mining techniques and knowledge discovery are association rules.

Association rules are statements of the form “98% of the customers that purchase tires and automobile accessories also get automotive services.” Association rules are a simple and natural class of database regularities, useful in various analysis and prediction tasks. We have considered the problem of finding all association rules satisfying user-specified support and confidence constraints that hold a given database.

We introduced two algorithms for mining association rules, and a simple modification to those algorithms that allowed us to mine inverse association rules.

An important part of mining association rules is finding large itemsets. We gave two examples, one for finding association rules, and one for finding inverse association rules, in which we use a “Native” algorithm for generating the itemsets. A better way, which generates the itemsets much faster, was presented in two algorithms, AIS and Apriori. When the Native algorithm and the Apriori algorithm were implemented in the “Association Rules Miner” program.

6.1. Future Work

While working on this paper, we came across an interesting phenomenon. While testing the “association rules miner” program, we saw that the Apriori algorithm, which in terms of efficiency works much faster when mining association rules, compared to the Native algorithms, showed very poor results when mining inverse association rules. A first look suggested that the difference between mining association rules and mining inverse association rules is in the “density” of the data table. The data table of inverse association rules is much more dense then the regular association rules. When trying to apply the Apriori algorithm on a denser data table, we saw that our first impression was wrong. It seems now, that the threshold function is the main reason for this phenomenon. When mining association rules, using a dense data table simply results with high values of threshold, which rules out many Itemsets faster. Such behavior cannot be reached when mining inverse association rules. A future study of the threshold function might result with a new threshold function, which can increase the efficiency of the Apriori algorithm for the inverse association rules mining process.

Other future work might focus on mining association rules when the known data is some items that the customer bought, some items that he didn't buy, and the other is unknown. This is different from the regular association rules and inverse association rules in terms of the known data. While mining association rules we are using only know data, and while mining inverse association rules, unknown data is automatically treated as items that the customer didn't buy. A future work might consider a more reliable data analysis.

Currently our mining algorithms are looking for association rules of the form $X \Rightarrow Y$, when X an Y are some items. We are not interested in the quantities of the items in the rules. Another way of defining association rules might be statements of the form "98% of the customers that purchase one loaf of bread also get two bottles of milk." Such association rules might be the result of new mining algorithms.

7. Acknowledgments

We would like to thank Prof. M. Schneider for his support throughout this paper. We would also like to thank Dr. V. Levit for his help with the sets algorithms, and his help with the interpretation of the results.

Special thanks to Noële Nelson, who despite her lake of time, read this paper and gave her comments.

8. References

[1] A. Shragai and M. Schneider, *Discovering Inverse Associations in Databases*, Tel-Aviv University, 2000.

[2] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen and A. I. Verkamo, *Fast Discovery of Association Rules*, Advances in Knowledge Discovery and Data Mining, U. Fayyad et, al (eds), The MIT Press, Cambridge Massachusetts, 1996.

[3] R. Agrawal, T. Imielinski, and A. Swami, *Mining Association Rules between Sets of Items in Large Databases*, In Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., 1993.

[4] Ryszard S. Michalsk and Kenneth A. Kaufman, *Data Mining and Knowledge Discovery: A Review of Issues and a Multistrategy Approach*, Machine Learning and Data Mining – Methods and Applications, Ryszard S. Michalsk, Ivan Bratko and Miroslav Kubat, John Wiley & Sons, 1997.

[5] R. Agrawal and R. Srikant, *Fast Algorithms for Mining Association Rules*, Proceeding of the 20th VLDB Conference Santiago, Chile, 1994.

[6] U. M. Fayyad, G. Piatetsky-Shapiro and P. Smyth, *From Data Mining to Knowledge Discovery: An Overview*, Advances in Knowledge Discovery and Data Mining, U. Fayyad et, al (eds), The MIT Press, Cambridge Massachusetts, 1996.

[7] R. Agrawal, T. Imielinski and A. Swami, *Database Mining: A Performance Perspective*, IEEE Transaction on Knowledge and Data Engineering. Special Issue on Learning and Discovery in Knowledge Based Databases, December 1993.

[8] R. J. Brachman and T. Anand, *The Process of Knowledge Discovery in Databases,* Advances in Knowledge Discovery and Data Mining, U. Fayyad et, al (eds), The MIT Press, Cambridge Massachusetts, 1996.

[9] B. Lent, A. Swami and J. Widom, *Clustering Association Rules*, Internet Profiles Corporation, San Francisco, California.

[10] K. Loudon, Mastering Algorithms With C, O'reilly, 1999.

[11] T. H. Cormen, C. E. Leiserson and R. L. Rivest, Introduction to Algorithms, The MIT Press, Cambridge Massachusetts, 1990.

[12] M. Cantù, Mastering Delphi 5, Sybex Inc., San Francisco, 1999.

[13] K. Ali, S. Manganaris and R. Srikant, *Partial Classification using Association Rules*, American Association for Artificial Intelligence, 1997.

[14] R. Srikant, Q. Vu and R. Agrawal, *Mining Association Rules with Items Constraints*, American Association for Artificial Intelligence, 1997.

[15] R. Srikant and R. Agrawal, *Mining Generalized Association Rules*, Proceedings of the 21st VLDB Conference, Zurich, Switzerland, 1995.

[16] R. Agrawal, M. Metha, J. Shafer, R. Srikant, A. Arning and T. Bollinger, *The Quest Data Mining System*.