Department of Computer Science
Series of Publications A
Report A-1999-1

A Knowledge Discovery Methodology for Telecommunication Network Alarm Databases

Mika Klemettinen

University of Helsinki
Finland

Department of Computer Science
Series of Publications A
Report A-1999-1

A Knowledge Discovery Methodology for Telecommunication Network Alarm Databases

Mika Klemettinen

To be presented, with the permission of the Faculty of Science of the University of Helsinki, for public criticism in Auditorium XII, University Main Building, on January 22nd, 1999, at 12 o'clock noon.

University of Helsinki
Finland

Contact information

Postal address:

Department of Computer Science
P.O.Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki
Finland

Email address: postmaster@cs.Helsinki.FI (Internet)

URL: http://www.cs.Helsinki.FI/

Telephone: +358 9 708 51

Telefax: +358 9 708 44441

ISSN 1238-8645
ISBN 951-45-8465-1
Computing Reviews (1998) Classification: H.2.7, H.2.8, H.3.3, I.2.1, I.2.6
Helsinki 1999
Helsinki University Printing House

A Knowledge Discovery Methodology for Telecommunication Network Alarm Databases

Mika Klemettinen

Department of Computer Science
P.O. Box 26, FIN-00014 University of Helsinki, Finland
Mika.Klemettinen@cs.helsinki.fi, http://www.cs.helsinki.fi/~mklemett/

PhD Thesis, Series of Publications A, Report A-1999-1
Helsinki, January 1999, 138 pages
ISSN 1238-8645, ISBN 951-45-8465-1

Abstract

Data mining (knowledge discovery in databases, KDD) aims at the discovery of interesting regularities or exceptions from large masses of data. The area combines methods from databases, machine learning, and statistics.

Fault management is an important but difficult area of telecommunication network management: networks produce large amounts of alarm information which must be analyzed and interpreted before faults can be located. Alarm correlation is a central technique in fault identification. While the alarm correlation systems are widely used and methods for expressing the correlations are maturing, acquiring the knowledge necessary for constructing an alarm correlation system for a network and its elements is difficult.

We describe a partial solution to the task of knowledge acquisition for alarm correlation systems. We present a method and a tool for the discovery of recurrent patterns of alarms in databases. These patterns can be used in the construction of real-time alarm correlation systems. The main type of patterns we look for are episode rules which have the following form: "If a certain combination of alarms occurs within a time period, then another combination of alarms will occur within a time period with a certain probability."

Our method finds all episode rules with respect to the given, rather loose criteria. With the described method and tool the construction of correlation systems becomes easier. This methodology has been implemented in a data mining system called TASA (Telecommunication Alarm Sequence Analyzer).

One of the main contributions of this thesis is the description of the complete interactive and iterative knowledge discovery process from both practical and analytical point of view. Discovering knowledge from data is a process containing several steps: understanding the domain, preparing the data set, discovering patterns, postprocessing of discovered patterns, and putting the results into use. Although the main emphasis of the thesis is in knowledge discovery from telecommunication data, the methods apply also to other types of data.

The work with the knowledge discovery process and large, real-life data sets, has directed our research towards inductive databases, i.e., databases that contain inductive generalizations about the data, in addition to the usual data. Within inductive databases, the KDD process can be described as a sequence of queries.

Computing Reviews (1998) Categories and Subject Descriptors:

H.2.7 Database Administration [Database Management]: Data warehouse and repository
H.2.8 Database Applications [Database Management]: Data mining
H.3.3 Information Storage and Retrieval: Information Search and Retrieval
I.2.1 Artificial Intelligence: Applications and Expert Systems
I.2.6 Artificial Intelligence: Learning

General Terms:

Algorithms, Experimentation, Design

Additional Key Words and Phrases:

Knowledge discovery process, Telecommunications, Episode rules, Interestingness, Inductive databases

Acknowledgments

To begin with, I would like to thank Professor Heikki Mannila, my supervisor, for his patience during the past few years, not to mention the encouragement he has given to me. Without you, with rather high probability, this thesis would not exist.

A great support has been our "BOHM" group with Barbara, Oskari, and Helena. With you, we created the concept of "soft research" which has guided us through the years in this sometimes cold and cruel world. And when I add the remaining data miners, especially Hannu, Pirjo, Inkeri, Kimmo, Ykä, and my new French friend Jef, not forgetting all the other friends and colleagues at the department, I have the pleasant group in which I have been privileged to work.

I would also like to thank my mother Anneli and brother Juha, not forgetting Emppu the dog, for their support and existence. One should not also forget the friends outside the department: Your constant inquiries about my progress with the thesis surely speeded up the completion of this work.

The next one is for You all.

My Friend
Please, hear my call
You know, without You
Oh yes, I would fall

You share my good times,
You also share my pain
You are there every day,
You are there to guide my way

Nothing can replace You,
'cause You are in my Heart
I hope, my Friend
we'll never fall apart

© *mk'98*

Many thanks to all of you, who gave invaluable comments to the manuscript versions of this thesis, and "showed me the light" during the dark times.

Finally, I'd like to thank the Technology Development Centre of Finland (Tekes) and industrial partners (Nokia Telecommunications/Cellular Systems, Helsinki Telephone Corporation HPY, Radiolinja and Tampere Telephone Corporation TPO), Nokia Foundation, and the Helsinki Graduate School of Computer Science and Engineering for their financial support, and the Department of Computer Science at the University of Helsinki, headed by Professors Martti Tienari and Esko Ukkonen, for providing the working conditions.

Mika Klemettinen

For the Reader

When I was a child
I had childish dreams
Dreams of things
I could never reach

When years went by
They fainted away
Those dreams of mine
I had one day

But I want to believe
In times that used to be
I want to believe
In that child in me

Take my hand
Believe with me
Believe in magic
Believe in dreams

Contents

Chapter 1

Introduction

Today, in their daily operation, enterprises and organizations gather and store gigabytes or even terabytes of data in, e.g., customer and transaction databases. Also measurement devices provide billions and billions of bytes of data, and it is no longer possible to analyze it manually using traditional methods and technologies well-known from statistics and computer science.

Understanding the data, or learning about the implicit information in the data, is often important for strategic decision support or for technical applications. As the amount of data grows fast, automatic or semiautomatic discovery methods are needed to promote data understanding. Consider, for instance, how a bank or a supermarket could utilize patterns discovered in their customer transactions, e.g., for product decisions, marketing or risk management, or how telecommunications network alarms could be correlated and predicted based on regularities in the alarm data. Finding interesting regularities manually, for example with statistical methods, is time consuming.

Data mining (or knowledge discovery in databases, KDD) is a field of increasing interest combining databases, machine learning, and statistics. The aim of KDD as a research field is to develop methods for analyzing large amounts of data by discovering interesting regularities and exceptions (see e.g. [FPSSU96, PSF91]).

This thesis focuses on the methods developed and knowledge acquired during the telecommunication oriented data mining research at the Department of Computer Science of the University of Helsinki. This three-year project started during fall 1994 in cooperation with four Finnish telecommunication companies, Nokia Telecommunications/Cellular Systems, Helsinki Telephone Corporation HPY, Radiolinja and Tampere Telephone Corporation, and the Technology Development Centre of Finland (Tekes).

Although the perspective in this work is in telecommunication area, the methods can be generalized for other domains as well; as examples, we consider student enrollment data and document collections. See the descriptions of the data sets used in this thesis in Section 1.3.

One of the main contributions of this thesis is to describe a complete knowledge discovery process. We review the discovery process of understanding the domain, preparing the data set, discovering patterns, post-processing of discovered patterns, and putting the results into use based on experience in applying rule-based discovery methods. Additionally, this thesis discusses inductive databases, i.e., databases that contain inductive generalizations about the data, in addition to the usual data. The KDD process can then be described as a sequence of queries on an inductive database.

Many of the ideas in this monograph are based on several published articles and reports, which are joint works with various members of our Data Mining Group led by professor Heikki Mannila. Articles [HKM+96a, HKM+96b, KMT96, KMT97, KMT99a, KMT99b] discuss TASA, telecommunications, and the KDD process. Interestingness-related issues are discussed in [KMR+94, TKR+95], but also in many of the articles above. Articles [AHKV97, AHKV98] consider mining document collections, and inductive database concepts are presented in [BKM98a, BKM98b].

1.1 Overview

Methods Used. *Association rules*, introduced by Agrawal, Imielinski, and Swami [AIS93], are a class of simple but powerful regularities in binary data. An association rule of the form $X \Rightarrow Y$, where X and Y are sets of attributes, states that in the rows of the database where the attributes in X have value true, also the attributes in Y have value true with high probability.

For association rule creation, we need a data set with events, where each event has one or more attributes and the order of events is not important. The first application areas before network alarms were, e.g., basket data and student enrollment data. If we take an example from a retail database, events correspond to customer transactions, and an attribute to one purchased item. Together these items make up so called basket data.

We mainly use a IF–THEN–WITH notation for rules, where the IF part represents the left-hand side of the rule, the THEN part the right-hand side, and the WITH part all the numerical values related to the rule. With association rules these numerical values are frequency and confidence, but in general they can be any measures reflecting the characteristics of the rule.

Typically, these values have to exceed some given threshold.

Simple examples of the use of association rules are the following.

- In basket data analysis (retail databases), we want to find what products are usually purchased together. This information can be used in changing the product selection, in discount planning, and in helping the product placing. We can find, e.g., a high-confidence rule

  ```
  IF    bread
        butter
  THEN  milk
  WITH  conf.(0.9)  freq.(0.05)
  ```

  stating that customers who buy bread and butter, buy quite often also milk. So, we can, e.g., expect that if we give some discount on some of these products, it should also increase the purchase of the two other products.

- In direct marketing, different advertising campaigns need sometimes customer interviews in order to define the potential customers for new products etc. If we find a strong rule

  ```
  IF    academic education
        male
        under 40 years
  THEN  interested in golf
  WITH  conf.(0.7)  freq.(0.06)
  ```

  it is worthwhile to direct, e.g., a golf club share campaign primarily to the group defined by the rule, i.e., for males under 40 years with academic education. Similar technique can be used in marketing new services.

This concept holds also in telecommunication networks alarm handling. Within alarm databases, an alarm usually equals to the event, and attributes are typically alarm identifier, alarm severity, alarming network element, occurrence time of the alarm, etc. If we find, e.g., a strong rule

```
IF    network element = 1
      day = Monday
      hour = 14
THEN  error level > 10
WITH  conf.(0.95)  freq.(0.05)
```

we know that within network element 1 there is a substantially high error level on Monday afternoons, and that it could be wise to cut down the network load or to increase the network capacity. To be more concrete, the above rule says that with a probability of 95%, if the alarm properties in the IF part are present, then the THEN part property is also present, and they both are present in 5% of the rows in the alarm database. Note that here we use a clarifying representation "attribute=value" to represent one binary attribute.

For sequential data, e.g. telecommunication alarms, plain association rules are not, however, usually enough, because they do not fully capture time-related phenomena. To this need we developed a modification of the concept of association rules, *episode rules* [MTV95, MT96, MTV97]. Such data can be seen as a sequence of events, where each event is a pair *(event type, time)*. For instance, in the case of WWW log data, an event corresponds to one page access, and time, accordingly, to the time of this event.

With episode rules, we can define time bounds for both the rule IF part and the whole rule. The next episode rule,

```
IF    Home Page of the Dept of Computer Science
      Research Groups
THEN  Home Page of the Data Mining Group
WITH  [20]  [40]  conf.(0.63)  freq.(630/1000)
```

reveals that in 63% of the cases, where pages "Home Page of the Dept of Computer Science" and "Research Groups" are accessed within 20 seconds, also page "Home Page of the Data Mining Group" is accessed within 40 seconds. This rule has 630 occurrences in the database, while the IF part of the rule has 1000 occurrences.

This concept, again, applies in telecommunication networks alarm handling. Within alarm databases, an event usually equals to the alarm identifier, and the time to the occurrence time of the alarm.

The general form of episode rules in alarm data is the following:

> If a certain combination of alarms occurs within a time period, then another combination of alarms will occur within a time period with a certain probability.

The rule format is fairly simple, but powerful enough to capture various phenomena in alarm sequences. Episode rules are easy to understand, and they closely correspond to the patterns used by many alarm correlation systems. For instance, the following ordered rule (the order of items in the rule corresponds to their order in the data)

```
IF     link alarm
       link failure
THEN   high fault rate
WITH   [5]   [60]   conf.(0.90)   freq.(151/168)
```

tells that in 90% of the cases, where alarms "link alarm" and "link failure" have occurred within 5 seconds (in that order), an alarm "high fault rate" has also occurred within 60 seconds. This rule has 151 occurrences in the database, while the IF part of the rule has 168 occurrences.

Postprocessing of Discoveries. With large data sets, it is not only the size of the data set that can cause problems, but also the size of the resulting knowledge data set. In addition, many rules can be trivial or too detailed. This leads to the problem of finding interesting rules from the set of all rules holding in a data set. In effect, this means addressing a data mining problem of second order. While formal statistical criteria for rule strength and (statistical) significance abound, it is much harder to know which of the discovered rules really interest the user. That is why the exploration of the discovered patterns must be assisted with, e.g., pruning, ordering, and structuring.

An almost trivial ordering criteria for association and episode rules are the values of rule confidence, frequency, and statistical significance. Rules can be pruned by setting thresholds on these properties. One interesting information theory based measure is the *J-measure* [SG92, DLM+98], which can be used in ranking both association and episode rules. A very useful formalism for pruning a rule set are *templates* [KMR+94], pattern expressions that describe the form of rules that are to be selected or rejected. In the context of association and episode rules, templates define a set of rules by specifying what attributes occur in the antecedent, IF part, and what in the consequent, THEN part.

KDD Process and Discovery Methodologies. Experience shows that data mining is not just running some algorithm, but it is an iterative and interactive process containing several steps from understanding the domain to the utilization of the results found.

We have introduced a methodology for knowledge discovery where one first discovers large collections of patterns at once, and then interactively retrieves subsets of the collection of patterns [KMT96, KMT97, KMT99b, KMT99a]. The proposed methodology suits such KDD formalisms as association and episode rules, where large collections of potentially interesting rules can be found efficiently. Our methodology emphasizes the following two characteristics:

1. A large collection of potentially interesting patterns is discovered at once.
2. Different views on the discovered patterns can be used iteratively and interactively.

In our methodology the user has total, explicit control over the operations on a rule set.

Inductive Databases. Maybe one of the major cornerstones in the introduction of data mining in large scale has been the mutual benefit between data mining and data warehouses. With data warehouses enterprises can store huge amounts of data and process it into unified format. Data mining fits into this concept extremely well, because now there is no need to preprocess the existing data just for mining purposes: the preprocessed and cleaned data already exists! One point of view is, though, to think of data mining as an added-value component that can be attached to a data warehouse system.

Still, something seems to be missing: A general, theoretically sound, and at the same time flexible model which could take advantage of existing database and query language technologies. Based on our experience, we introduce a novel framework called *inductive databases* [Man97, BKM98a, BKM98b]. Inductive databases contain inductive generalizations — such as association and episode rules — about the data, in addition to the usual data. The KDD process can then be described as a sequence of, e.g., SQL3 queries on an inductive database. This concept has been suggested in [IM96].

Our basic approach is very simple:

- An inductive database consists of a normal database associated to a subset of patterns from a class of patterns, and an evaluation function that tells how the patterns occur in the data.
- An inductive database can be queried (in principle) just by using normal relational algebra or SQL, with the added property of being able to refer to the values of the evaluation function on the patterns.
- Modeling KDD processes as sequences of queries on an inductive database enables reasoning about and optimizing these processes.

1.2 Organization of the Thesis

First in Chapter 2, we give an overview of the problems related to telecommunication networks and alarm databases. That is, we briefly discuss the network surveillance and the current methods for alarm flow processing. We also introduce our knowledge discovery system TASA (Telecommunication Alarm Sequence Analyzer).

Then, in Chapter 3 we present our knowledge discovery formalisms, association and episode rules, and outline the algorithms. We also give an extensive set of experiments and show that even moderate-size data sets can produce so many rules that it is hard to manage them.

In Chapter 4 we define several techniques and concepts to be used in finding interesting information. We complement the discussion with experiments on several real-life data sets, rule formalisms, and thresholds/techniques.

We then describe in Chapter 5 the whole pattern discovery process and concentrate on our methodology. We give our scenario for building alarm correlation systems and present the whole process from raw data to building blocks for correlation systems.

In Chapter 6 we generalize our approach, and show how a rule-based methodology like association or episode rules could be used in a novel framework called inductive databases.

Finally, Chapter 7 contains concluding remarks.

1.3 Data Sets Used in the Thesis

In this thesis, our focus is on sequential data, like stock exchange rates, telecommunication alarms, World Wide Web log files, user action registration for Human-Computer Interaction studies, or any other series of events consisting of an event type and a time of occurrence. Text can also be seen as sequential data, in many respects similar to the data collected by sensors, or other observation systems. We have selected telecommunication alarm data (Section 1.3.1), and document collections (Section 1.3.2) for experiments with sequential data, and student enrollment data (Section 1.3.3) and telecommunication alarm data for association rule experiments.

1.3.1 Telecommunication Alarm Data

Our principal case study concerns telecommunication network alarm sequences, and episode rules derived from episodes. A telecommunication

#	Alarms	First alarm	Last alarm	Duration	Alarm types
1	19 429	16.08.96 at 21:32:32	19.08.96 at 04:08:32	~2 days	1 982
2	58 616	13.10.96 at 00:00:20	25.10.96 at 11:46:54	~12 days	1 956
3	22 158	12.10.96 at 21:35:14	21.10.96 at 16:57:04	~6 weeks	41
4	98 413	01.04.96 at 07:38:35	10.12.96 at 14:29:52	~8 months	212
5	43 478	26.09.94 at 08:26:23	05.10.94 at 11:49:00	~10 days	2 223
6	73 679	01.02.95 at 05:09:23	22.03.95 at 23:59:41	~7 weeks	287

Table 1.1: Alarm data sets used in this thesis; see also Tables 2.1 and 2.2.

alarm database contains event data from a number of interconnected components, such as switches, in telecommunication networks. Alarms are produced by the components to report abnormal situations. There are typically thousands of alarms a day, and they are of thousands of different types. Alarm data represents an ideal data set for a data mining project: There is lots of data, it is easy to collect, and it is not really used currently but the discovered knowledge has potentially high add-value.

The characteristics of the alarm data sets are described in Table 1.1. See Tables 2.1 and 2.2 in Section 2.1 for a more detailed description of the data sets.

1.3.2 Document Collections

In the experiments with document collections, we use a sample of 14 documents taken from a large collection of Finnish legal texts, originally in SGML (Standard Generalized Markup Language) format. The sizes of the documents varied between 2 500 and 60 000 words (see Table 1.2). The number of words includes punctuation marks; the number of real words is about 20% lower and the number of words and their morphological interpretations is about 20% higher, respectively. The SGML tags, used only in the discovery of structural dependencies, are not included in the word counts. In other experiments, we have also considered, e.g., mining Reuters news articles [AHKV98].

1.3.3 Student Enrollment Data

Our final, simpler case study concerns association rules in the student enrollment database of courses in computer science at the University of Helsinki.

Statute (act/decree)	Size of the SGML file (kilobytes)	Number of Real Words	Number of Words	Number of Words + Interprets	Number of Sentences
Painovapauslaki	44	3 349	4 273	5 233	144
Oikeudenkäymiskaari	381	28 535	38 970	47 250	1 381
Ilmansuojelulaki	29	1 920	2 622	3 143	87
Osuuskuntalaki	218	15 720	19 682	24 646	767
Kolttalaki	74	5 513	6 427	7 869	259
Vesilaki	644	49 330	61 559	75 889	2 053
Tiekuljetussopimuslaki	63	4 938	5 815	7 307	206
Kansaneläkelaki	211	15 497	20 756	25 065	724
Kemikaaliasetus	43	3 161	3 997	4 781	89
Sosiaalihuoltolaki	61	4 190	5 491	6 636	187
Kemikaalilaki	79	5 260	7 169	8 353	241
Laki rikosvahinkojen korvaamisesta valtion varoista	34	2 502	3 445	4 234	124
Tilintarkastuslaki	56	3 857	4 576	5 516	159
Jäteasetus	54	4 187	5 239	6 153	269

Table 1.2: Document collection test material. Number of real words means the actual word count without special characters, commas, etc., while number of words contains also the extra information used in our experiments.

The database instance consists of registration information from fall 1989 to fall 1993 for which we selected from 4 735 students the 1 066 students who had registered for at least two intermediate or advanced level courses; see example courses in Table 1.3. In the test database, there were 113 different courses from the pool of 355 courses available.

The data set was restricted to intermediate or advanced level courses only, because basically everybody has to take almost all the basic level courses, while in intermediate and advanced levels there are several alternative paths you can take.

The goal in analyzing student enrollment data is to obtain accurate and useful information about the interrelationships between enrollments to various courses. Such relationships can be valuable, e.g., for expert systems aimed at planning curriculums and allocating resources. The enrollment data also offers a fruitful ground for experimenting, not only because the data is understandable but also because it contains simple hierarchies (Basic $\leftrightarrow$ Intermediate $\leftrightarrow$ Advanced).

Course number	Course level	Course name
580211-9	Basic	Introduction to Computing
58022-3	Basic	Programming I (Pascal)
58036-0	Intermediate	VAX/VMS System
58038-4	Intermediate	Database Systems I
58042-1	Intermediate	Information Systems
58053-7	Advanced	Design and Analysis of Algorithms
58060-5	Intermediate	Computer Networks
58064-3	Advanced	User Interfaces
58066-7	Advanced	Artificial Intelligence
58127-1	Intermediate	Programming in C
58128-8	Basic	Introduction to UNIX
58147-9	Advanced	Machine Learning
58152097-3	Advanced	Seminar: Research Seminar on Machine Learning and Data Mining

Table 1.3: Example courses of the student enrollment database.

Chapter 2

Telecommunication Networks and Alarm Data

Telecommunication networks are growing fast in size and complexity, and their management is becoming more difficult. Network elements produce large amounts of alarms about the faults in a network. The *operations and maintenance center* (OMC) of the telecommunication network management system receives the alarms generated by the nodes in the network (Figure 2.1). It stores them in an alarm database, may filter them, but most importantly it displays the alarms to an operator, who then decides what has to be done with them. Fully employing this valuable data in network management is difficult, however, due to the high volume and the fragmented nature of the information. Moreover, changes in equipment, software, and network load mean that the characteristics of the alarm data change.

In this chapter we first give an overview of the telecommunication networks alarm databases in Section 2.1 and define some terminology. Then, in Section 2.2, we discuss alarm correlation, for both real-time and off-line analysis of the alarm flow received by the operations and maintenance centers. Our approach to the task of processing alarms is supplementary to alarm correlation. In Section 2.3 we introduce our approach and the knowledge discovery system supporting it.

2.1 Overview and Terminology

There are two main types of network structures, mobile networks and fixed telephone networks. Regardless of the network type, a telecommunication network can be viewed as consisting of a number of interconnected components: switches, exchanges, transmission equipment, etc. In Figure 2.2, there is an example of a mobile network structure. A network management

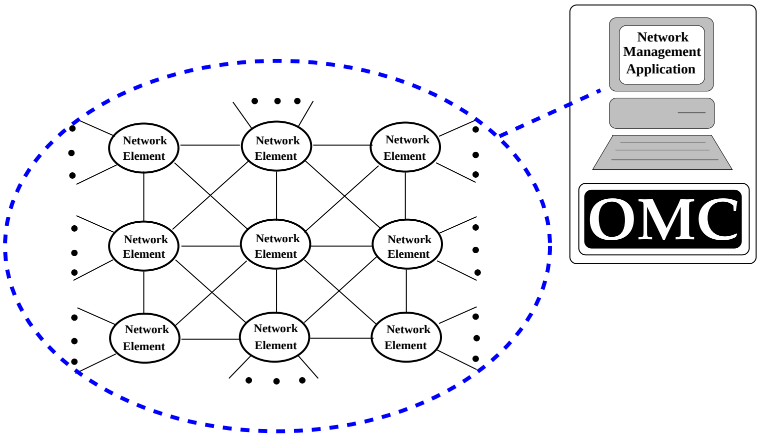


Figure 2.1: The operations and maintenance center (OMC) of the telecommunication network management system receives the alarms generated by the nodes in the network.

system (NMS) controls several mobile station controllers (MSC), which in turn consist of base station controllers (BSC) and base station transceivers (BTS). Furthermore, each component mentioned above contains several subcomponents and actual physical devices. The number of components depends on the abstraction level used in viewing the system, but a network operated by a metropolitan area telephone company can be considered to consist of hundreds or thousands of different-level components. Although the implementation, structure, and problems of mobile and fixed telephone networks are not always the same, in the abstract level these domains are close enough to each other to be considered as a single application domain in this thesis.

Each (sub)component and software module in a telecommunication network can produce *alarms*. An *alarm* is a message emitted by a network element, typically when a problem is encountered. They are messages describing some sort of abnormal situations; they do not necessarily indicate that there has been a problem in the network that is visible to the users.

Unfortunately, each network element has only a narrow view to the network, and can only report the symptoms of the fault from its limited viewpoint. Furthermore, one fault can result in a number of different alarms from several network elements; see Figure 2.2 for the alarm route from lower-level elements via higher-level ones to the network management system. To be exact, it should be noted that alarms do not use the transmission network, but an either virtually or physically separate “management network”.

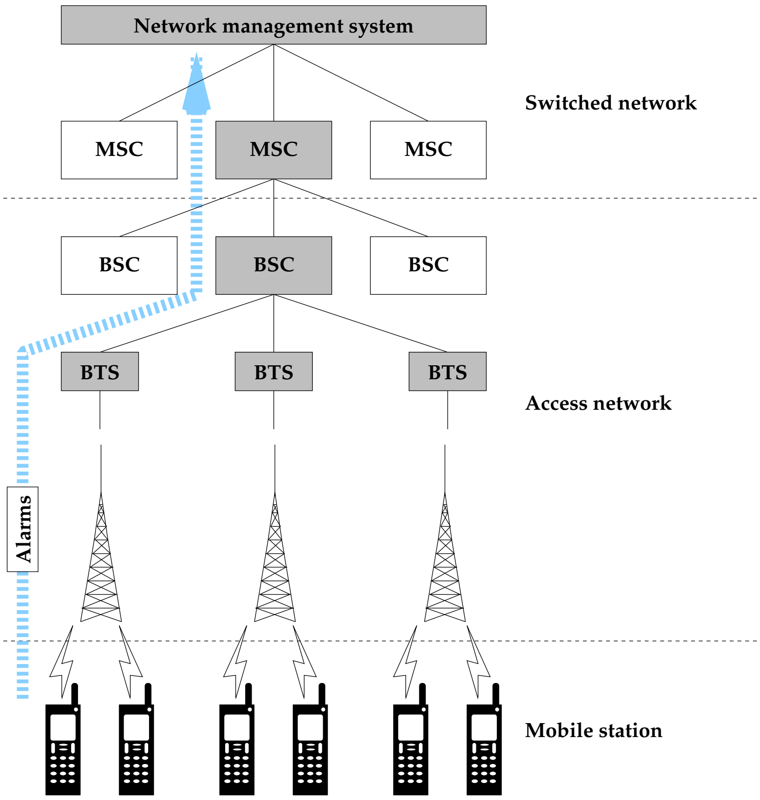
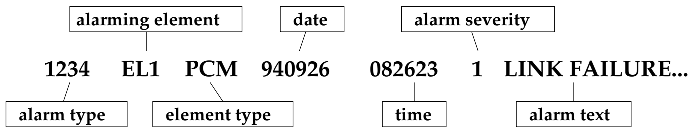


Figure 2.2: Mobile network structure and alarm route from lower-level elements via higher-level ones to the network management system.

We view an occurrence a of an alarm as a triple $a = (t, s, m)$, where t is the *time* of the alarm a, s is the *sender* of the alarm, and m is the *alarm message*. The time is recorded by the sender, typically at a granularity of one second. The sender of the alarm can be identified at the level of, e.g., a network element or a managed object. The alarm message contains all the information about the problem. Each alarm message must, of course, be interpreted in the context of the sender.

The information contents of alarm messages vary a lot. Some alarms concern problems in logical concepts, such as virtual paths, some concern physical devices, such as power supplies. Some alarms report a distinct failure, e.g., that the incoming signal is missing, whereas some only report a high error rate without any hint for the cause.

Example 2.1 The format of an actual alarm can be as follows:



Here, the *date* and *time* fields make up the alarm time t and the *alarming element* field is the sender s. The alarm message m is then considered to contain the information of the rest of the fields. Actually, the alarm type often determines uniquely the other information of m, and in such cases it is possible to consider the alarm type alone as the value of m. □

The properties of alarms that we consider in the rules are formalized as *alarm predicates*. An alarm predicate φ is an expression that can be evaluated for a single occurrence $a = (t, s, m)$ of an alarm. Alarm predicates typically express properties such as "alarm type is *link failure*", "alarm occurred during office hours", or "alarm type is *link failure* or *link alarm*, and alarm severity class is 3".

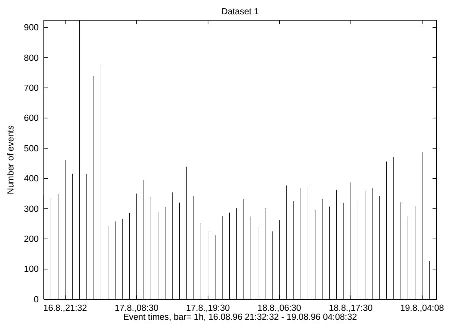
Even in a small telecommunication network there are many alarms. Typically there can be about 200–10 000 alarms a day. Tables 2.1 and 2.2 describe the characteristics of the six example alarm data sets. These data sets have been preprocessed so that only actual alarms have been selected, i.e., warnings, notifications etc., have been removed. The original data sequences usually contain substantially more events — depending on the data and the system the ratio between actual alarms and all events can be almost anything between 5% and 95%. Depending on the data set, also the identification and selection of alarm predicates need work and some background knowledge. Here the number of alarm predicates contains all somehow relevant predicates and some derived or aggregate predicates; see Section 5.2.1 for more detailed discussion of preprocessing the data.

2.2 Alarm Flow Processing with Alarm Correlation

Fault management is an important but difficult area of telecommunication network management: networks produce large amounts of alarm information which must be analyzed and interpreted before faults can be located. Alarm correlation is a central technique for processing the flow of alarms arriving in a management center into a smaller but more useful set of reports.

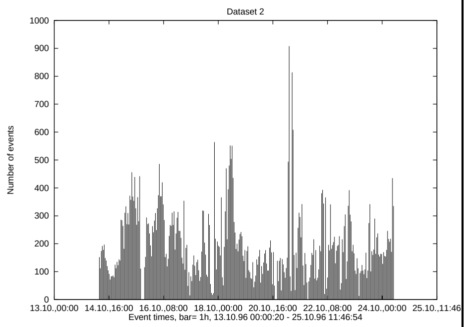
The task of a management center is to correlate the alarms as they are received from the network (Figure 2.3). *Correlating* alarms means combining the fragmented information they contain and interpreting the flow of

Data set 1 characteristics



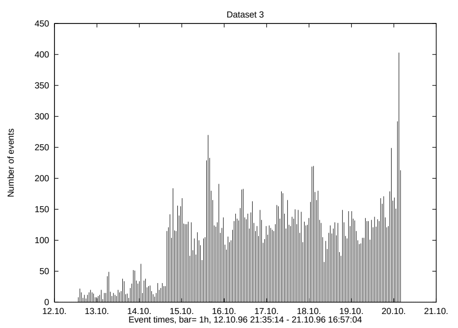
First alarm	16.08.96 at 21:32:32
Last alarm	19.08.96 at 04:08:32
Duration	2d:6h:36m:0s
Alarms	19 429
Alarm predicates	16
Alarm types	1 982
Avg frequency	5.93 alarms/min

Data set 2 characteristics



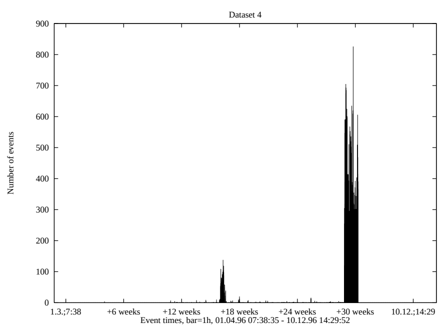
First alarm	13.10.96 at 00:00:20
Last alarm	25.10.96 at 11:46:54
Duration	12d:11h:46m:34s
Alarms	58 616
Alarm predicates	15
Alarm types	1 956
Avg frequency	3.26 alarms/min

Data set 3 characteristics



First alarm	12.10.96 at 21:35:14
Last alarm	21.10.96 at 16:57:04
Duration	8d:19h:21m:50s
Alarms	22 158
Alarm predicates	16
Alarm types	41
Avg frequency	1.75 alarms/min

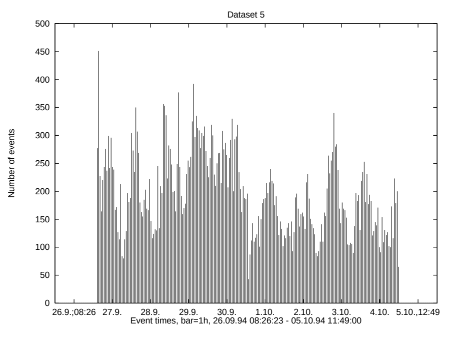
Data set 4 characteristics



First alarm	01.04.96 at 07:38:35
Last alarm	10.12.96 at 14:29:52
Duration	253d:6h:51m:17s
Alarms	98 413
Alarm predicates	61
Alarm types	212
Avg frequency	0.27 alarms/min

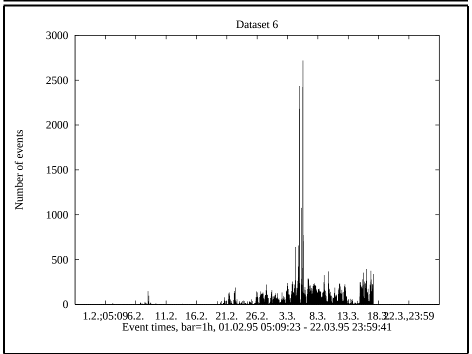
Table 2.1: Example alarm data sets 1–4 characteristics with alarm occurrence times graphs.

Data set 5 characteristics



First alarm	26.09.94 at 08:26:23
Last alarm	05.10.94 at 11:49:00
Duration	9d:3h:22m:37s
Alarms	43 478
Alarm predicates	45
Alarm types	2 223
Avg frequency	3.30 alarms/min

Data set 6 characteristics



First alarm	01.02.95 at 05:09:23
Last alarm	22.03.95 at 23:59:41
Duration	49d:18h:50m:18s
Alarms	73 679
Alarm predicates	29
Alarm types	287
Avg frequency	1.03 alarms/min

Table 2.2: Example alarm data sets 5–6 characteristics with alarm occurrence times graphs.

alarms as a whole. Alarm correlation systems typically perform operations such as

1. *removing* alarms carrying redundant information,
2. *filtering out* low-priority alarms when higher-priority alarms are present, or
3. *substituting* a set of alarms by some new information.

We adopt the following formal view to alarm correlation, similar to the one taken, e.g., in [JW95]. Abstractly, the input to a correlation system is an ordered *alarm sequence* $(t_1, s_1, m_1), (t_2, s_2, m_2), \ldots$ of alarm occurrences, where $t_i \leq t_{i+1}$. A *correlation pattern* describes a situation that can be recognized in an alarm sequence within a time window, or a *correlation window*, of a given length. Typically, a correlation pattern is an expression on the set of active alarms of, e.g., the last five minutes. If in a given window there is a set of alarms that matches the correlation pattern, then the set is said to be an occurrence of the pattern. Associated with each correlation pattern is a *correlation action*, which is to be executed when there is an occurrence of the corresponding pattern in a window. For every correlation, there can be a *correlation lifespan* which, for example, tells how long the

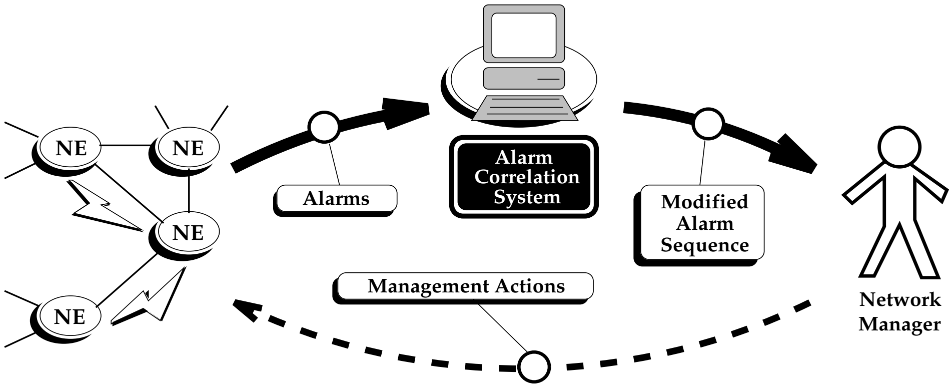


Figure 2.3: The flow of alarms from a telecommunication network is processed in an alarm correlation system into a more useful form.

combined alarm created by correlation action is valid.

In [JW93] and [JW95], Jakobson et al. have considered the following types of alarm correlation:

1. $[A, A, \ldots, A] \Rightarrow A$ — *Compression*: Reducing multiple occurrences of an alarm into a single alarm
2. $[A, p(A) \notin H] \Rightarrow \emptyset$ — *Filtering*: Filter out alarms A whose values $p(A)$ do not belong to the set H of legitimate values
3. $[A, \mathbf{C}] \Rightarrow \emptyset$ — *Suppression*: Inhibit alarm A, e.g. a low-priority alarm, in context $\mathbf{C}$, e.g. in the presence of a high-priority alarm
4. $[n \times A] \Rightarrow B$ — *Count*: Count and threshold the number of repeated arrivals of the "same alarm"; e.g., replace n occurrences of A with single B
5. $[A, A \subset B] \Rightarrow B$ — *Generalization*: Replace an alarm by its superclass
6. $[A, A \supset B] \Rightarrow B$ — *Specialization*: Replace an alarm by an alarm of a more specific subclass
7. $[A\ \mathbf{T}\ B] \Rightarrow C$ — *Temporal relation* $\boldsymbol{T}$ *("before"/"after")*: Correlate alarms depending on the time and order of their arrival

Jakobson et al. have implemented their correlation model, including the above correlation types, in IMPACT, an intelligent management platform

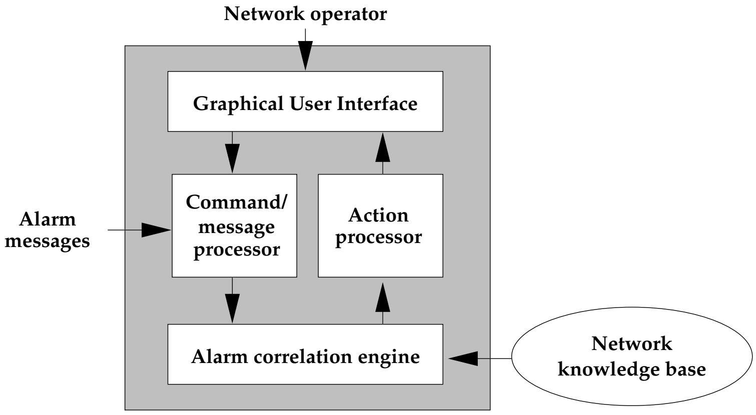


Figure 2.4: Simplified structure of IMPACT.

for alarm correlation tasks. Figure 2.4 presents the simplified structure of IMPACT: Incoming alarms are preprocessed by a message processor, correlation is done within an alarm correlation engine using the attached knowledge base, and actions are taken care of by action processor. Other alarm correlation and filtering systems for telecommunication network management have been presented, e.g., in [GHJ$^+$89, GALU95, JW93, JW95, JP93, Nyg95, BCFK95, Haj96, WBES98].

Given correlation patterns and actions, and a sequence S of alarms, an alarm correlation process continuously observes the incoming alarm sequence S, considers the last time window on S, and executes the actions associated with the patterns that match the window; for an example, see Example 2.2. See also Example 2.3 for a reference to an existing fault management platform.

Example 2.2 Consider a correlation pattern containing two alarms, "link alarm from X with severity 1" (alarm type A) and "high fault rate in X" (alarm type B), where the variable X has to be replaced by the same sender in both alarms. Assume that when alarms of types A and B co-occur, i.e., occur within the correlation window in the right order, they are known to be followed by fatal problems in their sender X. The correlation action could combine the information in the alarms to a high priority warning message "X will probably collapse within an hour" (alarm type C), present it to the network manager, and filter out the original alarms (see Figure 2.5). □

Example 2.3 A nice review of some correlation systems can be found in [GH98]. We briefly describe an existing fault management platform

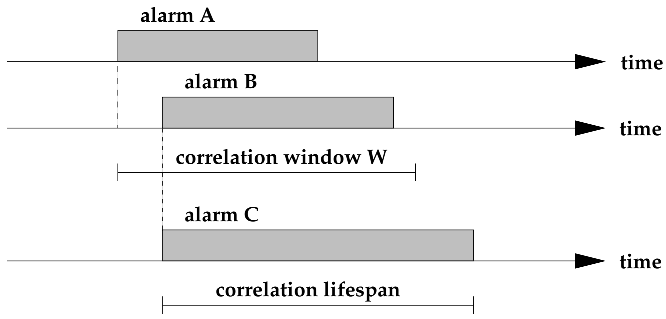


Figure 2.5: A correlation pattern with corresponding correlation action.

(FMP) by Hewlett-Packard [Haj96]. This platform is selected to be a representative example of the existing fault management environments, because it was also used by some of our cooperative industrial companies.

The FMP provides tools for handling alarms of different types of devices from multiple vendors. In practice, this is important, because telecommunication networks can contain a very heterogeneous set of devices which all can emit dissimilar alarms.

The FMP contains special mediation modules that pick up the valid events from the incoming alarm flow, parse these events into uniform representation (X.733 alarm reporting format), and correlate repeated, transient, and related alarms.

The FMP also allows further correlation in the FMP server which actually manages the network alarms. The FMP has two strategies for correlation:

- To correlate repeated/transient alarms, i.e., alarms that are being emitted either continuously or temporarily by a network element.
- To correlate root-cause/related alarms, i.e., alarms that have occurred because of a root-cause alarm.

In the FMP server, it is possible to use the topological knowledge of the entire network, while in the mediation module level this is not the case. The reason for this two-level correlation is that only the relevant alarms should remain to be inspected more thoroughly.

After filtering, correlating, and topology checking the remaining alarms are shown to the network managers in graphical interfaces for further actions.

The correlation is done in the HP OpenView Event Correlation Services (ECS) engine. The ECS engine works on-line and it executes a set of correla-

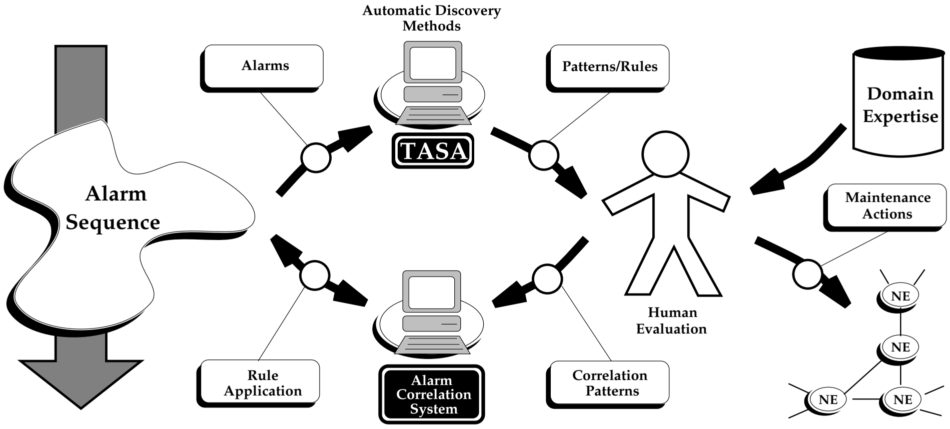


Figure 2.6: An alarm correlation model using discovered alarm patterns.

tion rules that control the processing of event streams. The correlation rules for the correlation engine are created by defining event correlation circuits.

The correlation rules usually contain some fixed orders for the incoming alarms so that correlation rules can be applied. Unfortunately, the alarms do not always come in the order of occurrence because of, e.g., transit delays. In ECS it is possible to handle the delayed and incorrectly ordered events in real time. This, however, requires that the clocks over the network are properly synchronized. □

2.3 Telecommunication Alarm Sequence Analyzer System TASA

While the use of alarm correlation systems is quite popular and methods for specifying the correlations are maturing, acquiring all the knowledge necessary for constructing an alarm correlation system for a network and its elements is difficult. The complexity and diversity of network elements and the large variation in the patterns of alarm occurrences pose serious problems for network management experts trying to build a correlation model.

Our approach to the task of processing alarms is supplementary to alarm correlation. We present methods that can be used in automatic discovery of patterns in alarm databases to help in the construction of alarm correlation systems (see Figure 2.6).

We have implemented our methodology in the TASA (Telecommunication Alarm Sequence Analyzer) system, which is a data mining tool for analyzing telecommunication networks alarms. The purpose of TASA is

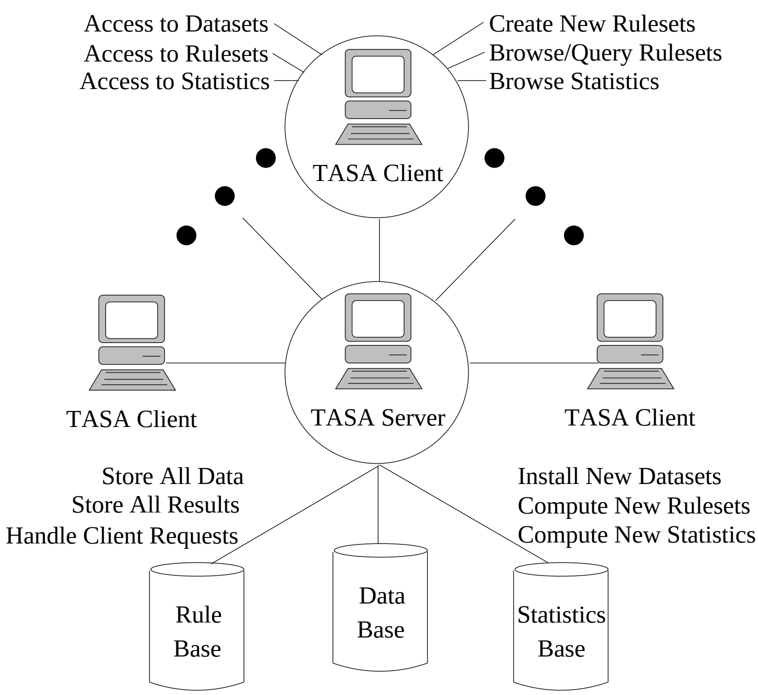


Figure 2.7: New TASA system structure based on client/server architecture.

to aid in the knowledge acquisition phase for creating an alarm correlation model, and to give new views of the alarms. First, a large database of alarms is analyzed off-line, and both temporal connections between different types of alarms, episode rules [MTV97], and relationships between alarm properties, association rules [AIS93], are discovered automatically. Then, the network management specialists analyze the rules found, and based on available background knowledge and knowledge inferred from the rules, select interesting ones. Finally, the selected rules are converted into correlation rules that are applied in real-time fault identification.

Different versions of our TASA system have been in prototype use in four telecommunication companies since the beginning of 1995. In this section we describe the current version of TASA; the discovery methods of the TASA system are presented in Chapter 3 and the main methodology in Chapter 5.

The current version of TASA is a Java-based system with a client/server architecture (see Figure 2.7). The server runs on Unix-based systems (SunOS, Linux etc.), and clients can be installed on any Java-supporting platforms.

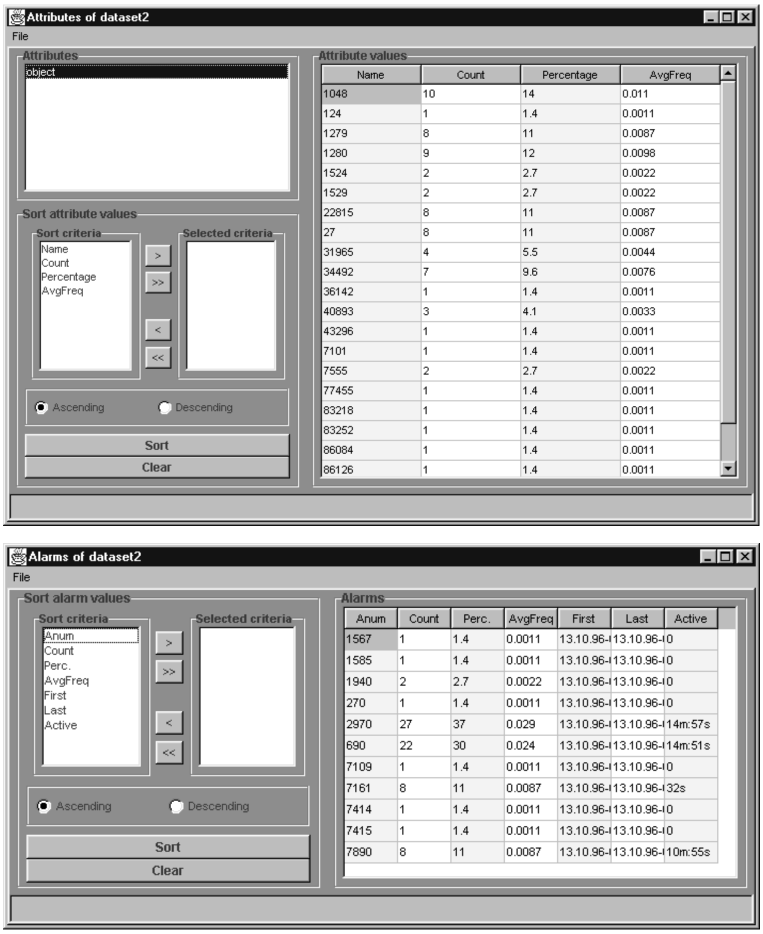


Figure 2.8: Attribute/alarm predicate (top) and alarm (bottom) information windows in the TASA system.

TASA Server. The main component of the system is the TASA server. All the data sets and generated rule sets reside at the server side, and they are stored either in flat files or in a database. This way all operations that require extensive calculation power and storage capacity can be performed efficiently, and the results can be viewed by all clients. Thus, TASA clients only initiate actions at the server side and then show the results. The server is multi-threaded, i.e., it can serve several clients simultaneously.

In fact, the TASA server contains three different servers: a file server, an RMI (Remote Method Invocation) server, and a socket server. The file server takes care of file management and process runs, while the RMI server handles and forwards remote method calls from clients. In the current

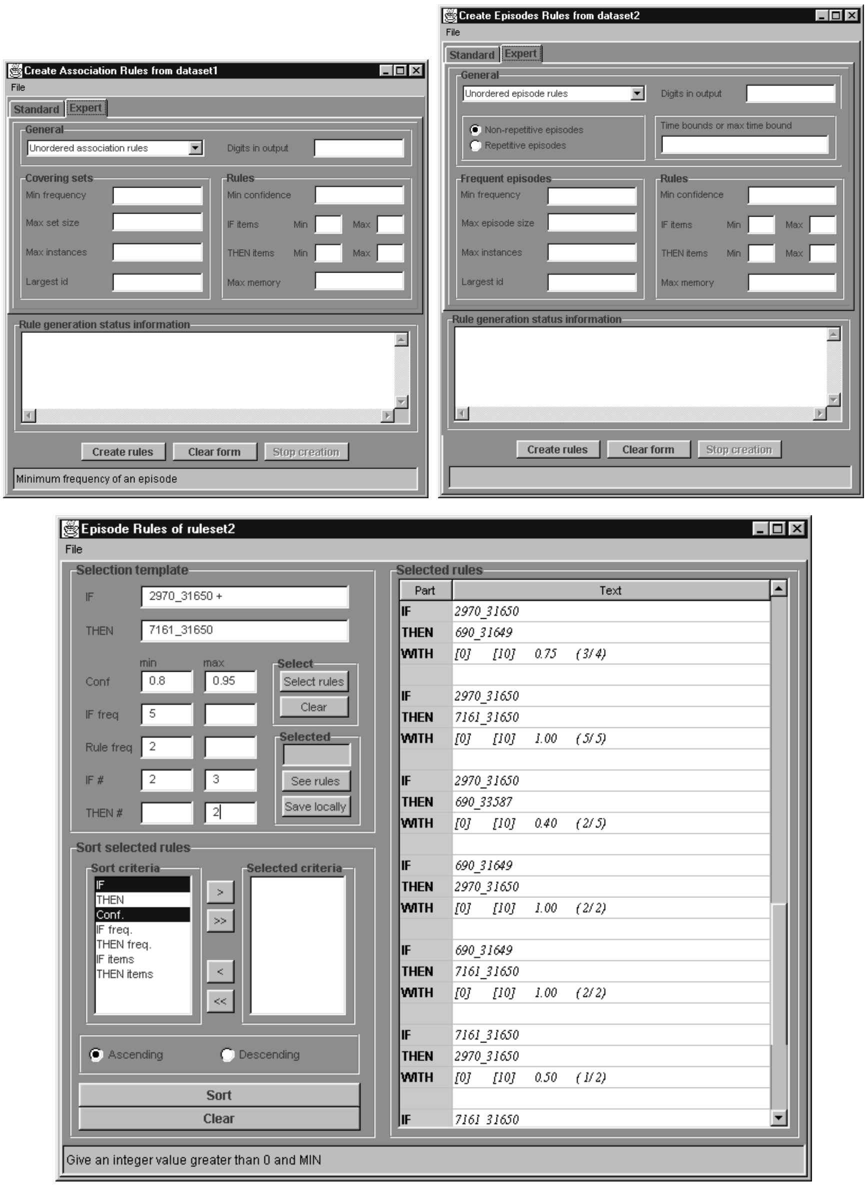


Figure 2.9: Association rule creation (left), episode rule creation (right), and rule viewing windows (bottom) in the TASA system.

system, also a socket server is needed in establishing a connection between a client and a server.

Preprocessing, i.e., preparing the raw data into a suitable form for the analysis, is done at the server side by authorized personnel using either tools provided by TASA or by the production database environment.

TASA Client. At the client side, the user can browse pre-generated information about the data to be analyzed, generate association and episode rules, view generated rules and select rules from the rule collection (i.e. make views). TASA client neither needs any additional working disk space nor superior computational power.

Using the File Manager window, the user can both select data sets for association and episode rule creation and generated rule sets for further analysis. The connection to the TASA Server is created by selecting the login window from the main menu.

The TASA system shows some basic statistics about the data sets, alarms of the data set, and predicates used; see Figure 2.8. The data set statistics contains, e.g., information about the total number of alarms and their average frequency. The alarm information, in turn, lists all alarm types and, e.g., their number and percentage of all alarms.

In the rule creation part, the user can give detailed instructions for rule generation as described in Section 5.2.2. The rule viewing window is based on the use of templates with some additional features for defining the number of items in the IF and THEN parts of the rule, and for sorting the rules. See Figure 2.9 for rule creation and rule viewing related windows; more detailed information about the template concept can be found in Section 4.3.

Usability Tests. For the acceptance of the KDD system, it is essential to meet the needs of the user and discuss the planned or to-be-implemented KDD system with actual workers. The usability of an early version of the TASA system was tested in the usability laboratory of the Helsinki University of Technology. Before the actual user tests, the system was evaluated heuristically with the *pluralistic usability walk-through* technique [Bia91] by those in charge of running the tests.

The group of test users consisted of four alarm handling experts from telecommunication companies. Their backgrounds were more or less technical, and they had no former experience with knowledge representation methods. The test persons were presented 11 different tasks consisting of navigation tasks and tasks of finding interesting rules from the discovered ones. Many of the test cases were hard for the test users, and only half of them accomplished the test tasks of the third set (finding interesting rules). The main reason was that the terminology used in the system was quite unfamiliar to them. Especially this was the case for the episode and association rule concepts. The visual walk-throughs were very fruitful. The mental models of the users — i.e. how the users understand the structure, the concepts, and the functionality of the system — were at least partially uncovered, clarifying the reasons for difficulties in the test cases. Despite the problems, the system as a whole got encouraging and positive comments.

As the goal of knowledge discovery is to find useful patterns, an obvious requirement for success in this task is that the KDD tools and the knowledge representation formalisms are such that the user is able to find and understand the useful patterns — even the most valuable information remains unusable, if it is not found or understood. While KDD is largely guided by real-world needs and applications, there still is the risk that a system is very efficient from the scientific point of view, but it fails to meet the users needs because of, e.g., an unintuitive pattern formalism or an unattractive user interface. In short, the usability of KDD tools has an essential, often perhaps under-estimated role.

2.4 Discussion

Telecommunication networks are large, network elements are complex, and different components produce lots of alarms. The number of correlation patters can be very large, and acquiring them from technical experts is a tedious task. Correlations may pass unnoticed by the experts, for different reasons. It can be that an expert knows a correlation but did not come to think about it, or a correlation can be such that the expert did not even know there was a connection between certain alarms. Both networks and network elements evolve quickly over time, so a correlation system is never complete. It also takes time for the experts to learn new correlations and to modify existing ones.

We propose a semi-automatic process, with automatic methods for the analysis of alarms, to aid in the knowledge acquisition phase, and to give new views to the alarms. The central idea is to discover recurrent patterns of alarms. Such a pattern can be, for instance, a set of alarms that occurs frequently within a time window of a given width, or a pattern can state that a certain alarm tends to be followed by another alarm from the same sender. The discovery process is further discussed in Chapter 5.

Chapter 3

Knowledge Discovery with Association and Episode Rules

Different methods of knowledge discovery can be used on a sequence of alarms, for example neural networks or rule-based representations. If the goal would be just to obtain good predictive performance, a neural network could be useful: there is impressive evidence on the wide applicability of neural networks.

We use rule-based formalisms for knowledge discovery in the TASA system. The reasons for choosing this type of knowledge to be discovered are the following.

1. *Comprehensibility*: episode and association rules are easy to understand for humans, and the operators currently handling the alarm sequences like to express their knowledge about the alarms using something close to these types of rules.
2. *Characteristics of the application domain*: episode and association rules can be representations of simple small causal relationships within the domain. It can be argued that such type of knowledge is suitable for telecommunication networks (as compared to, say, a large causal network describing the whole structure of the domain all at once).
3. *Existence of efficient algorithms*: episode and association rules can be discovered efficiently.

In the following, we first present our rule formalisms, association and episode rules (Sections 3.1 and 3.2). In 3.3, we then show the basic ideas behind the algorithms for these two techniques, and examine their performance through several experiments in Section 3.4.

3.1 Association Rules

Association rules describe how things occur together in the data. In the context of alarm databases the rules are of the following form:

> If an alarm has certain properties, then it will have other given properties.

More formally, an association rule is an expression $X \Rightarrow Y$, where X and Y are sets of predicates. Given a set of alarms, the *confidence* of such a rule is the conditional probability with which all the predicates in Y are satisfied by an alarm in the database given that all the predicates in X are satisfied by the alarm. In other words, association rules describe which properties of alarms occur together, and with which probability. The *frequency* of an association rule is the fraction of alarms that satisfy the rule. The rule is called *frequent*, if its frequency exceeds a user-given threshold; i.e., if all the predicates in $X \cup Y$ occur together frequently enough. Association rules were introduced by Agrawal, Imielinski, and Swami [AIS93]; we use the algorithm Apriori from [AMS$^+$96].

Example 3.1 In the course enrollment database we find an association rule

```
IF    Introduction to Unix
THEN  Programming in C
WITH  conf(0.84)  freq(0.34).
```

The rule states that 84% of the students that have taken "Introduction to Unix" also have taken "Programming in C", and that 34% of all the students actually have taken both courses.

Examples of association rule types found by TASA are:

- If an alarm a is of type "link alarm", then it was sent by the network element "EL1" with probability 0.5 and frequency 0.1.

```
IF    link alarm
THEN  EL1
WITH  conf(0.50)  freq(0.10)
```

- If an alarm a is sent during "office hours" by an element of type "base station", then the alarm has severity level 1 with probability 0.9 and frequency 0.05.

```
IF    office hours
      base station
THEN  severity level = 1
WITH  conf(0.90)  freq(0.05)
```

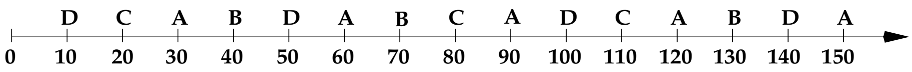


Figure 3.1: Example sequence of event types.

Such rules give an overview of what the alarms in the database are like. A comparison with association rules from last week, last month, etc. can point out trends and changes. Unusually high confidences may point to potential anomalies in the network. □

3.2 Episode Rules

Episode rules describe temporal relationships between alarms. The general form of an episode rule is the following:

> If a certain combination of alarms occurs within a time period, then another combination of alarms will occur within a time period.

The rule format is fairly simple, but powerful enough to capture various phenomena in alarm sequences. Episode rules are easy to understand, and they closely correspond to the patterns used by many alarm correlation systems.

Formally, an *episode* is a pair $\alpha = (V, \leq)$, where V is a collection of alarm predicates, and $\leq$ is a partial order on V.

Given a sequence S of alarms, an episode $\alpha = (V, \leq)$ *occurs* within S if there is a way of satisfying the alarm predicates in V using the alarms of S so that the partial order $\leq$ is respected. Intuitively, episodes consist of alarms that have certain properties and occur in a certain partial order.

The most useful types of partial orders are (1) total orders, i.e., the predicates of each episode have a fixed order; such episodes are called *serial* (or "ordered"), and (2) trivial partial orders, where the order is not significant at all; such episodes are called *parallel* (or "unordered").

Example 3.2 Consider alarm predicates that test the type of the alarm. We can thus identify alarms simply with their types, as in Figure 3.1, where events of types A, B, etc. are shown on a time line. [1]

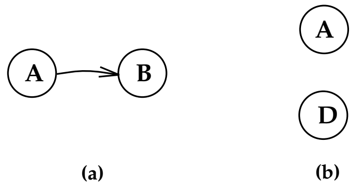
The serial episode consisting of alarm types A and B, i.e., of A and B in the fixed order, occurs several times in the sequence. Alarms A and B may

---

[1] With episode rules, we typically use alarm predicates that only concern alarm types. With association rules, however, also other alarm predicates are useful.

be causally related — the network management expert should verify this — and perhaps a correlation pattern that predicts B when A has just occurred could be useful. A parallel episode consisting of alarm types A and D, i.e., A and D in either order, occurs also several times.

Episodes can be drawn as directed acyclic graphs:



Here, the picture of serial episode (a) shows that an alarm of type A is followed by an alarm of type B, whereas parallel episode (b) specifies no relative order on the alarms of types A and D. □

It seems obvious that serial episodes are useful: typically the events that cause alarms have a certain order, and an episode should take that into account. However, parallel (i.e., unordered) episodes are often more useful in practice. The order of incoming alarms may have changed during their transmission, and inaccuracies in the synchronization of clocks in network elements can make it impossible to infer the correct original order reliably.

Most of the interesting rules are in practice not absolutely certain. TASA outputs all rules from a class specified by the user, and gives with each rule a confidence factor, which tells exactly how strong the rule is in the analyzed alarm database.

We have two different methods to discover episode rules. By WINEPI we denote the method based on sliding windows [MTV95, MTV97], and by MINEPI the method which uses so called minimal occurrences [MT96, MTV97].

3.2.1 The WINEPI Approach

The name of the WINEPI method comes from the technique it uses: a sliding window. Formally, given a set E of *event types*, an event sequence $S = (s, T_s, T_e)$ is an ordered sequence of events event_i such that $\text{event}_i \leq \text{event}_{i+1}$ for all $i = 1, \ldots, n-1$, and $T_s \leq \text{event}_i < T_e$ for all $i = 1, \ldots, n$. A *window* on event sequence S is an event sequence $S_w = (w, t_s, t_e)$, where $t_s < T_e, t_e > T_s$, and w consists of those pairs (event, t) from s where $t_s \leq t < t_e$. The value $t_e - t_s$ is called window *width*, W. By definition the first and last windows on a sequence extend outside the sequence, so that the first window contains only the first time point of the sequence, and the last window contains only the last time point.

We define the *frequency* of an episode α as the fraction of windows in which the episode occurs, i.e.,

$$fr(\alpha, S, W) = \frac{|S_w \in \mathcal{W}(S, W) \mid \alpha \text{ occurs in } S_w|}{|\mathcal{W}(S, W)|}$$

where $\mathcal{W}(S, W)$ is the set of all windows S_w of sequence S such that the window width is W.

Intuitively, the WINEPI rules are like association rules, but with an additional time aspect: If alarms satisfying the IF part occur in the right order within W time units, then also the THEN part occurs in the location described by $\leq$, also within W time units. Formally, an *episode rule* is an expression $\beta \Rightarrow \gamma$, where β and γ are episodes such that β is a subepisode of γ. (An episode β is a subepisode of γ, if the graph representation β is a subgraph of the representation of γ.) The fraction $fr(\gamma, S, W)/fr(\beta, S, W)$ is the *confidence* of the episode rule. The confidence can be interpreted as the conditional probability of the whole of γ occurring in a window, given that β occurs in it.

Example 3.3 In our telecommunication alarm database, using WINEPI we find a serial episode rule

```
IF    self reset
      restart failure
THEN  link unavailable
WITH  [60]   conf(0.99)   freq(0.00001)
```

which tells us that in 99 per cent of the cases, where "self reset" and "restart failure" occurred in the given order within 60 seconds, they were also followed by [2] "link unavailable" within the same time period with frequency 0.00001; thus, very rarely. □

3.2.2 The MINEPI Approach

Consider now the question of deciding which alarms in a time window make up the occurrence of an episode. If there are, for instance, two alarms of

[2]Note that using the episode and episode rule formalisms, the THEN part events could occur, e.g., within the IF part. For example, in addition to the rule (1) "self reset, restart failure ⇒ (self reset, restart failure,) link unavailable" (as above) we could have a rule (2) "self reset, restart failure ⇒ self reset, link unavailable, restart failure". We, however, limit the discussion throughout the thesis to the implemented class represented by the rule (1).

type A and one of type B, does the parallel episode consisting of A and B appear once or twice? If once, then with which alarm of type A? The answers to these questions clearly depend on the nature and relationship of the alarms, and there is no single answer. Consider also, e.g., the rule given in Example 3.3: We know that alarms "self reset" and "restart failure" followed by alarm "link unavailable" have occurred within 60 seconds in the given order. We can not, however, separate within what (perhaps shorter) time period the IF part episode has occurred. For these reasons we look at *minimal occurrences* of episodes.

We identify minimal occurrences with the time intervals in which they occur in the following way. Given an episode α and an event sequence S, we say that the interval $[t_s, t_e]$ is a minimal occurrence of α in S, (1) if α occurs in the window corresponding to the interval, and (2) if α does not occur in any proper subinterval. For instance in the sequence

(1) (alarm A, Mar 22 23:46:49)
(2) (alarm A, Mar 22 23:46:57)
(3) (alarm B, Mar 22 23:47:19)
(4) (alarm B, Mar 22 23:47:22)

episode (A_2, B_3) with A in (2) and B in (3) is a minimal occurrence, while (A_1, B_3) and (A_2, B_4) are not. The set of minimal occurrences of an episode α in a given event sequence is denoted by $mo(\alpha)$:

$$mo(\alpha) = \{[t_s, t_e] \mid [t_s, t_e] \text{ is a minimal occurrence of } \alpha\}.$$

We now move on to discuss episode rules. An episode rule gives the conditional probability that a certain combination of alarms occurs within some time bound, given that another combination of alarms has occurred within a time bound. Formally, an *episode rule* is an expression $\beta\,[w_1] \Rightarrow \alpha\,[w_2]$, where β and α are episodes such that β is a subepisode of α, and w_1 and w_2 are integers. The interpretation of the rule is that if episode β has a minimal occurrence at $[t_s, t_e]$ with $t_e - t_s \le w_1$, then the whole superepisode α occurs at interval $[t_s, t'_e]$ for some t'_e such that $t'_e - t_s \le w_2$.

The *confidence* of the rule $\beta\,[w_1] \Rightarrow \alpha\,[w_2]$ is the conditional probability that α occurs, given that β occurs, under the time constraints specified by the rule. To be exact, the confidence of the rule is defined as the fraction $|mo(\alpha)|/|mo(\beta)|$, where $|mo(\beta)|$ is the number of minimal occurrences $[t_s, t_e]$ of β such that $t_e - t_s \le w_1$, and $|mo(\alpha)|$ is the number of such occurrences where there is also an occurrence of α within the interval $[t_s, t_s + w_2]$. The *frequency* of the rule is $|mo(\alpha)|$, i.e., the number of times the rule holds in the database.

Example 3.4 An example of the episode rules based on minimal occurrences and discovered by the TASA system is

```
IF    link alarm
      link failure
THEN  high fault rate
WITH  [20]  [40]       conf(0.63)  freq(246/1056)
```

which tells us that in 63 per cent of the cases, where "link alarm" and "link failure" occurred within 20 seconds, also "high fault rate" occurred within 40 seconds. Moreover, in the data the IF part episode occurred 1 056 times and in 246 cases it was followed by the THEN part episode within the given time window. □

These types of rules suggest groups of alarms that are potentially causally related, and also indicate their temporal relation. Given such rules that hold in an alarm database and some supporting tools, a fault management expert is able to verify whether the rules are useful or not. Some of the rules may be based on known causal connections, some may be irrelevant, while some rules give new insight to the behavior of the network elements.

3.3 Algorithms for Finding Rules

In this section we look more closely at the basic ideas behind both association and episode rule algorithms. We also show some concrete examples of how the algorithms work in practice, while they search for frequent episodes (or predicate sets) and generate rules.

The episode rule discovery task for MINEPI rules in the TASA system can be formalized as follows. Given a sequence S of alarms, a set E of alarm predicates, a class $\mathcal{E}$ of episodes built from the predicates of E, a frequency threshold c, and a set W of integers, find the confidences of all episode rules of the form $\beta\,[w_1] \Rightarrow \alpha\,[w_2]$, where $\beta, \alpha \in \mathcal{E}$, β is a subepisode of α, and $w_1, w_2 \in W$, and the frequency of the rule is at least c. It is not useful to search for episodes with arbitrary partial orders. The size of the result would be vast already for very small episodes, and the resulting episodes would contain a lot of redundancy in the form of different partial orders for the same set of predicates. We use the episode class $\mathcal{E}$ to restrict the search to certain partial orders, in practice to either parallel or serial episodes.

For WINEPI rules, the general setting is quite similar; only the set W of integers is replaced by window width W. The task of finding episodes, however, differs. First, we want to find all frequent episodes $\mathcal{F}(S, W, c)$, and then the confidences of all rules of the form $X \Rightarrow Y$, where $X, Y \in \mathcal{E}, \mathcal{F}$.

1. $C_1 := \{\{e\} \mid e \in E\}$;
2. $i := 1$;
3. **while** $C_i \neq \emptyset$ **do**
4.   *recognition*: read the sequence S, and let F_i be the collection of episodes of C_i that occur at least c times;
5.   *building*: compute C_{i+1} to contain those episodes in $\mathcal{E}$ that have $i+1$ alarm predicates and whose all subepisodes in $\mathcal{E}$ occur at least c times;
6. $i := i + 1$;
7. **od**;
8. **for** all i, output F_i;

Figure 3.2: Algorithm for finding frequent episodes and predicate sets. This is a variation of the generic level-wise algorithm that exploits the apriori trick.

For association rule discovery, an alarm database S, a set E of alarm predicates, and a frequency threshold c are given, and the task is to find the confidences of all rules $X \Rightarrow Y$, $X, Y \in \mathcal{E}$, whose frequency is at least c.

The main algorithms for the three tasks are based on the same basic idea. Our focus will be on the more complex task of discovering MINEPI episode rules. We first discuss how the frequently occurring episodes (or predicate sets for association rules) are computed. We then show how rules can be obtained from this information.

3.3.1 Computing Frequent Episodes or Predicate Sets

We use MINEPI for the following task. Given S, E, $\mathcal{E}$, c, and W as above, find all episodes that have at least c minimal occurrences of length at most win^*, where $win^* = \max\{w \in W\}$. The task for WINEPI would be quite similar: Given S, E, $\mathcal{E}$, W, and c as above, find the frequencies of all episodes from $\mathcal{E}$ such that the frequency is at least c. The general algorithm in Figure 3.2 outputs all frequent episodes for both tasks.

The algorithm starts with simple episodes and proceeds to larger episodes. The algorithm works iteratively: first a collection of candidate episodes C_i is generated and then the number of occurrences is counted for each candidate. Candidates occurring often enough are saved in the collection F_i, and finally new candidates are generated.

Essential for the efficiency of the algorithm is the following observation:

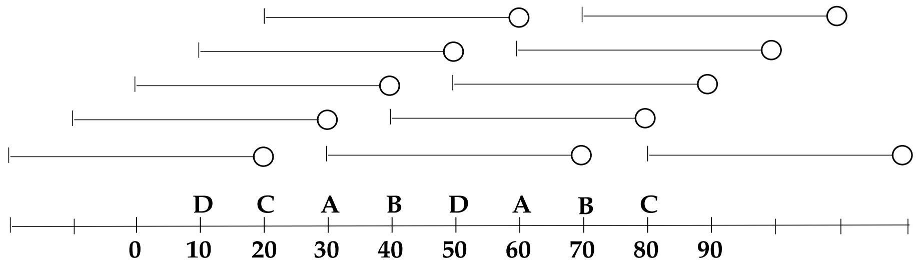


Figure 3.3: Alarm sequence with consecutive windows.

If an episode does not occur often enough, then its superepisodes — which are more specific — cannot occur often enough. Therefore, the candidate collection C_i of episodes is built (Step 5) to contain only episodes whose *all* subepisodes occur often enough. More details can be found in [MTV95, MT96, MTV97].

To find frequent predicate sets with APRIORI, essentially the same algorithm and the same observation hold. Candidates are now sets of predicates, and the input S is seen as an unordered collection of alarms. In Step 4, "recognition", for each candidate set the number of satisfying alarms is computed and compared to the frequency threshold. In Step 5, "building", new candidate sets are constructed such that *every* subset of a candidate set is frequent [AMS$^+$96].

In Examples 3.5 and 3.6, the computing of frequent sets is demonstrated. Analysis of the time complexities are discussed in [MTV97].

Example 3.5 In Figure 3.3 we have an alarm sequence of length 70 seconds (10–80). Window width W is set to 40 seconds and window movement step is set to 10 seconds. When we slide the window through the data, we will get 11 windows $U_1..U_{11}$. By definition the first window contains only the first time point of the sequence, and the last window contains only the last time point. The windows are half-open, i.e., the last second of each window is not included in the window. For example alarm C which occurs at time moment 20 seconds is not in the first window $U_{1,[-20,20[}$. Frequency threshold is set to 40%, i.e., an episode must occur at least in five of the eleven windows.

Suppose that the task is to find all parallel episodes. First, we create singletons, i.e., parallel episodes of size one: A, B, C, D. Then we go to the recognition phase and select those basic episodes whose frequency is at least 40%; here all basic episodes are frequent. From those frequent episodes we then build candidate episodes of size two (set C_2): AB, AC, AD, BC, BD, CD.

Window U_i	Contents of U_i	Parallel episodes occurring in U_i
$U_{1,[-20,20[}$	$[\_,\_,\_,D]$	$\{D\}$
$U_{2,[-10,30[}$	$[\_,\_,D,C]$	$\{C,D\}$, $\{CD\}$
$U_{3,[0,40[}$	$[\_,D,C,A]$	$\{A,C,D\}$, $\{AC,AD,CD\}$, $\{ACD\}$
$U_{4,[10,50[}$	$[D,C,A,B]$	$\{A,B,C,D\}$, $\{AB,AC,AD,BC,BD,CD\}$, $\{ABC,ABD,ACD,BCD\}$, $\{ABCD\}$
$U_{5,[20,60[}$	$[C,A,B,D]$	$\{A,B,C,D\}$, $\{AB,AC,AD,BC,BD,CD\}$, $\{ABC,ABD,ACD,BCD\}$, $\{ABCD\}$
$U_{6,[30,70[}$	$[A,B,D,A]$	$\{A,B,D\}$, $\{AB,AD,BD\}$, $\{ABD\}$
$U_{7,[40,80[}$	$[B,D,A,B]$	$\{A,B,D\}$, $\{AB,AD,BD\}$, $\{ABD\}$
$U_{8,[50,90[}$	$[D,A,B,C]$	$\{A,B,C,D\}$, $\{AB,AC,AD,BC,BD,CD\}$, $\{ABC,ABD,ACD,BCD\}$, $\{ABCD\}$
$U_{9,[60,100[}$	$[A,B,C,\_]$	$\{A,B,C\}$, $\{AB,AC,BC\}$, $\{ABC\}$
$U_{10,[70,110[}$	$[B,C,\_,\_]$	$\{B,C\}$, $\{BC\}$
$U_{11,[80,120[}$	$[C,\_,\_,\_]$	$\{C\}$

Table 3.1: Windows from the alarm sequence in Figure 3.3; "_" means an empty time slot.

In the recognition phase of the second round, we find all parallel episodes AB, AC, AD, BC, BD, CD to be frequent. From those episodes we build the candidate set C_3 for episodes of size three: ABC, ABD, ACD, BCD.

In the candidate set C_3, only ABD occurs in more than four windows, and there are no candidate episodes of size four. □

Example 3.6 Consider the alarm sequence of Figure 3.3, and assume we want to find all serial episodes. Maximum time bound is set to 40 seconds, we use window sizes 10, 20, 30, and 40 seconds, and we require at least one occurrence (absolute number) for every episode.

First, as in the windowing method, we create singletons, i.e., basic episodes of size one: A, B, C, D. While creating them, we also create an occurrence table for them (see "Minimal occurrences" column in Table 3.3); after this first database pass, we do not have to scan the database anymore, but use the created inverse tables. In the recognition phase we see that all the basic episodes are frequent.

From those frequent episodes we then build episode candidates of size two (set C_2): $AB, BA, AC, CA, AD, DA, BC, CB, BD, DB, CD, DC$.

Episode frequencies	Example rules
`D : 0.73`	`IF D`
`C : 0.73`	`THEN A`
`A : 0.64`	`WITH conf(0.75) supp(0.55)`
`B : 0.64`	
`D C : 0.45`	`IF D`
`D A : 0.55`	`A`
`D B : 0.45`	`THEN B`
`C A : 0.45`	`WITH conf(0.82) supp(0.45)`
`C B : 0.45`	
`A B : 0.55`	
`D A B : 0.45`	

Table 3.2: Episode frequencies and example rules with WINEPI.

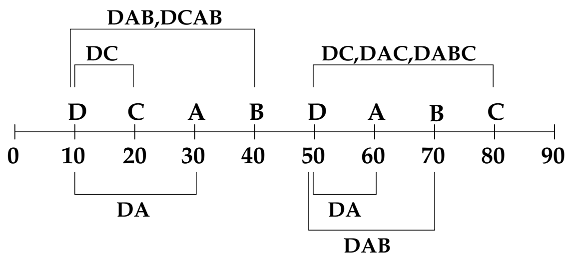


Figure 3.4: Alarm sequence with minimal occurrences.

Then, we use the inverse table to create minimal occurrences for the candidates. E.g., for AB we take all subepisodes, namely A and B, and compute $mo(AB)$ as follows: We read the first occurrence of A (30-30), and find the first following B (40-40). Then we take the second occurrence of A (60-60), and find the first following B (70-70). Then, we continue with BA.

In the recognition phase of the second round, we find all episodes frequent and build the candidate set C_3. Again, due to the low frequency threshold, almost all candidates are frequent. Finally, the same procedure is repeated for candidate set C_4, and episodes $DCAB$ in 10-40, $DABC$ in 50-80, $CABD$ in 20-50, $CBDA$ in 20-60, and $BDAC$ in 40-80 are found to occur (and thus they are frequent for a threshold of 1).

Figure 3.4 gives the minimal occurrences of the example rules in Table 3.3 with two or more alarms. In this example, all singleton alarms are also minimal occurrences. □

Minimal occurrences	Episode frequencies	Example rules
D 10-10 50-50	D : 2	IF D
C 20-20 80-80	C : 2	THEN C
A 30-30 60-60	A : 2	WITH [0] [10] 0.00 (0 / 2)
B 40-40 70-70	B : 2	[0] [20] 0.50 (1 / 2)
D C 10-20 50-80	D C : 2	[0] [40] 1.00 (2 / 2)
D A 10-30 50-60	D A : 2	
D B 10-40 50-70	D B : 2	
C D 20-50	C D : 1	IF D
C A 20-30	C A : 1	THEN A
C B 20-40	C B : 1	C
A D 30-50	A D : 1	WITH [0] [10] 0.00 (0 / 2)
A C 60-80	A C : 1	[0] [40] 0.50 (1 / 2)
A B 30-40 60-70	A B : 2	
B D 40-50	B D : 1	
B C 70-80	B C : 1	
B A 40-60	B A : 1	IF D
D C A 10-30	D C A : 1	A
D C B 10-40	D C B : 1	THEN C
D A C 50-80	D A C : 1	WITH [40] [40] 0.50 (1 / 2)
D A B 10-40 50-70	D A B : 2	[20] [40] 1.00 (1 / 1)
D B C 50-80	D B C : 1	
C D A 20-60	C D A : 1	
C A D 20-50	C A D : 1	IF D
C A B 20-40	C A B : 1	C
C B D 20-50	C B D : 1	THEN A
C B A 20-60	C B A : 1	B
A D B 30-70	A D B : 1	WITH [40] [40] 0.50 (1 / 2)
A B D 30-50	A B D : 1	[30] [40] 1.00 (1 / 1)
A B C 60-80	A B C : 1	
B D C 40-80	B D C : 1	
B D A 40-60	B D A : 1	
B A C 40-80	B A C : 1	IF D
D C A B 10-40	D C A B : 1	A
D A B C 50-80	D A B C : 1	B
C A B D 20-50	C A B D : 1	THEN C
C B D A 20-60	C B D A : 1	WITH [40] [40] 0.50 (1 / 2)
B D A C 40-80	B D A C : 1	[30] [40] 1.00 (1 / 1)

Table 3.3: Minimal occurrences table and episode frequency table.

3.3.2 Rule Generation

We now show how the confidences of rules can be computed, once the frequent episodes or predicate sets are known. In Tables 3.2 and 3.3, the frequent episodes and example rules of Examples 3.5 and 3.6 are given.

Rule generation with Apriori**.** For association rules the rule generation is very simple. Given a frequent predicate set X, for all its subsets $Y \subset X$ we have that the confidence of the rule $(X \setminus Y) \Rightarrow Y$ is the frequency of X divided by the frequency of $X \setminus Y$. Since the frequencies of all frequent sets have already been computed in Step 4, the database is not needed for the rule generation at all.

Rule generation with Winepi**.** Once we know how often each episode from a class of episodes occurs in an alarm sequence, finding the confidences of the frequent rules is easy. In this case, the rule is frequent if the episode α is frequent, i.e., if $fr(\alpha, S, W)$ exceeds the given frequency threshold. The potentially holding rules can now be generated efficiently from the frequent episodes, similarly to the case of association rules: In a single pass through the data, compute the rule confidences and output those rules whose confidence exceeds the confidence threshold.

Rule generation with Minepi**.** Recall that we defined an episode rule as an expression $\beta\,[w_1] \;\Rightarrow\; \alpha\,[w_2]$, where β and α are episodes such that β is a subepisode of α, and w_1 and w_2 are integers. To find such rules, first note that for the rule to be frequent, the episode α has to be frequent. So, rules of the above form can be enumerated by looking at all frequent episodes α, and then looking at all subepisodes β of α. The evaluation of the confidence of the rule $\beta\,[w_1] \;\Rightarrow\; \alpha\,[w_2]$ can be done in one pass through the minimal occurrences of β and α, as follows. For each minimal occurrence $[t_s, t_e]$ of β with $t_e - t_s \leq w_1$, locate the first minimal occurrence $[u_s, u_e]$ of α such that $t_s \leq u_s$. Then check whether $u_e - t_s \leq w_2$.

With the Minepi algorithm it is possible to discover *distinct rules* (rule skeleton consisting of the IF and THEN parts) with different time bound combinations, by fixing not only two bounds but a set of time bounds, e.g., (5,10,20,30,60). Then, all pairs of time bounds respecting the Minepi rule formalism are created. For instance, the distinct rule like

```
IF    alarm:79
      alarm:76
THEN  alarm:77
WITH  [5]   [60]  conf(0.9099)  freq(434/477)
      [10]  [60]  conf(0.9029)  freq(828/917)
      [20]  [60]  conf(0.8513)  freq(1506/1769)
      [30]  [60]  conf(0.7864)  freq(1915/2435)
      [60]  [60]  conf(0.6414)  freq(2209/3444)
```

gives us a quite nice picture of the behavior of the relationship as time bounds change. Although this is quite useful, in the experiments of this thesis we mainly fix the two time bounds for practical reasons. For example, if we want to compare confidences of two or more distinct rules, it is ambiguous what is the confidence of the rule above, while it is unambiguous with the rule in Example 3.4. The time requirement increases slowly as more time bounds are used, but it would not have significant effect on the experiments.

If information about the locations of minimal occurrences of frequent episodes is stored, the confidences for various types of episode rules can be obtained without looking at the original data; they can be computed using the stored frequency values. The time complexity of the confidence computation for a given episode and given time bounds w_1 and w_2 is then $O(|mo(\beta)| + |mo(\alpha)|)$. The confidences for all w_1, w_2 in the set W of time bounds can be found, using a table of size $|W|^2$, in time $O(|mo(\beta)| + |mo(\alpha)| + |W|^2)$. For reasons of brevity and the scope of this thesis, we omit the details.

3.4 Experiments and Performance Results

We have evaluated the efficiency of our rule-based algorithms using several alarm databases from fixed and cellular networks. The experiments have been run on a Citum 4/275 Alpha server with 512 MB of main memory and Red Hat Linux release 5.0 (Hurricane) with kernel 2.0.30.[3] A flat text file copy of the necessary alarm information was used in the tests.

In our test environment typical running times were from a few seconds to an hour, depending on the database and the parameters. Episodes with the alarm type as the only predicate can be discovered from a sequence of about 100 000 alarms with a window width of 60 seconds in a few seconds,

[3] Using a Pentium-based PC with 200MHz processor, kernel 2.0.34, and 64 MB of main memory, the Winepi runs, for instance, were 30–50% faster, but due to the disk space requirements all experiments were performed in the given environment.

whereas finding association rules with over a thousand different predicates can take several hours (the predicates were derived by considering every bit of information contained in alarm messages, e.g., "hour = 0", "hour = 1", etc). In these experiments, we mainly consider the episode rules, but we also give results from association rule experiments as reference.

3.4.1 Experiments with MINEPI

Serial/parallel episodes with/without repetition. Table 3.4 gives performance statistics for finding frequent episodes in the alarm data set 4 with various frequency thresholds. To compare, Table A.1 in Appendix A reflects the characteristics of the data set when repetition is allowed in episodes. By repetition we mean that an episode can contain alarms of the same type, e.g., AAA.

Table 3.4 represents performance for finding (a) serial and (b) parallel episodes without repetition from data set 4, with the maximum time bound $win^* = 60$ s. Table A.1, again, represents performance for (a) serial and (b) parallel episodes with repetition from data set 4, with the maximum time bound $win^* = 60$ s. In Table A.1(a), virtual memory exceeded during iteration 11 with threshold 10. [4] Most of the iterations contained only 1 candidate and frequent episode. For example, with threshold 500 there was only one candidate/frequent episode starting from iteration 13 and with threshold 20 episodes starting from iteration 43. Also in Table A.1(b), most of the iterations contained only 1 or a few candidates and frequent episodes. In both Tables 3.4 and A.1, the number of iterations equals the sizes of the largest episodes + 1; the bound for the iterations was set to 100.

The number of frequent episodes decreases rapidly as the frequency threshold increases. Without repetition, the running times are moderate; between 23 and 115 seconds for thousands of discovered episodes. With repetition and low frequency thresholds, the amount of candidates and frequent episodes increases, and thus cause running times of 1 to 10 minutes. One has to remember that in all cases, a "constant" component is the first iteration when frequent singleton episodes are searched for; in this case the amount of time spent during the first iteration was 21–22 seconds; see Table A.2(a)–(h) in Appendix A for iteration examples.

When searching for serial episodes with and without repetition, the number of candidate episodes was greater than when searching for equivalent parallel episodes; in the former the ratio was 2–3, and in the latter around 1–2. With the number of frequent episodes, on the other hand, the ratio

[4] Unlike APRIORI and WINEPI, the implementation of MINEPI uses only main memory.

behaved similarly with low frequencies, but as the threshold increased, the ratio changed to the opposite; the change was much faster when repetition was present.

Figure 3.5 shows the amount of episode candidates and their generation times, and Figure 3.6 does the same for frequent episodes and their calculation times. The time requirement of the algorithm is linear in the number of alarms, and much larger databases can be analyzed with acceptable response times.

Serial episodes and rules for all alarm data sets. Table 3.5 gives the performance of finding serial episodes with time bounds 10–30 s, 20–60 s, and 30–120 s, and frequency threshold 10. Also, the number of candidates, frequent episodes, and distinct rules are given.

As the time bounds are increased, the number of both candidate and frequent episodes increases. With data set 4, the shift from time bounds 10–30 s to 20–60 s is quite dramatic: the number of candidate episodes grows by a factor of 4, and the number of frequent episodes by a factor of 7, while for the other data sets the corresponding factors are around 1.5 and 2–3. In reality, the jump is even more dramatic between time bounds 20–60 s and 30–120 s, when the number of candidates explodes into 1 050 339. This can not, however, be seen in the result table, Table 3.5, because virtual memory was exceeded during the frequent episode calculation, and we had to use frequency threshold 40 in order to get some results.

A similar phenomenon as with data set 4 occurs also with data set 5 when shifting from 20–60 s to 30–120 s. By carefully inspecting the numbers and the data using these results as a guidance, the network manager can, for instance, get valuable information about the nature of the data and its relationships. For instance, by looking at one distinct rule instance [5]

```
IF    alarm:13
      alarm:75
THEN  alarm:82
WITH  [10]  [30]   conf(0.04128)  freq(9/218)
      [20]  [60]   conf(0.09390)  freq(40/426)
      [30]  [120]  conf(0.14038)  freq(89/634)
```

we can see the evolution of both confidence and support; even more useful is to create rules by using a set of time bounds, e.g., (10,20,...,120) and study all combinations from 10–120 s to 120–120 s. This inspection, if not enough

[5]The alarm is from real rule sets but the alarm identifiers are changed in order to prevent the identification of the data origin.

MINEPI: SERIAL EPISODES WITHOUT REPETITION, $win^* = 60$ s					
Frequency threshold	Candidates	Time (s)	Frequent Episodes	Time (s)	Iterations
10	166929	41.38	58393	74.92	8
20	72165	13.93	23482	64.84	7
30	44237	7.42	13913	52.89	7
40	30718	4.68	9889	48.61	6
50	22971	3.26	7438	45.50	6
100	9898	1.14	2912	38.52	6
250	3369	0.28	855	31.77	5
500	1538	0.09	295	29.21	4
1000	602	0.04	91	25.19	4
2000	326	0.01	35	23.72	3

(a)

MINEPI: PARALLEL EPISODES WITHOUT REPETITION, $win^* = 60$ s					
Frequency threshold	Candidates	Time (s)	Frequent Episodes	Time (s)	Iterations
10	51872	11.46	42131	57.16	12
20	24807	4.48	19699	48.37	11
30	16304	2.68	12549	44.31	11
40	11543	1.75	8744	41.73	11
50	9210	1.34	6995	40.21	10
100	4423	0.61	3359	35.39	9
250	1688	0.19	1030	30.38	8
500	898	0.09	414	28.45	7
1000	439	0.08	169	26.49	6
2000	302	0.03	73	23.84	5

(b)

Table 3.4: Performance of the MINEPI algorithm for serial (a) and parallel (b) episodes without repetition from data set 4; maximum time bound $win^* = 60$ s.

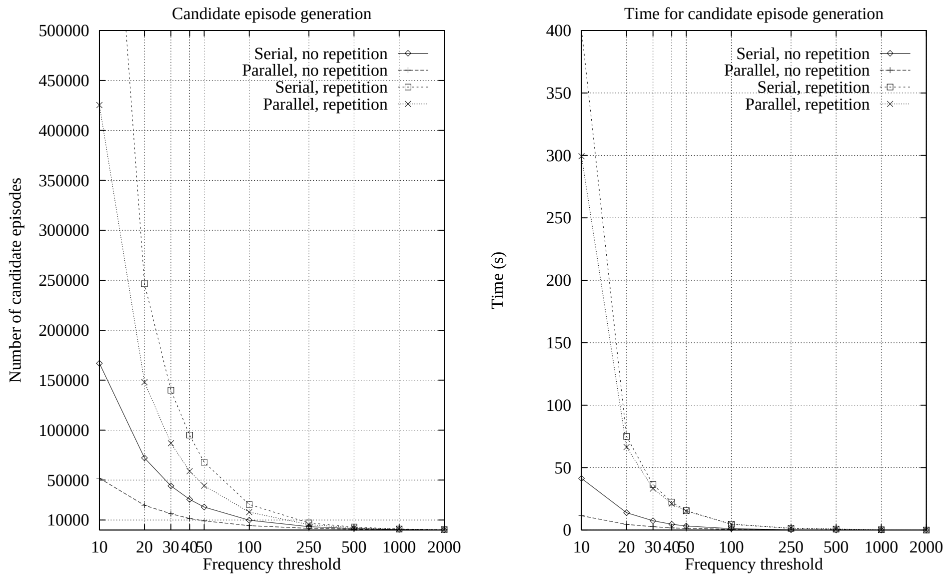


Figure 3.5: The number of candidate episodes and the time needed for generating them from data set 4 with serial and parallel episodes, without and with repetition.

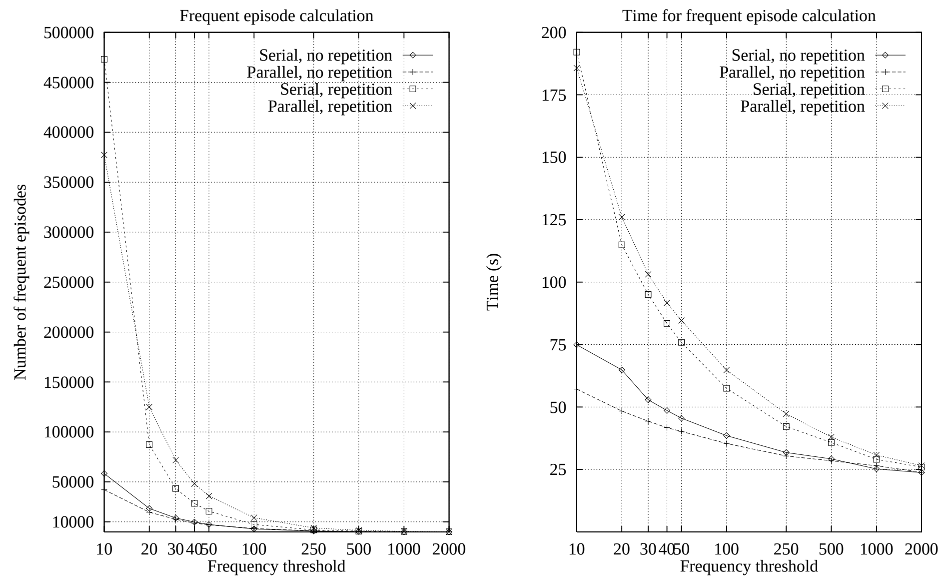


Figure 3.6: The number of frequent episodes and their calculation times from data set 4 with serial and parallel episodes, without and with repetition.

background knowledge is available, can be employed in adjusting appropriate time bounds for subsequent runs so that the episodes and episode rules capture enough relationships without exploding their amounts.

3.4.2 Experiments with WINEPI

Parallel/serial episodes and rules for all alarm data sets. Table 3.6 gives the performance of WINEPI for searching parallel episodes with window sizes 30, 60, and 120 s, and window shifts 3, 6, and 12 s, respectively. Note that the frequency thresholds vary from 0.0001 to 0.0045; the thresholds are adjusted to be as low as possible while retaining reasonable execution times. For comparison, Table A.3(a)–(b) in Appendix A presents the performance for serial episodes with window sizes 30 and 60 s, and window shifts 3 and 6 s.

Figure 3.7 illustrates the differences between serial and parallel rule generation in example data sets 3 and 4; Figures A.1 and A.2 in Appendix A illustrate the behavior of the remaining four data sets. The ratio between the number of candidates (see Figure 3.7(a)–(b)) with parallel and serial methods is about 2 favoring the serial episodes, the numbers of frequent episodes were much closer to each other (see Figure 3.7(c)–(d)). One clear exception was the data set 3 with which the ratio with both the number of candidates and frequent episodes was 7–8.

In Table A.3(c), a clear performance difference can be seen between parallel and serial episode generation: The execution times from parallel to serial increase with a factor from 2 up to 30.

3.4.3 Experiments with APRIORI

Execution times of APRIORI with the six example data sets are given in Table 3.7. With APRIORI, a notable thing is the connection between the number of alarm predicates (or actually different values of predicates) and the execution time. With data set 4, the preprocessing phase has clearly failed because even with a support threshold 0.7 the execution time is tremendous. This is a clear real-life example of the requirement which says that for the association rule algorithm the database should be quite sparse. One should have selected the relevant predicates more carefully; now there are too many predicates with too many predicate values (see Table 5.1 in Section 5.2.1).

Minepi: Serial episodes with time bounds 10 and 30 s						
#	Candidate episodes	Time (s)	Frequent episodes	Time (s)	Rules (distinct)	Time (s)
1	64141	4.09	1800	7.97	2647	1.17
2	427137	32.12	9512	51.42	23246	12.24
3	624	0.05	183	5.34	218	0.62
4	42439	6.13	8680	43.05	15228	45.16
5	83793	9.17	1898	20.92	2159	2.88
6	37068	2.40	1636	31.22	1822	2.68

(a)

Minepi: Serial episodes with time bounds 20 and 60 s						
#	Candidate episodes	Time (s)	Frequent episodes	Time (s)	Rules (distinct)	Time (s)
1	80408	9.91	3768	8.34	7632	2.46
2	441265	56.52	12643	52.10	34927	16.32
3	859	0.10	325	5.43	477	1.25
4	165895	39.88	58058	74.52	122076	310.17
5	97367	17.41	5666	24.29	8174	11.38
6	48621	4.65	3725	33.55	4890	7.22

(b)

Minepi: Serial episodes with time bounds 30 and 120 s						
#	Candidate episodes	Time (s)	Frequent episodes	Time (s)	Rules (distinct)	Time (s)
1	118624	15.53	10996	11.58	20310	7.87
2	490337	115.01	32380	56.61	75329	46.29
3	1065	0.11	515	5.44	855	1.97
4	171431	40.53	87060	129.23	172266	699.45
5	146559	35.86	23679	33.48	38035	51.49
6	79540	12.00	9588	37.77	13907	22.68

(c)

Table 3.5: Performance of the Minepi algorithm for serial episodes with time bounds 10–30 s, 20–60 s, and 30–120 s, and frequency threshold 10 (except for 40 for data set 4 in table (c)); # = data set.

Winepi: Parallel episodes with window size 30 s and shift 3 s							
#	Frequency threshold	Candidate episodes	Time (s)	Frequent episodes	Time (s)	Rules (distinct)	Time (s)
1	0.0045	6970	0.13	164	21.66	424	19.22
2	0.0005	56655	0.83	736	65.23	1916	18.97
3	0.0001	651	0.07	298	29.48	1862	19.40
4	0.0001	1589	0.14	426	119.08	2984	19.56
5	0.0005	23912	0.48	648	68.60	2462	19.13
6	0.0001	9676	0.25	560	104.76	1926	19.51

(a)

Winepi: Parallel episodes with window size 60 s and shift 6 s							
#	Frequency threshold	Candidate episodes	Time (s)	Frequent episodes	Time (s)	Rules (distinct)	Time (s)
1	0.0045	15795	0.34	659	29.19	3001	19.10
2	0.0005	175928	3.92	9682	103.45	288956	36.33
3	0.0001	4139	0.38	3585	38.34	112307	27.26
4	0.0001	4895	0.40	2639	148.46	54349	21.98
5	0.0005	105348	3.73	3666	125.93	28108	19.85
6	0.0001	21253	0.88	2618	131.22	21076	20.45

(b)

Winepi: Parallel episodes with window size 120 s and shift 12 s							
#	Frequency threshold	Candidate episodes	Time (s)	Frequent episodes	Time (s)	Rules (distinct)	Time (s)
1	0.0045	65822	2.60	11936	47.28	294350	39.20
2	0.0005	437556	21.79	56650	227.31	1775594	172.10
3	0.0001	25032	2.73	24357	62.77	1055357	159.32
4	0.0001	19382	2.12	14678	271.77	463268	54.23
5	0.0005	305373	40.63	58520	363.97	1646546	165.80
6	0.0001	64592	6.08	23268	247.60	504470	49.30

(c)

Table 3.6: Performance of the Winepi algorithm for parallel episodes; (a) window size 30 s, shift 3 s, (b) window size 60 s, shift 6 s, and (c) window size 120 s, shift 12 s; # = data set.

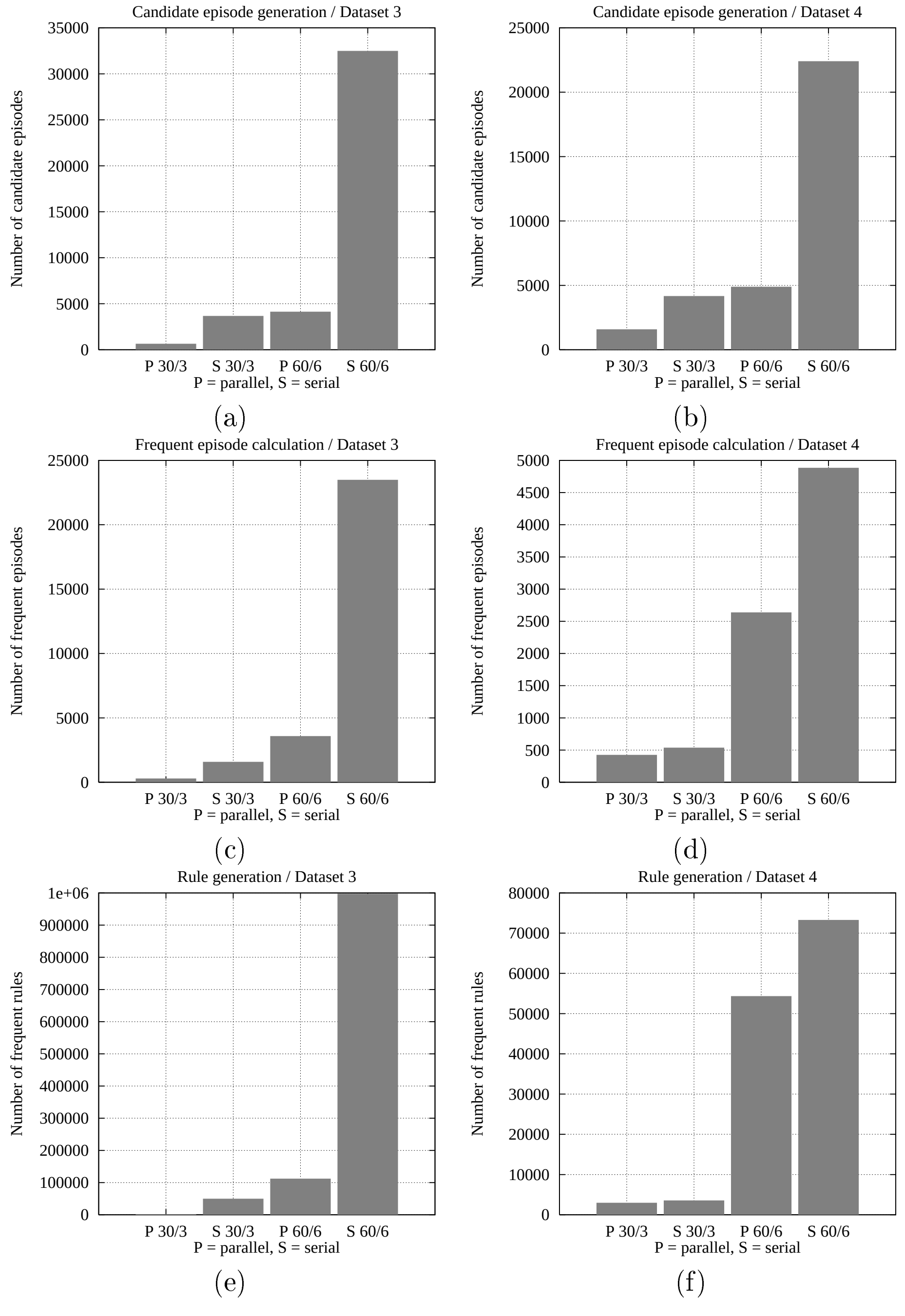


Figure 3.7: A comparison between the number of parallel and serial (a-b) candidate episodes, (c-d) frequent episodes and (e-f) rules with window sizes 30 and 60 s, and shifts 3 and 6 s. Results on the left (a,c,e) are from data set 3 and on the right (b,d,f) from data set 4.

APRIORI: PARALLEL							
#	Frequency threshold	Candidate episodes	Time (s)	Frequent episodes	Time (s)	Rules	Time (s)
1	0.05	32687	0.60	1991	136.05	55917	26.33
2	0.05	31000	0.47	271	108.39	2503	20.05
3	0.05	33034	0.67	2733	618.44	74519	27.89
4	0.70	235838	31.41	205821	757085.50	10740837	2659.03
5	0.10	109719	12.02	79311	64855.73	3750667	860.60
6	0.10	43600	1.70	13321	10478.93	509075	143.35

Table 3.7: Association rules example with alarm data.

3.5 Discussion

The methods presented in this chapter, association rules and two kinds of episode rules (WINEPI and MINEPI), have different characteristics, but they also capture different things. As we saw in Section 3.4, in the case of episode rule discovery, the MINEPI method is faster than WINEPI. To some extent, MINEPI also seems to be able to avoid the problem of the overwhelming amount of rules which occurs with WINEPI as the time window increases — thanks to the two different window sizes, or time bounds, that can be set.

It also became apparent that the execution time of the association rule algorithm APRIORI depends quite much on the number of predicates and predicate values. This emphasizes the importance of elaborate preprocessing of the data. Regardless of the method, the number of resulting rules can be quite high, although the examples given in Section 3.4 were at least partly extreme ones. It seems that the plain discovery of rules as such is not enough, if one wants to get useful results.

In spite of the problems, both association and episode rule techniques seem to work well, and they find lots of rules. They also are scalable: If the size of the data set increases (or if the sequence becomes longer), it can still be analyzed flexibly by increasing the threshold values accordingly. However, if the data set size doubles, the time needed for candidate generation remains the same and the database scanning time doubles. It should be noted that memory run-outs do not usually happen due to the size of the data set, but due to inappropriate parameter selection.

Chapter 4

Methods for Finding Interesting Patterns

One of the basic problems in data mining is to know what interests the user. In this section we consider the problem of finding interesting rules from the set of all association or episode rules holding in a data set. First we discuss the concept of interestingness in Section 4.1. In Section 4.2 we present different threshold-based measures for selecting interesting information and discuss experiments on our example alarm data sets. Then in Section 4.3, we show that it can be helpful to inquire the user for some simple additional information about the structure of the data applying *templates*, a form of pattern expressions. Finally, some additional methods including so called *rule covers* are described in Section 4.4.

4.1 Perspectives on Interestingness

As a result of the pattern discovery phase of our KDD process model, discussed in detail in Chapter 5, all patterns from a certain class are produced. The choice of the class of patterns (episode or association rules) defines in a way what patterns are considered potentially interesting. However, all patterns will not be interesting to a particular user or in a particular situation. Discovered rules can fail to be interesting for several reasons. The following are some simple examples.

- *A rule can correspond to prior knowledge or expectations.* For example, it can be known from the implementation of the telecommunication network that if an element sends an alarm A, it will also send an explanatory alarm B.

```
IF    Data Communications       IF    Data Communications
      Programming in C                Programming in C
THEN  Unix Platform                   Introduction to Unix
WITH  conf(0.14)   freq(0.03)   THEN  Unix Platform
                                WITH  conf(0.14)   freq(0.03)
```

Figure 4.1: An example of rule redundancy.

If the user knows this, the rule "A is often followed by B" will not be very interesting. On the other hand, if the user is not aware of the details of the implementation, the rule might actually be valuable and interesting.

- *A rule can refer to uninteresting attributes or attribute combinations.* E.g., a rule containing low priority alarms may be non-interesting, and the user would like to filter out all such rules.
- *Rules can be redundant.* For example, the rightmost rule in Figure 4.1 has no additional predictive power regarding the course "Unix Platform" over the leftmost rule, and might be pruned as redundant.

Klösgen [Klo95, Klo96] also defines some criteria for interestingness:

- evidence (significance of a finding measured by a statistical criterion)
- redundancy (similarity between findings)
- usefulness (does the finding meet the user's needs and goals)
- novelty (is the finding already prior knowledge)
- simplicity (syntactical complexity)
- generality (the fraction of the population a finding refers to)

Silberschatz and Tuzhilin [ST95, ST96], in turn, divide the interestingness criteria in two parts:

- objective measures: J-measure, certainty factor, strength
- subjective measures: unexpectedness and actionability

They define unexpectedness and actionability as "pattern is interesting if it is surprising to the user" and "a pattern is interesting if the user can do something with it to his or her advantage". The former can be formalized using measures like J-measure and frequency, while the latter is close to the concept of benefit by Piatetsky-Shapiro and Matheus [PSM94].

In this section, we present methods for exploring large sets of association and episode rules. The ideas of this section can be best applied on large, unstructured sets of rules and other similar, simple pattern formalisms.

For the most part, what is interesting depends on the situation at hand, and is highly dependent on the user's personal aims and perspective (see, e.g., [Klo95, PS91]). Knowledge trivial to one expert may not be trivial to another. With proper tools each expert can filter the rule collection based on his/her personal background knowledge.

Based on our experience, the following three types of operations are useful in processing a large collection of rules:

- *Pruning*: reduction of the number of rules,
- *Sorting*: ordering of rules according, e.g., to statistical significance, and
- *Structuring*: organization of the rules, e.g., to clusters or hierarchies.

Obvious pruning and sorting criteria for association and episode rules are rule confidence, frequency, and statistical significance. Rules can be pruned by setting thresholds on these properties. Examples of the effect of threshold-value based pruning with alarm data are presented in Section 4.2.

TASA offers a variety of focusing and ordering criteria for rules and supports iterative retrieval from the discovered knowledge. Network management experts can manipulate the rule set by using selection and sorting operations, as well as by using more complex operations for including or excluding certain classes of rules. Without sufficient background knowledge or proper tools, a large number of rules may satisfy simple threshold-like restrictions, such as rule frequency and confidence (see Section 4.2.1). In TASA, this problem can be alleviated by selecting rules to or removing rules from the view by *templates* [KMR$^+$94]. Although rather simple, this technique is surprisingly powerful (see Section 4.3).

The issue of interestingness of discovered knowledge has been discussed in general by Piatetsky-Shapiro [PS91]. Hoschka and Klösgen [HK91] have also used templates for defining interesting knowledge, and their ideas have strongly influenced our work. Their approach is based on few fixed statement types and partial ordering of attributes, whereas our approach is closer to regular expressions. Han et al. [HCC92] present an attribute-oriented approach for pruning uninteresting relations. Piatetsky-Shapiro and Matheus discuss the interestingness of deviations [PSM94].

4.2 Objective Measures for Interestingness

Typically, discovered rules contain two basic parts: a pattern part, here IF–THEN part, and some numerical values. In general, these values can be any measures reflecting the characteristics of the rule. Selection of potentially interesting rules can be based on these values. In the following, we consider three different objective measures: frequency, confidence, and J-measure.

4.2.1 Frequency and Confidence

The confidence and frequency thresholds ensure that the discovered rules have enough positive evidence. However, a given relation may satisfy a large number of such association or episode rules. By setting the thresholds sufficiently high, it may of course be possible to obtain a rule set with only a few rules. The problem with this method is, however, that such pruning can lose important information. In addition, rule confidence and frequency do not reveal the significance of the rule, i.e., whether the values could have been expected just by looking at the frequencies of the IF and THEN parts. Rule confidence also does not take into account the frequency of the rule right-hand side, THEN part, which can in many cases be an important factor when determining whether the rule is interesting or not. Just consider two rules with the same alarm A in the IF part, and alarms B and C in the THEN part, respectively. Suppose also, that the rule confidences are approximately the same. So, by looking only at the rule these two rules appear to be equally interesting, while in fact alarm C can be so frequent that it occurs very often and has apparently associations with the majority of all other alarms, and alarm B is much more unusual.

In Figures 4.2 and 4.3, the effect of rule frequency and confidence to the number of parallel Winepi and serial Minepi rules in the example alarm data sets are illustrated. First, in Figure 4.2 we have parallel Winepi rules with windows 30 (a&b), 60 (c&d), and 120 (e&f) seconds. Then, in Figure 4.3 we have serial Minepi rules with time bounds 10–30 (a&b), 20–60 (c&d), and 30–120 (e&f) seconds. For reference, Figure B.1(a)–(d) in Appendix B illustrates the effect of rule frequency and confidence to the number of parallel association rules with the same alarm data sets.

Clearly, the frequency threshold has a much stronger effect than the confidence threshold. With frequency, the pruning effect is usually very dramatic at certain point, so the number of rules depends heavily on the carefully selected threshold value. With confidence, the effect is similar but much smoother. This suggests that the two thresholds should be applied one after the other: frequency threshold for “brute force” pruning and confidence threshold for fine tuning.

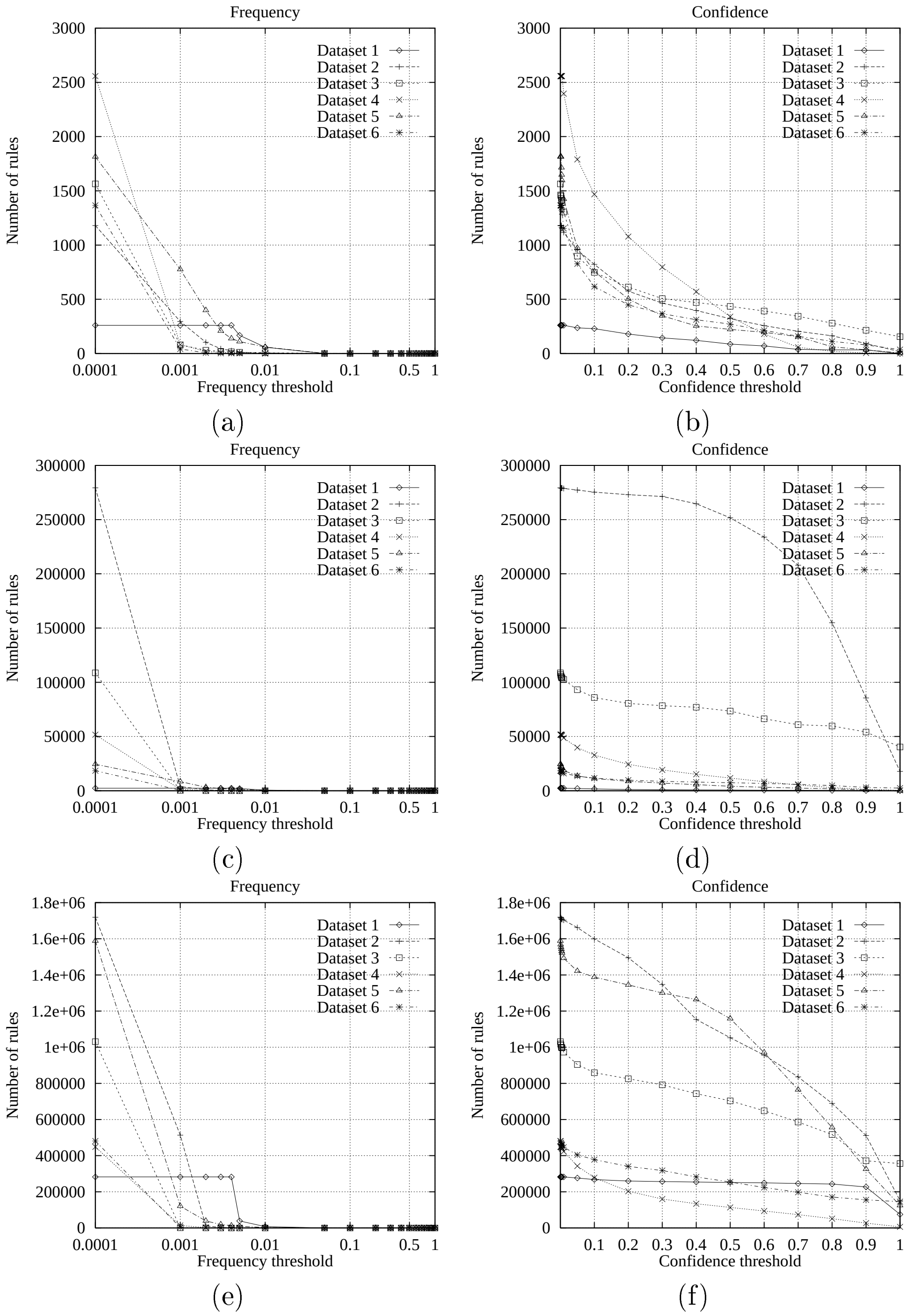


Figure 4.2: Effect of rule frequency (on the left) and confidence (on the right) thresholds to the number of parallel WINEPI rules in the example alarm data sets with window 30 s and shift 3 s (a&b), 60 s and shift 6 (c&d), and 120 s and shift 12 s (e&f).

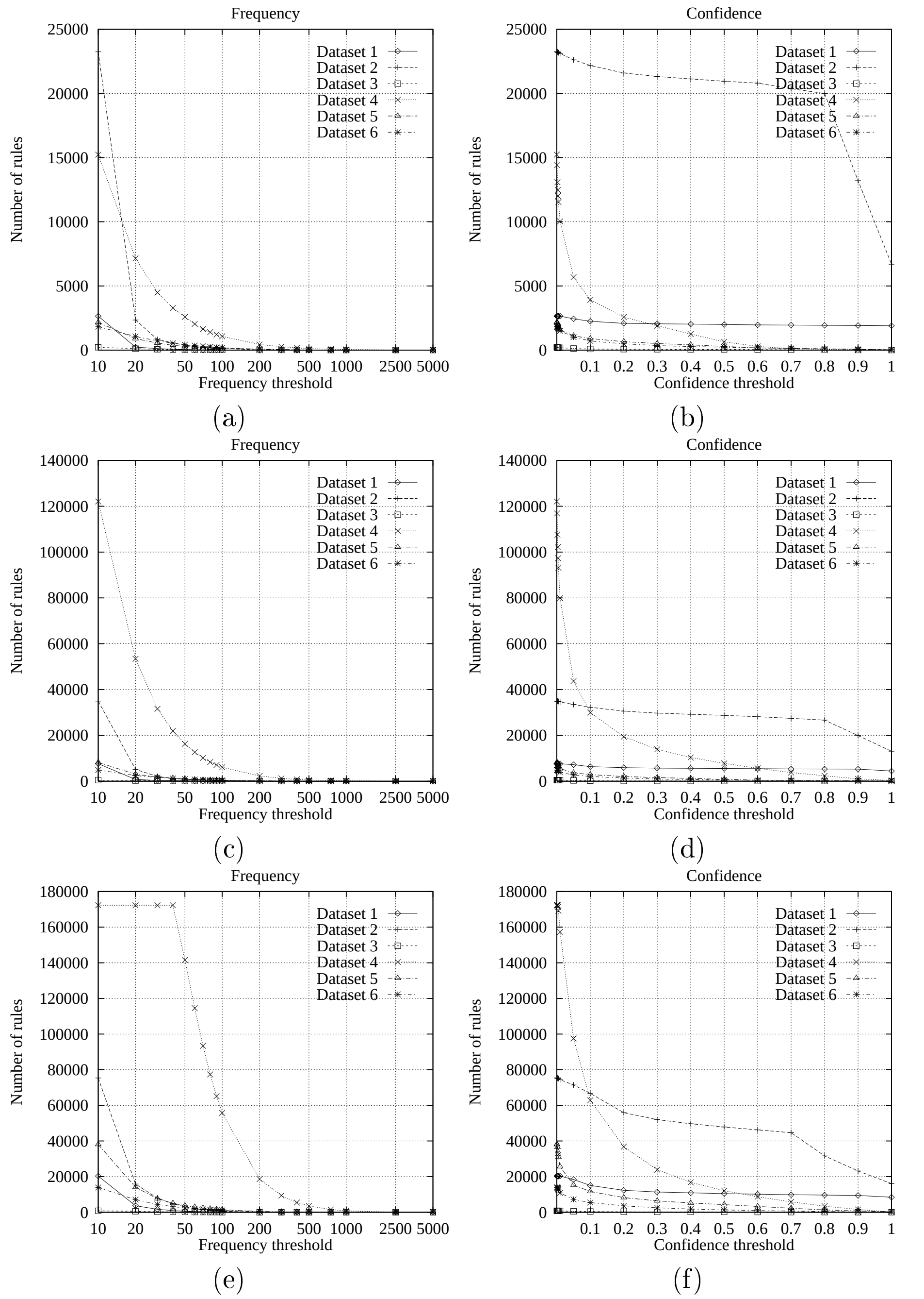


Figure 4.3: Effect of rule frequency (on the left) and confidence (on the right) thresholds to the number of serial Minepi rules in the example alarm data sets with bounds 10–30 (a&b), 20–60 (c&d), and 30–120 (e&f) seconds.

4.2.2 J-measure

There are different, more complex metrics which can be used to rank rules beside frequency and confidence values; see, e.g., [PS91] for a general overview of such methods.

With *significance* one can characterize the "randomness" of a rule. In fact, significance tells whether the strength, i.e. confidence, of the rule is expected based on the data. A rule is unexpected if its confidence differs significantly from the expected value using the frequencies of the rule left-hand side (IF part) and rule right-hand side (THEN part).

Here we use the J-measure [SG92] for rule-ranking. By replacing the probabilities with confidences, we end up in terms that are familiar from association and episode rules. This formalization is presented in [DLM+98].

$$\begin{aligned} J(A \Rightarrow B) \quad &= \quad conf(A)\Big(conf(A \Rightarrow B)\log(\frac{conf(A \Rightarrow B)}{conf(B)}) + \\ &\qquad (1 - conf(A \Rightarrow B))\log(\frac{1 - conf(A \Rightarrow B)}{1 - conf(B)})\Big) \end{aligned} \tag{4.1}$$

Here, in the context of episode rules, $conf(A)$ is the probability of symbol A occurring at a random location in the sequence (note that $conf(A) = freq(A)$), $conf(B)$ is the probability of at least one B occurring in a randomly chosen window of duration t (again, $conf(B) = freq(B)$), and $conf(A \Rightarrow B)$ is the probability of at least one B occurring in a randomly chosen window of duration t given that the window is immediately preceded by an A.

The J-measure is always between 0 and 1. The J-measure also contains two important criteria for evaluating a hypothesis, namely *simplicity* and *goodness-of-fit* between the hypothesis and the data. The simplicity comes from the first term $conf(A)$, which favors rules that occur more frequently. The other criterion, goodness-of-fit, is represented by the cross-entropy, i.e., the information gained in going from *a priori* probability, $conf(B)$, to *a posteriori* probability, $conf(A \Rightarrow B)$. From a practical viewpoint, the measure provides a useful method for ranking rules in a manner which trades-off rule frequency and rule confidence.

The J-measure avoids the problem we have with the rule confidence, i.e., that the rule confidence does not take into account the frequency of the rule right-hand side, the THEN part. Consider again the two rules with the same alarm A in the IF part, and alarms B and C in the THEN part (see Section 4.2.1). Now the J-measure of the rule with alarm B, i.e. the more interesting rule, gets higher a value than the other.

Figure 4.4 shows the effect of pruning based on the value of the J-measure on the number of parallel Winepi rules in the example alarm data sets. Furthermore in Figures 4.5, 4.6 and 4.7, the relationships between confidence and J-measure, frequency and J-measure, and confidence and frequency based on the data are given.

Again, for reference, Appendix B contains figures for association rules. Figure B.1(e)–(f) shows the effect of the J-measure threshold to the number of parallel association rules, and Figures B.2, B.3 and B.4 the relationships between the measures.

As seen from Figure 4.4, the effect of the J-measure threshold with Winepi is quite similar to the effect of the frequency threshold, but in Figure B.1(e)–(f) the J-measure threshold with association rules tends to behave like the confidence threshold. This is probably partly due to the much higher frequency values of association rules compared to Winepi rules; also the confidence values of association rules are higher and there are many rules with confidence value 1.0.

When looking at the relationships between the three measures, the clearest correlation is between frequency and J-measure. This was expected from the simplicity criteria. One can also see a similar, but fuzzier tendency between confidence and J-measure. If Figures 4.6 and 4.7 are compared, confidence seems to behave rather similarly with J-measure and frequency.

Figure 4.8 gives some simple example rules with rule confidence (Conf), frequencies for IF and THEN part episodes (F/Lhs and F/Rhs), rule frequency (F/Rule), and J-measure (J-meas.). These rules are calculated from the data set 2 with Winepi using window size 60 s and window shift 6 s. If we take, for instance, rules number 7 and 9, we can see that confidence would prefer rule number 9, while (rule) frequency would not make any difference between the two rules. On the other hand, e.g., rule number 19 is over 3 times stronger (high confidence) than rule number 7, still having the same (rule) frequency. The rule is, though, less surprising, because the frequency of rule THEN part (Rhs) is much higher than the frequency of rule IF part (Lhs). The J-measure smoothens the confidence-given difference, and rules number 7 and 19 have almost the same J-measure value.

As we noticed in Section 4.2.1, it is beneficial to use frequency and confidence together. Now we have one more measure to use. First we could, e.g., set appropriate frequency and confidence thresholds to express our requirements for strength and generality, and then rank the rules with J-measure value.

But does J-measure really give us a chance to get better quality rules? Based on the experiments and the J-measure characteristics, the answer is in most cases yes — although the impact is not always very dramatical. Also, in the end everything depends on what rule characteristics we value.

4.3 Rule Selection with Templates

Templates are pattern expressions that describe the form of rules that are to be selected or rejected [KMR⁺94]. They are one way of focusing a view to a large space of rules. A template is an expression $A_1, \ldots, A_k \Rightarrow A_{k+1}, \ldots, A_l$, where each A_i is either an attribute name, a class name, or an expression $C+$ or $C*$, where C is a class name. Here $C+$ and $C*$ correspond to one or more and zero or more instances of the class C, respectively; a class could be, e.g., "basic course", "intermediate course", or "advanced course" in Figure 4.10. A rule $B_1, \ldots, B_m \Rightarrow B_{m+1}, \ldots, B_n$ matches a template if the rule can be considered to be an instance of the pattern. The order of the components in the templates has no significance if the attribute order in the rules is not significant (association rules and parallel episode rules). With templates, the user can explicitly specify both what is interesting and what is not. To be interesting, a rule has to match a *selective template.* If a rule, however, matches an *unselective template*[1], it is considered uninteresting. To be presented to the user, a rule must be considered interesting, i.e. match one of the selective templates, and it must not be uninteresting, i.e. not match any of the unselective templates.

In TASA, the template concept is implemented following the above definitions. The user interface of TASA to the template mechanism is shown in Figure 4.9. In case of episode rules, the user can define the alarms that should or should not occur in the rule, and also the bounds for, e.g., frequency, confidence, and number of alarms in one rule (see Example 4.1).

Example 4.1 The selective template in Figure 4.9 matches rules that have

- alarm 2970_31650 and possibly also other alarms in the IF part,
- alarm 7161_31650 in the THEN part,
- confidence of at least 0.8 but no more than 0.95, IF part frequency of at least 5, rule frequency of at least 2, and
- at least 2 but no more than 3 items in the IF part, no more than 2 items in the THEN part.

Furthermore, the user is about to select "IF" and "Conf" for sorting criteria for the resulting rules. Applying these criteria would mean sorting

[1] "Selective template" and "unselective template" were in [KMR⁺94] referred to as "inclusive template" and "restrictive template", respectively.

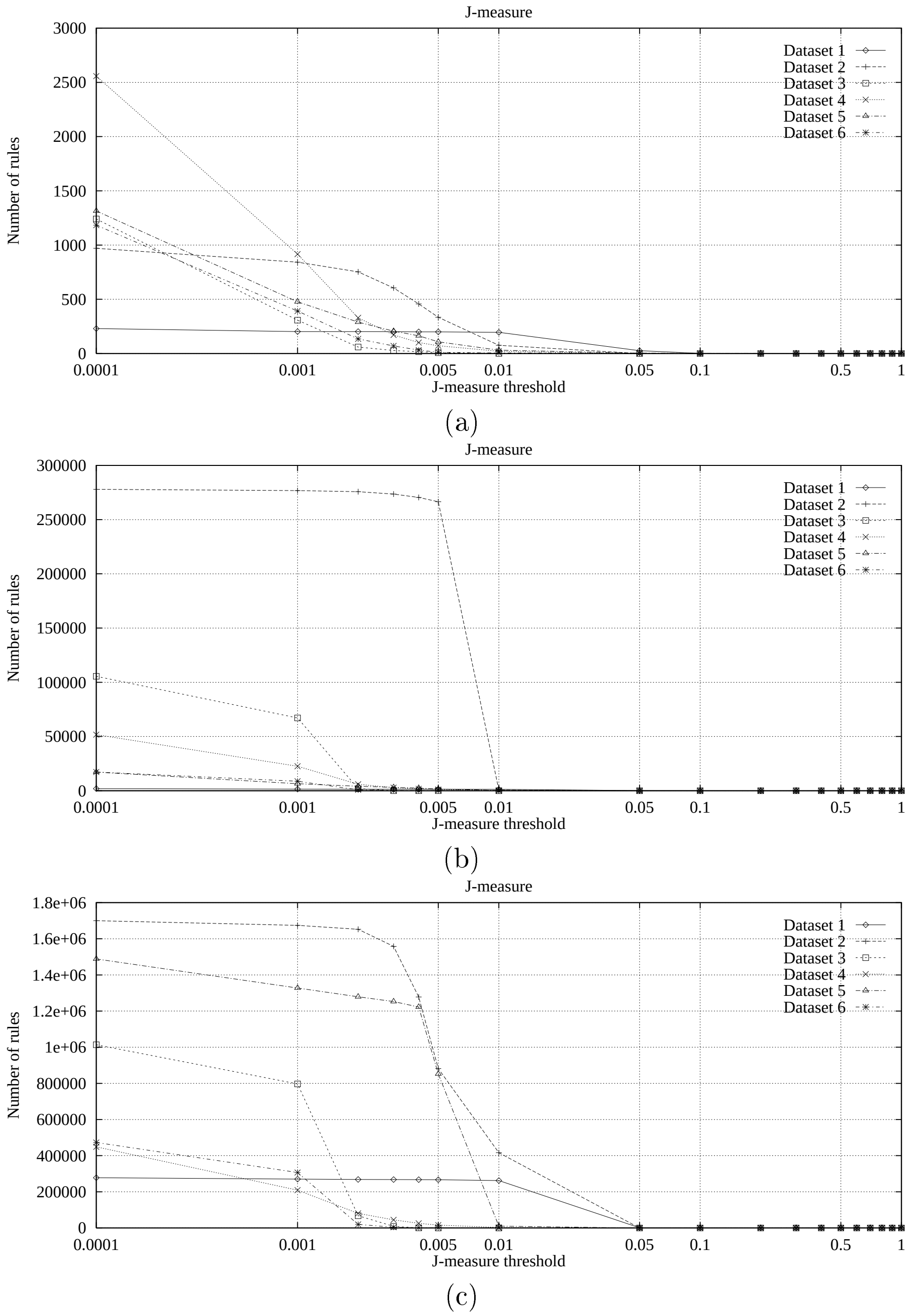


Figure 4.4: Effect of J-measure thresholds to the number of parallel Winepi rules in the six example alarm data sets with (a) window size 30 s and shift 3 s, (b) window size 60 s and shift 6 s, and (c) window size 120 s and shift 12 s.

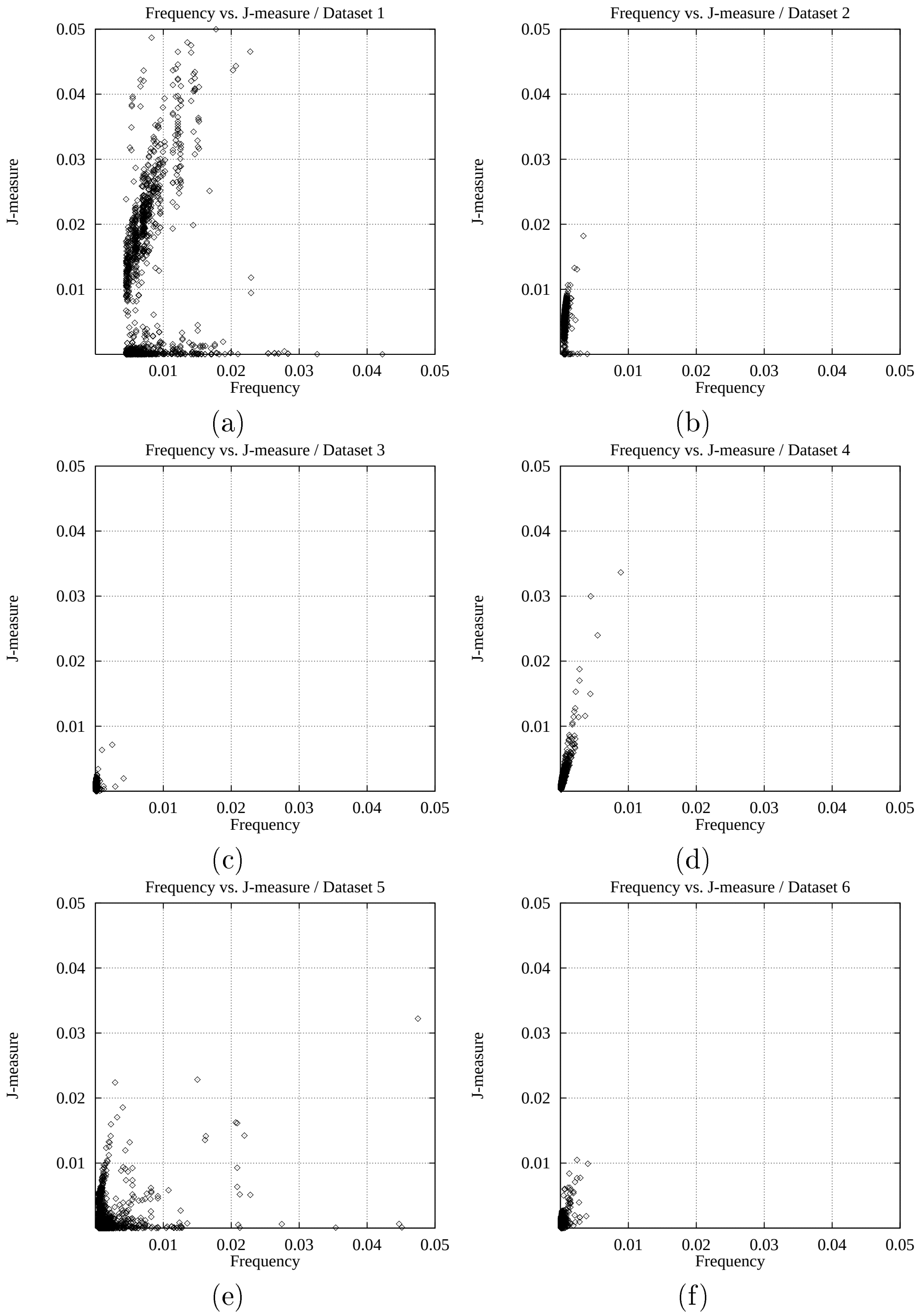


Figure 4.5: A comparison between frequency and J-measure in parallel WINEPI rules from alarm data sets 1–6 with window size 60 s and shift 6 s.

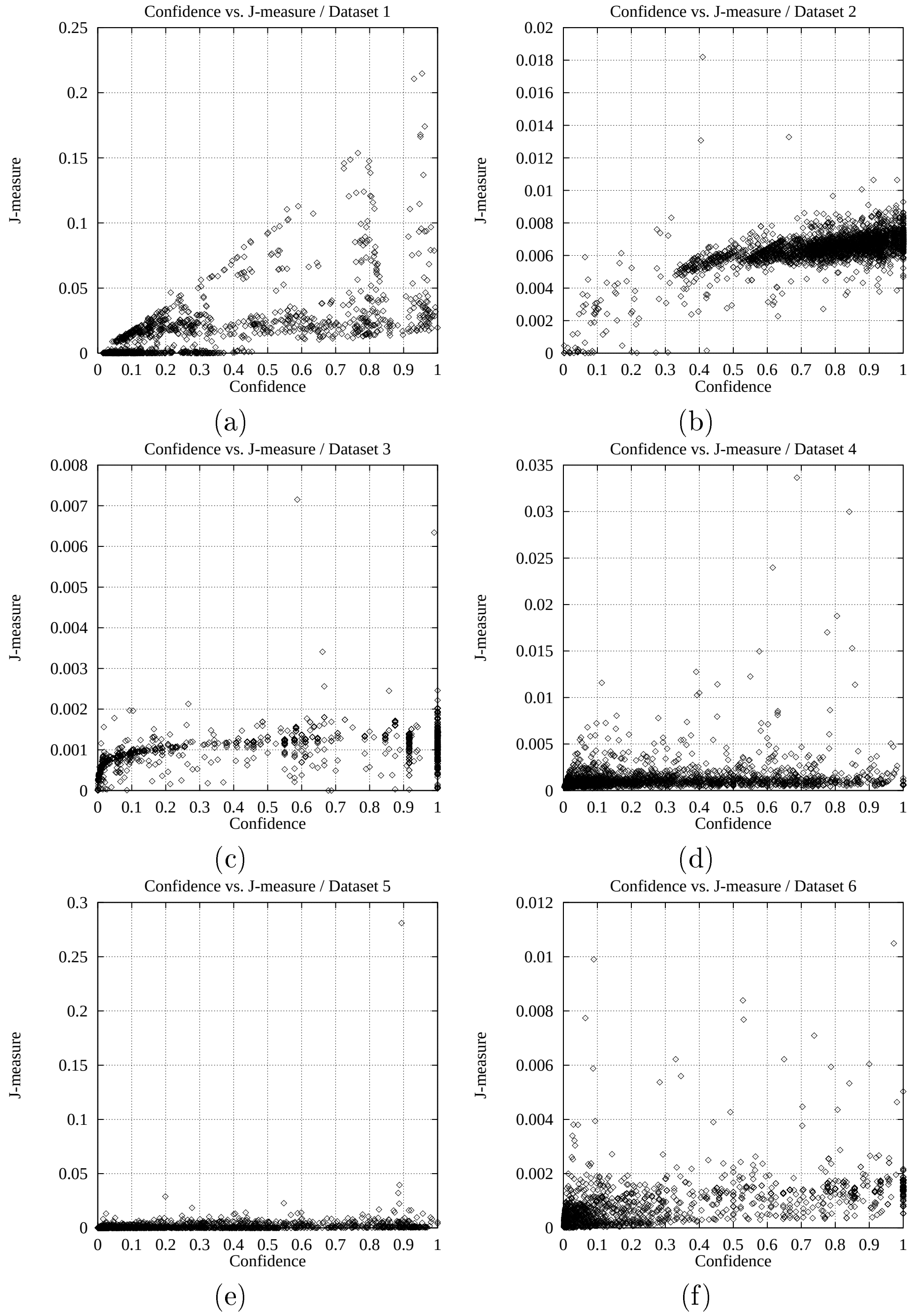


Figure 4.6: A comparison between confidence and J-measure in parallel Winepi rules from alarm data sets 1–6 with window size 60 s and shift 6 s.

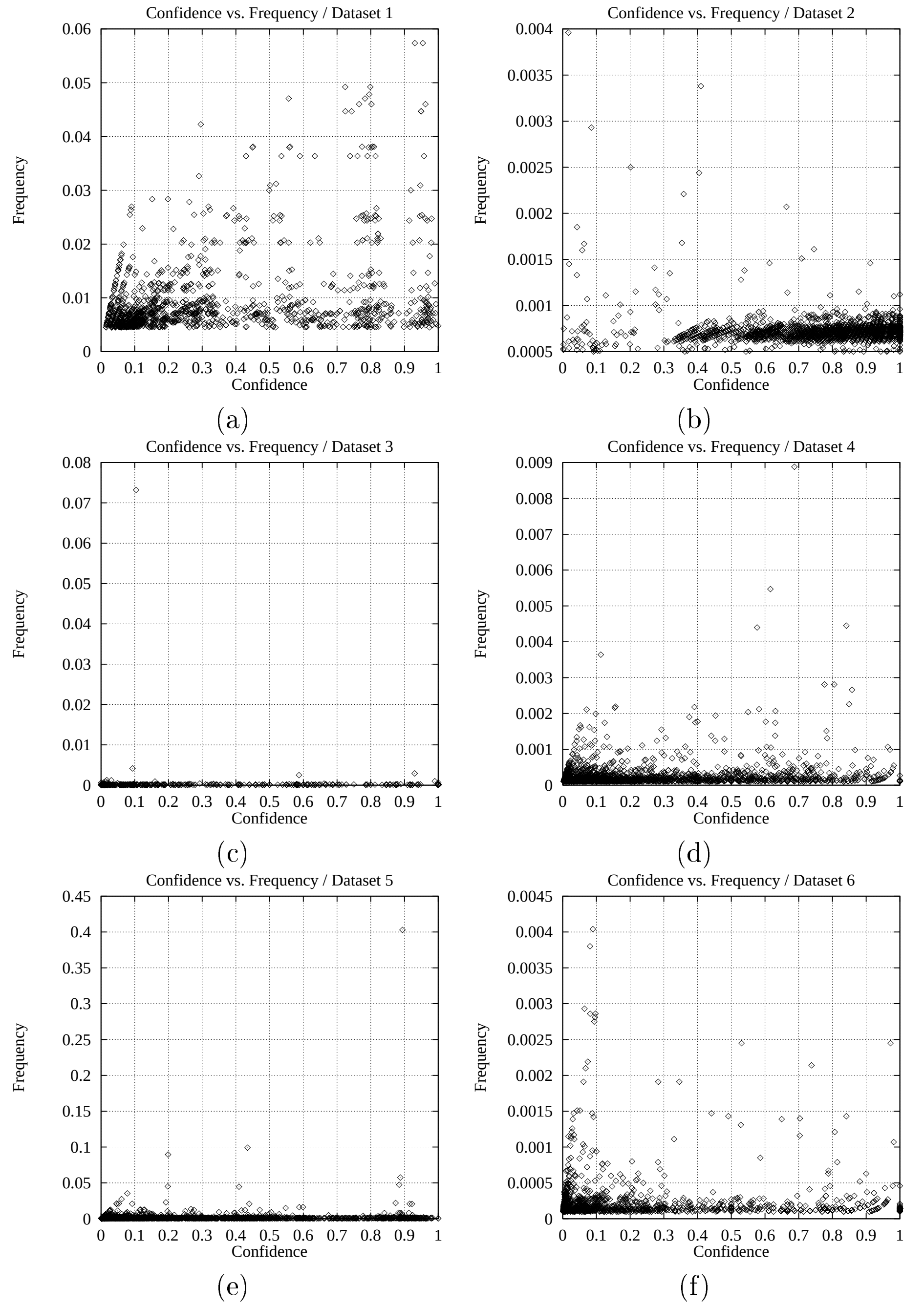


Figure 4.7: A comparison between confidence and frequency in parallel WINEPI rules from alarm data sets 1–6 with window size 60 s and shift 6 s.

Row	Lhs		Rhs	Conf	F/Lhs	F/Rhs	F/Rule	J-meas.
1	4	=>	2	0.69296	0.00938	0.01721	0.00650	0.02982
2	2	=>	4	0.37769	0.01721	0.00938	0.00650	0.02747
3	6	=>	2	0.33760	0.01798	0.01721	0.00607	0.01929
4	2	=>	6	0.35270	0.01721	0.01798	0.00607	0.01937
5	8	=>	2	0.01524	0.11023	0.01721	0.00168	0.00002
6	2	=>	8	0.09762	0.01721	0.11023	0.00168	0.00002
...								
7	363	=>	203 204 362	0.28510	0.00463	0.00146	0.00132	0.00845
8	362	=>	203 204 363	0.24490	0.00539	0.00138	0.00132	0.00822
9	362 363	=>	203 204	0.38040	0.00347	0.00491	0.00132	0.00681
10	204	=>	203 362 363	0.22260	0.00593	0.00134	0.00132	0.00807
11	204 363	=>	203 362	0.81988	0.00161	0.00152	0.00132	0.01126
12	204 362	=>	203 363	0.82500	0.00160	0.00140	0.00132	0.01144
13	204 362 363	=>	203	0.90411	0.00146	0.00543	0.00132	0.00927
14	203	=>	204 362 363	0.24309	0.00543	0.00146	0.00132	0.00810
15	203 363	=>	204 362	0.94286	0.00140	0.00160	0.00132	0.01182
16	203 362	=>	204 363	0.86842	0.00152	0.00161	0.00132	0.01139
17	203 362 363	=>	204	0.98507	0.00134	0.00593	0.00132	0.00962
18	203 204	=>	362 363	0.26884	0.00491	0.00347	0.00132	0.00668
19	203 204 363	=>	362	0.95652	0.00138	0.00539	0.00132	0.00959
20	203 204 362	=>	363	0.90411	0.00146	0.00463	0.00132	0.00957

Figure 4.8: Examples of parallel WINEPI rules from data set 2 with window size 60 s and window shift 6 s. Simple rules with events 2, 4, 6, and 8, and rules in which exactly the events (203,204,362,363) occur. Here, "Lhs" stands for rule left-hand side (IF part), "Rhs" for rule right-hand side (THEN part), and "F" for frequency.

the rules based on the contents of the rule IF part and the confidence value in ascending order. □

Example 4.2 In our example data set of computer science course enrollment, the courses are divided into three classes: basic, intermediate, and advanced courses. In addition, all courses belong to the parent class 'any course'. Thus, we have such generalizations as in Figure 4.10. Such a high level classification of attribute types can be useful for selecting certain types of rules.

Using the selective template (a) in Figure 4.11 the user obtains a number of rules with (at least) an advanced course in the IF part and the course "Design and Analysis of Algorithms" in the THEN part. Examples of such rules are (b), (c) and (d). Of the 2 010 rules in the example database there are 15 rules that match the above specifications and have confidence of 0.2

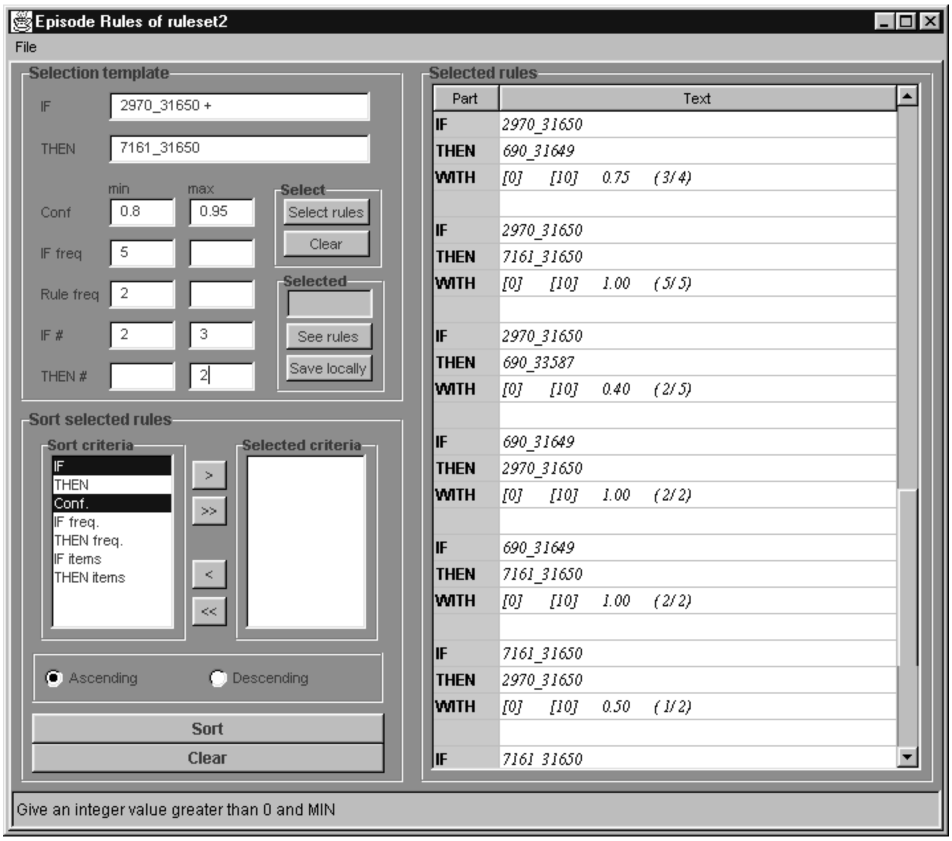


Figure 4.9: Rule Viewing window of the new TASA system containing a template skeleton with additional functionalities.

or more and frequency of 0.01 or more. This sort of rules give an idea what the students do in addition to taking "Design and Analysis of Algorithms", and the discovered rules can be utilized when planning the course. Note that here the IF–THEN structure implies no temporal order.

The user may consider some of the resulting rules uninteresting, such as (b) including basic courses. By adding an unselective template with "Basic Course" and "Any Course∗" in the IF part and "Any Course∗" in the THEN part the user can filter out all rules with a basic course in the IF part. □

This technique is surprisingly powerful. For example, background knowledge about the normal curricula of computer science studies can be taken into account by using templates that unselect all rules that correspond to the normal course of studies.

Example 4.3 As an example of how the template concept can be used in case of alarm data, consider the following scenario. Both the device name and the situation are fictional, but the numbers reflect actual results and the effect of methods used in the exploration of the discovered alarm rule base.

Assume the network manager has used TASA to discover ordered episode rules for the past 10 days using a maximum time bound of 2 minutes. The manager knows that during that one-and-half week period, a device *AA–16* caused serious problems. So, he/she wants to find some explanation of that phenomenon. First the manager selects all rules that have *AA–16* in the THEN part, thus wanting to trace some alarms that potentially have some causal relationship with the alarm type in consideration.

The number of such rules is, however, large: 1 848. The network manager decides to restrict the THEN part to contain only one alarm. The number of selected rules is still large, 901, so he/she uses an unselective template to remove all rules with alarms from the same network area (indicated by the first two characters in the device identificator, here *AA*), which he/she knows by experience not to have any connection to each other.

The number of rules is still large, 240 rules, so the manager further tightens the scope and uses a selective template to get rules with at most two alarms in IF part and confidence of at least 75%. The resulting rule set contains 19 rules from the 5 498 original ones. By browsing the resulting rules he/she finally finds alarms which indicate that the problems with device *AA–16* had been caused by high fault rate in a similar nearby device. □

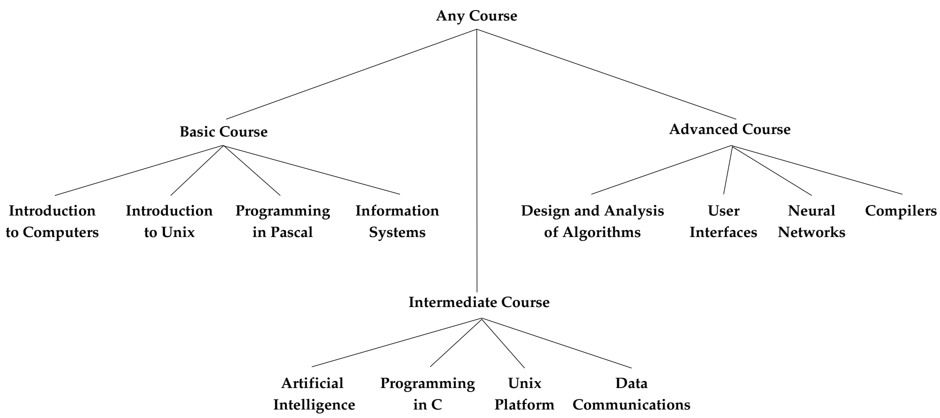


Figure 4.10: Course hierarchy.

```
IF     Advanced Course            IF     Compilers
       Any Course*                       Introduction to Computers
THEN   D & A of Algorithms        THEN   D & A of Algorithms
                                  WITH   conf(0.16)   freq(0.01)
             (a)                                (b)

IF     Neural Networks            IF     User Interfaces
       Data Communications               Information Systems
THEN   D & A of Algorithms        THEN   D & A of Algorithms
WITH   conf(0.33)   freq(0.01)    WITH   conf(0.02)   freq(0.10)
             (c)                                (d)
```

Figure 4.11: A template (a) and rules matching it (b, c and d).

4.4 Redundancy Elimination, Clustering and Visualization

The collection of all association rules that holds in a given relation provides a lot of information about the data set. The problem with this approach and also the use of episode rules is that the collection of all rules can be very large, especially when one is interested also in the behavior of small subgroups of the original data set. Sometimes lots of rules remain, even when the user has found the desired focus with the methods described in the previous sections.

Although the template mechanism is useful in many ways, also automatic pruning, sorting, and structuring methods should at this point be available for invocation by the user, especially for removal of redundancy. In Section 4.4.1 we present rule covers, a domain independent method of reducing the number of rules by elimination of redundancy. We also shortly discuss grouping of association rules by clustering (Section 4.4.2) and the use of visualization to get a general picture of the rule set (Section 4.4.3).

4.4.1 Rule Covers

A *rule cover* [TKR+95] is a subset of the original set of rules such that for each row in the relation there is an applicable rule in the cover if and only if there is an applicable rule in the original set.

Consider a collection Γ_Y of association rules with the same attribute set Y as the consequent:

$$\Gamma_Y = \{X_i \Rightarrow Y \mid i = 1, \ldots, n\}.$$

Every rule in Γ_Y gives a description of the presence of the attributes Y. Discovered rule sets often are heavily redundant in a sense that several rules describe the same database rows.

Given a rule set Γ_Y, its subset $\Delta \subseteq \Gamma_Y$ is a *rule cover*,[2] if

$$\bigcup_{(X \Rightarrow Y) \in \Gamma_Y} m(XY) = \bigcup_{(X \Rightarrow Y) \in \Delta} m(XY),$$

where $m(XY)$ is the set of rows matched by XY, i.e., rows where the rule $X \Rightarrow Y$ holds. In other words, the rules in a rule cover Δ describe Y in all the cases in the database that the original rule set Γ_Y does. (See e.g. [MMHL86a, MMHL86b, CN89] for similar notions in the machine learning literature.)

It is useful to constrain the rule sets and their covers to contain rules with high confidence only, for two reasons. First, for justification and prediction of the consequent Y, mostly rules of a high confidence are of interest. Second, the rules in a rule set and in its cover are then known to have similar confidences, and not to make contradictory justification and predictions. These two statements are true also with rules that have a high J-measure.

If one wants to be sure that the pruning methods do not lose any information, one has to assume the following. The data set should be monotonic in the sense that if there is a matching rule for Y with a high confidence, there should not exist a more special matching rule with a lower confidence. These restrictions seem to hold in many real world data sets.

In our experiments, rule covers turned out to produce useful short descriptions of large sets of rules. Particularly useful are structural rule covers which can be computed very efficiently. In the following we give the basic definitions and examples of both structural and rule covers.

Structural Rule Cover

We first describe a conservative but efficient way of pruning, based on the structure of the rules. The method is based on the observation that for all attribute sets X, Y, and Z we have

$$m(XYZ) \subseteq m(XZ),$$

i.e., a rule $X \Rightarrow Z$ matches a superset of the database rows matched by the rule $XY \Rightarrow Z$. Thus if Δ is a cover with $X \Rightarrow Z \in \Delta$ and $XY \Rightarrow Z \in \Delta$, then $\Delta \setminus XY \Rightarrow Z$ is also a rule cover.

[2] The usage of the term *cover* is borrowed from database theory, where it is extensively used (see [Ull88]); there is also a close similarity to the set cover problem.

A set of rules $\Delta \subseteq \Gamma_Y$ is a *structural rule cover* for Γ_Y, if for all rules $(X \Rightarrow Y) \in \Delta$ there is no rule $(X' \Rightarrow Y) \in \Gamma_Y$ such that $X' \subset X$. A structural cover thus contains the most general rules of the original rule set, i.e., exactly those rules whose antecedents do not include the antecedent of any other rule. Note that a structural cover can be completely computed without looking at the rows of the relation; computation is straightforward.

Example 4.4 The latter of the following rules has no additional predictive power over the first one, and could be pruned as redundant:

> Programming in C, Object Data Bases $\Rightarrow$ Data Communications (0.90, 0.02)
> Programming in C, Object Data Bases, Computer-Supported Cooperative Work $\Rightarrow$ Data Communications (0.90, 0.01)

□

Structural covers are an efficient way of pruning rule sets. Next we describe an algorithm that finds more optimal rule covers. The algorithm can be used to improve structural covers.

Algorithm for Finding Rule Covers

A close to optimal rule cover can be found with the greedy algorithm in Figure 4.12. By optimal we mean that the number of rules is as small as possible and all the rows are covered.

The algorithm gets as input the original rule set Γ_Y and the sets of rows matched by each of these rules; the matched sets can be discovered by one pass over the database in a straightforward manner. Rule cover Δ is initialized to an empty set. Variable s' is used to store those database rows that are not matched by the rules in Δ; the sets s_i contain those rows in s' that are matched by the rule $X_i \Rightarrow Y$. Iteratively, the rule in Γ_Y that matches most of the rows in s' is moved from the rule set Γ_Y to the rule cover. The rows matched by this rule are removed from s'. This is repeated until all the rows matched by the original rule set are matched by the rule cover, i.e., until the set s' is empty.

The covers computed by the above algorithm seem to produce useful short descriptions of large sets of rules.

Example 4.5 From the alarm data set 5 the association rule algorithm discovered 28 843 rules (with support threshold 0.02 and confidence threshold 0).

Algorithm RuleCover
Input: Set of rules $\Gamma_Y = \{X_i \Rightarrow Y \mid i = 1, \ldots, n\}$.
Sets of matched rows $m(X_iY)$ for all $i \in \{1, \ldots, n\}$.
Output: Rule cover Δ.
Method:

```
Δ := ∅;                              // rule cover
s' := ⋃_{i=1}^n m(X_i Y);            // rows unmatched by cover
for all i ∈ {1,...,n} do
    s_i = m(X_i Y);                  // rows of s' matched by rule i
end;
while s' ≠ ∅ do
    choose i ∈ {1,...,n} so that (X_i ⇒ Y) ∈ Γ_Y and |s_i| is largest;
    Δ := Δ ∪ {X_i ⇒ Y};              // add the rule to cover
    Γ_Y := Γ_Y \ (X_i ⇒ Y);          // remove the rule from original set
    for all (X_j ⇒ Y) ∈ Γ_Y do
        s_j = s_j \ m(X_i Y);        // remove matched rows
    end;
    s' := s' \ m(X_i Y);             // remove matched rows
end;
```

Figure 4.12: Algorithm RuleCover.

Using templates, we selected a subset of 1 461 rules with the attribute "deviceType:TR" (transmission device) as the consequent, i.e. in the THEN part, and with a confidence of at least 0.90. The structural cover of the subset consisted of 20 rules, which is over 98% improvement. A further rule cover computed by Algorithm RuleCover from the 20 rules only contains these 5 rules:

networkElement:NL $\Rightarrow$ deviceType:TR (1.00, 0.33)
networkElement:NR $\Rightarrow$ deviceType:TR (1.00, 0.08)
alarmText:PowerSupplyUnbalance $\Rightarrow$ deviceType:TR (1.00, 0.40)
alarmText:ErrorRate $> 10^{-5}$ $\Rightarrow$ deviceType:TR (1.00, 0.04)
priority:Low $\Rightarrow$ deviceType:TR (1.00, 0.25)

These rules tell that network elements NL and NR are transmission devices, and that only alarms sent by transmission devices contain certain alarm texts and have low priorities. □

This greedy algorithm computes a rule cover Δ which matches exactly the same rows as the original rule set Γ_Y. It can be useful to loosen this restriction by allowing that some small portion ε of the rows is not matched by the rule cover Δ.

4.4.2 Rule Grouping with Frequent Sets

Given an attribute set Y, the cover for the set of rules of the form $X \Rightarrow Y$ can still be so large and fragmented that the user cannot get a clear picture of the behavior of the rule consequent. The cover can be made more understandable by ordering and grouping of rules.

Structure can be added to the set of rules by ordering them by their attributes. Rules can be clustered so that those making statements about the same attributes are close together in the representation. Such structural clustering may, however, fail to reflect the data set characteristics.

A more useful clustering method is to group those rules together that make statements about the same database rows. This way, also the way the grouping is done describes the database.

We have implemented such a method. In the experiments we were able to find interesting and useful rule clusters from rule covers. The distance measures of rules and parameter settings of the clustering algorithms are subject to further investigation.

Distance Measures

For clustering, the distance between two association rules can be defined in various ways. Three possibilities are given in the following. The distance of a rule $Y \Rightarrow Z$ from another rule $X \Rightarrow Z$ can be defined as the number of rows that are matched by $X \Rightarrow Z$ but not by $Y \Rightarrow Z$:

$$\begin{aligned} d_1(X \Rightarrow Z, Y \Rightarrow Z) &= |m(XZ) \setminus m(YZ)| \\ &= |m(XZ)| - |m(XYZ)|. \end{aligned}$$

The minimum of the above asymmetric distances between two rules is a symmetric distance:

$$d_2(X \Rightarrow Z, Y \Rightarrow Z) = min(|m(XZ)|, |m(YZ)|) - |m(XYZ)|.$$

A metric is obtained by summing the asymmetric distances between two rules:

$$d_3(X \Rightarrow Z, Y \Rightarrow Z) = |m(XZ)| + |m(YZ)| - 2|m(XYZ)|.$$

Note that the number of matching rows $|m(X)|$ has already been computed for all frequent attribute sets X while forming association rules, and the distances are thus fast to compute.

Clustering Methods

The clusters are formed from the structural cover of an attribute. However, in the case of a large database the clusters can still be quite large. The number of rules in them can be reduced further by calculating the rule cover for each cluster. By calculating first the structural cover and then clustering the selected rules, the size of input sets for the rule cover algorithm decreases to a fraction of what it would have been otherwise. This speeds up the automatic rule pruning and grouping process.

Example 4.6 From the 2 010 rules discovered from our example computer science course enrollment database, we selected those 52 rules that have the course "Data Communications" as the consequent, i.e. in the THEN part, and a confidence of at least 0.90. The structural cover has 29 rules, with 15 different attributes. A cover computed by Algorithm RuleCover consists of 20 rules.

We clustered the 29 rules of the structural cover using SAS, a statistical analysis program. We used a metric with definition d_3 for distance and a non-parametric density estimation method similar to the classification or assignment stage of algorithms described by Gitman [Git73] and Huizinga [Hui78].

With different parameter values, two solutions were obtained: one with 4 clusters, another with 3. From those two alternatives, the one with 3 clusters turned out to be better in its ability to break the rule set into clearly separate clusters. For instance, only cluster 1 contains rule left-hand sides mentioning courses "VAX/VMS" (column C1) and "Distributed Operating Systems" (column C8) (see Figure 4.13). Students matched by rules in the cluster clearly are those specializing in operating systems. Cluster 2, in turn, contains rules describing students specializing in information systems: frequent or distinctive attributes are "Object Data Bases" (column C11), "Database Systems II" (column C9), and "Computer Graphics" (column C5). Cluster 3 does not match so clearly with any particular curriculum in the department, and is therefore somewhat surprising. The distinctive courses are "Artificial Intelligence" (column C4), "Database Systems II" (column C9), and "Programming in C" (column C12). □

Rule clusters seem to be very useful. For instance, in the simple case of Example 4.6 clustering gives interesting information about courses and

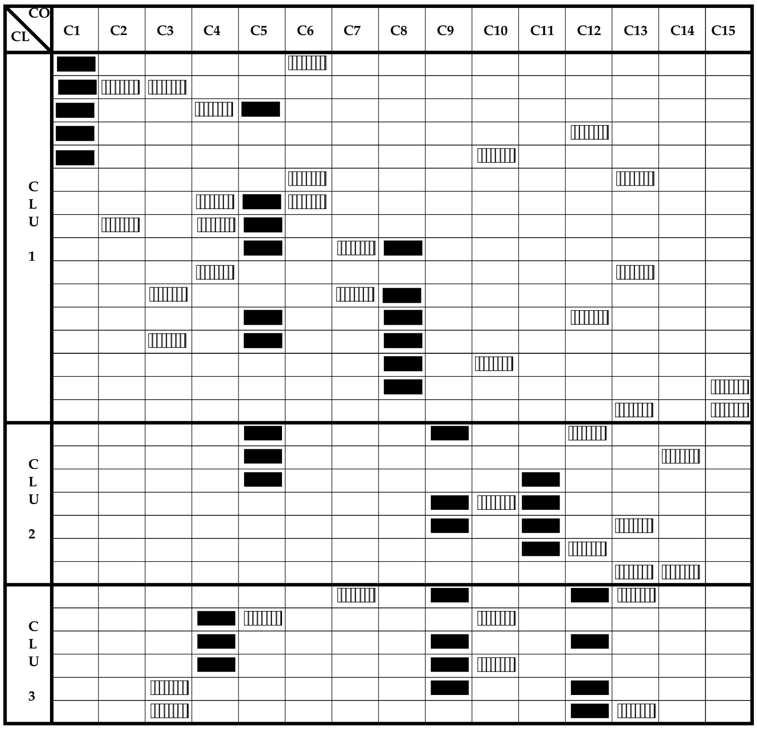


Figure 4.13: Clustering chart. Columns represent attributes, i.e. courses, and rows represent antecedents of rules. For example, the first row corresponds to a rule "VAX/VMS, Computer Uses in Education ⇒ Data Communications". (The most distinctive courses have been emphasized manually.)

students. Firstly, clusters correspond to intuitive groups of students and the clustering gives correct information about the distribution of courses. Certain courses, such as "VAX/VMS" and "Distributed Operating Systems", are often taken by the same students. Even though there was no rule in the cover with both of these courses, they ended up in the same cluster.

4.4.3 Visualization of the Rules and Rule Sets

Even with reasonable thresholds, templates, and grouping, a certain number of rules remain. This number is usually small or at most moderate, but just glancing at a list of rules seldom gives a useful overview of them. So, there is a clear need for visualizing rules and rule sets.

For single rules we can for example show confidence, frequency, and J-measure in a graphical form. In Figure 4.14 there is an example of a

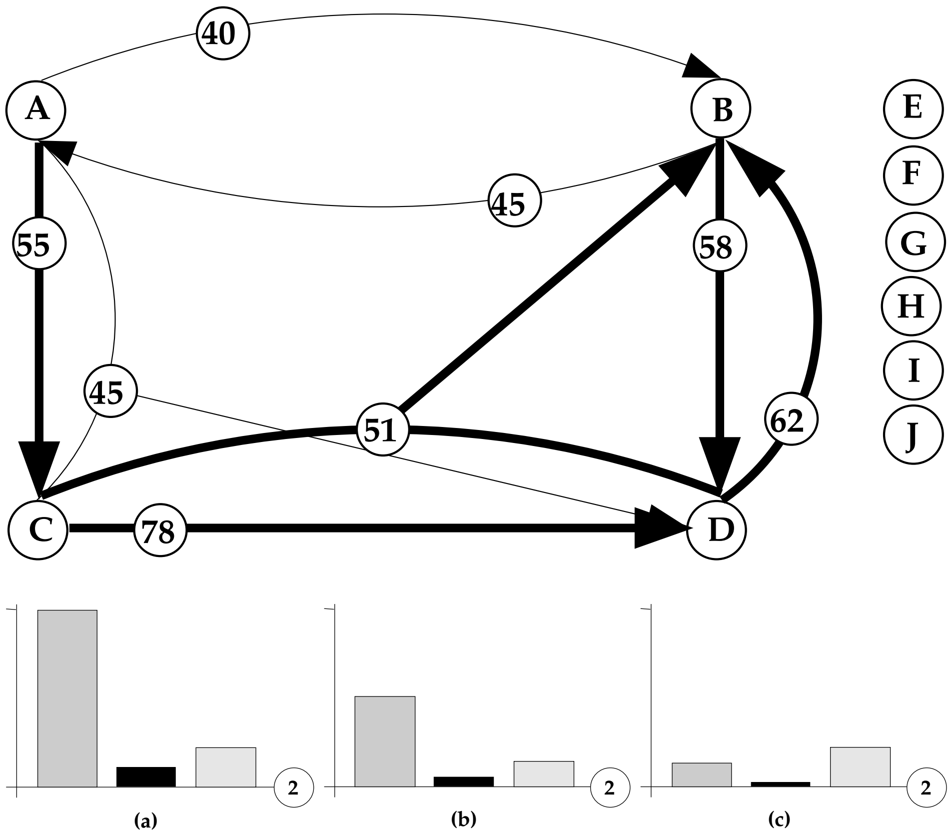


Figure 4.14: A weighted hypergraph for visualizing a set of rules (top) and example bar graphs for visualizing single rules (bottom). In the bar graphs, the leftmost and rightmost bars represent the confidence and the frequency of the rule, respectively, while the middle bar represents the J-measure. The number of attributes in the rule is indicated in the circle by the bar graph.

bar graph presentation. The leftmost and rightmost bars represent the confidence and frequency of the rule, respectively, while the middle bar represents the J-measure. The number of attributes in the rule is indicated in the circle by the bar graph. Another way to visualize rule significance is to create rectangles in which one side represents the confidence and the other side the frequency of a rule. Also the Kiviat graphs [Kol73, KK73] could be useful for rule visualization. Often, it is efficient to combine the textual and graphical representations of the rules. The user can first select interesting rules using their graphical patterns — which is a fast way to pick up exceptional cases — and then refine the selection in textual mode.

Visualizing a set of rules is much harder than showing a single rule. In the following we focus on visualizing association rules, which was our primary visualization task at first. Basically, the problem is to visualize a hypergraph, where an edge consists of all the attributes appearing in a rule, and each edge has a distinguished vertex (the attribute of the right-hand side, i.e. the THEN part) and two weights (frequency and confidence). The wide research in drawing and visualizing ordinary graphs shows that even that problem is by no means easy (see, e.g., [PT90, GNV88, RDM+87]); thus for the much harder problem of visualizing weighted hypergraphs we must be content with fairly modest solutions. The basic idea of the *attribute graph model* is to represent attributes as nodes and associations as directed arcs (see Figure 4.14). The thickness of an arc represents the confidence or frequency of the corresponding rule. Colors can be used for visualizing additional information.

Node placement is a hard problem. Irrespective of the algorithm, even with a relatively small amount of attributes and association rules, the graph easily becomes very dense and hard to understand. A simple but very powerful and useful way to limit the complexity is to let the user limit the rule sizes. For instance, the user may choose to view only rules with at most two attributes in the IF part, the left-hand side of the rule. This is closely connected to the use of templates and related concepts.

Similarly, deleting a node from a graph equals to entering two unselective templates: one with the attribute in the IF part and another with the attribute in the THEN part. In Figure 4.14, attributes E through J have been removed. Similarly, nodes can be marked interesting; this corresponds to the addition of two selective templates. Joining nodes is a third way to reduce complexity. The user can join nodes — typically a strongly connected cluster of them — to be represented by a single node of combined effect. If a node is joined with another node it is not obvious which of the connections of the joint node are strong enough to be visualized.

For visualization of episode rules, the TASA system offers some simple facilities. E.g., an interesting picture of the interaction between the IF and THEN part of an episode rule can be obtained by drawing a histogram showing the distance from each occurrence of the IF part episode to the nearest occurrence of the THEN part episode. Such histograms are valuable guides for locating possible periodic relationships between the IF and THEN parts, as is demonstrated by Figure 4.15. Note that the episode rule formalisms do not fully capture the details of such relationships. While in TASA the visualization aspects are not heavily supported, much more — such as visual and interactive maps for selecting the relevant data, interesting rules or rule

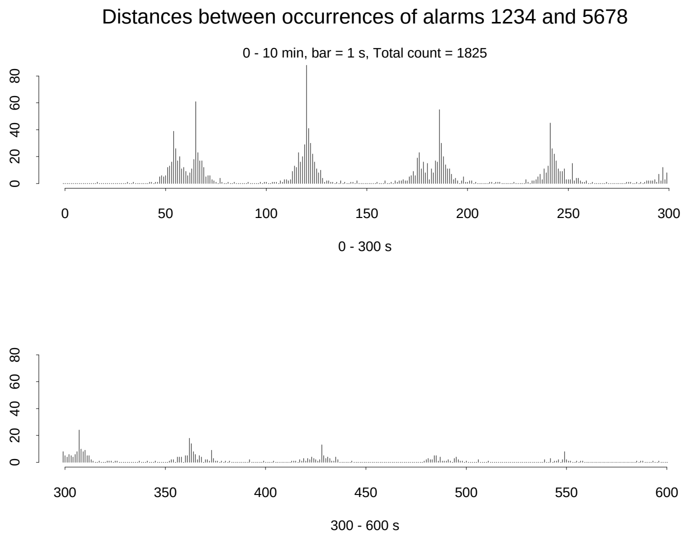


Figure 4.15: Distance histogram for the IF part and the THEN part of a rule.

clusters etc. — could be added in order to help the user.

4.5 Discussion

Association and episode rules are a simple and useful form of knowledge that can be efficiently discovered from large data sets. The collection of all rules that holds in a given relation provides a lot of information about that data. The problem with these data mining techniques is that the collection of all rules is typically very large, especially when one is interested also in the behavior of small subgroups of the original data set.

We have considered the problem of finding interesting rules from the collection of all association or episode rules. Our starting observation is that pruning based on objective measures such as support, confidence, statistical significance, and J-measure thresholds is quite effective with respect to the number of rules. Based on our experience, though, several objective measures might be useful at different stages of the KDD process to capture subjectively interesting knowledge. For reasons of efficiency, they — or at least all of them — should not be calculated in advance; so, there should exist some kind of evaluation functions for the IF–THEN part patterns that could be called when needed.

The above measures fail to take into account special subjective interests or domain knowledge. We have described how the simple idea of using templates defined in terms of a hierarchy on the attributes can be used to prune the rule sets effectively and according to the user's intuition. We have implemented and experimented with the template mechanism. It has proved to be useful: templates make it possible to define different areas of interest in a straightforward manner. For recent related research, see, e.g., [GdBV98].

A problem that remains is redundancy: different rules might express about the same thing. Large amounts of rules could potentially be pruned, if there were appropriate ways to remove redundant or nearly redundant rules. To alleviate this problem, we have considered the problem of pruning and grouping of rules, in order to improve the understandability of the collection of discovered association rules. The methods we presented for these tasks — forming rule covers and rule clusters — are based on how the rules match the data set. In our experiments, rule covers turned out to produce useful short descriptions of large sets of rules. Particularly useful are structural rule covers which can be computed very efficiently. In addition, as we saw in Section 4.2.2, we could use the J-measure to help in pruning redundant rules: If two or more rules sharing the same items or events have (approximately) the same confidence, we could take the one(s) with the highest J-measure value(s). However, as there is no absolute definition for redundancy, it remains a problem.

In order to structure the rule set, we created rule clusters. We derived the distance measures from the matched database rows. In the experiments we were able to find interesting and useful rule clusters from rule covers. The distance measures of rules and parameter settings of the clustering algorithms are subject to further investigation. These automatic methods of pruning and grouping of rules should be augmented with user-guided methods, such as templates, to take into account domain knowledge and user's interests.

Although not presented here, also some other possibilities exist for eliminating unnecessary details from a set of rules. Han et al. [HCC92, HF96] consider an attribute-oriented approach for finding generalizations of relations. This approach could perhaps be applied in finding generalizations of rules. On the other hand, association rule algorithms can be extended to find such general rules, too, if an attribute hierarchy is available [HF95, HKMT95, SA95]; the methods presented for association rules can be applied on episodes as well.

Chapter 5

Knowledge Discovery Process

In this chapter, we discuss our view of the whole knowledge discovery process from data collection to using the discovered knowledge. First, we describe our methodological framework (Section 5.1), and then we discuss the phases of the process in TASA based on the given methodology (Section 5.2).

5.1 A Methodology for Knowledge Discovery

KDD is an iterative and interactive process. In the core of the KDD process are the algorithms for discovering different types of patterns (rules, trends, etc.) from data. However, in the whole problem of obtaining useful knowledge, the inference of patterns is only a small part.

A general KDD process (adapted from [FPSS96b]) consists of several phases and iteration between them (Figure 5.1). The main steps are the following:

1. Data preprocessing (selection, cleaning, error and missing data handling, etc.),
2. Data transformation and input selection for discovery phase,
3. Discovery of patterns (data mining);[1]
4. Presentation/postprocessing of the results, and
5. Interpretation and utilization of the results.

Most knowledge discovery systems prune and sort the set of discovered patterns during the search for the patterns. In essence, this implies a predefined or fixed focus for the intended results. We adopt a different approach. We use methods that locate all rules from a given class that satisfy

[1] We use here the terminology of [FPSS96b], where "data mining" refers to the pattern extraction part of the KDD process.

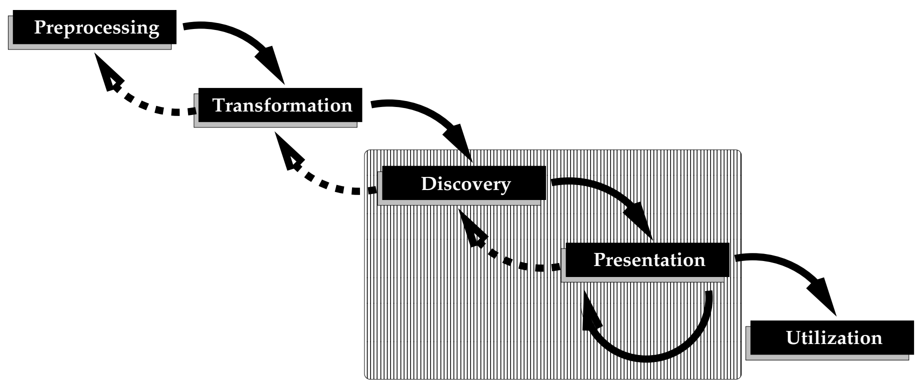


Figure 5.1: The KDD process model. The black arrows from left to right represent the order of steps in the KDD process, while the dotted arrows from right to left represent steps backwards that are sometimes necessary but should usually be avoided. The shaded box emphasizes the most important phases of our KDD methodology with iteration on the presentation phase.

certain simple frequency constraints. Discovered rules are not pruned automatically, but the user has explicit control over the operations on the rule set. This way the user can always be sure that his/her view of the data is in a sense complete.

Our methodological approach is based on two simple assumptions:

- *Iteration is essential in KDD.* It can be hard, or even impossible to specify beforehand, what is interesting or relevant.
- *Pattern generation from very large data sets is time consuming.* Efficient algorithms exist, but the pattern extraction phase still is a barrier for smooth interaction.

The algorithms we consider find all rules that fulfill some given threshold criteria, and the decision of what is useful is for the most part left to the domain specialist to explore. By producing all patterns at once different views of the data can be created very efficiently in the pattern presentation stage. To be more concrete, we propose a KDD process where emphasis is on two central phases:

1. In the pattern discovery phase, find *all* potentially interesting patterns according to some rather loose criteria.
2. Provide flexible methods in the presentation phase for iteratively and interactively creating different views of the discovered patterns.

In the field of knowledge discovery, the goal is to discover previously unknown information. That is why it can be hard, or even impossible, to specify beforehand, what is interesting. This is particularly true with telecommunication alarms, because the networks are continuously changing.

Our motivation for discovering a lot of rules at once is that the requests of the network management experts for different viewpoints to the data can then be responded very quickly: a new pattern discovery phase is not necessary, as the request can be fulfilled by a new view to the already discovered patterns. By producing all rules at once different views of the data can be created very efficiently in the presentation phase. The idea is that any rule that occurs frequently enough can potentially be interesting. The algorithms find all rules that fulfill the given threshold criteria, and the decision of what is interesting is for the most part left to the network manager.

Preprocessing and transformation steps are typically needed before repeating patterns can be discovered (Figure 5.1). A goal of our model is to avoid repeating the time consuming phases of preprocessing, transformation, and discovery.

There are several KDD formalisms that fit in this setting. Association and episode rule algorithms can efficiently discover thousands of rules from databases. The basic idea — iteration in the pattern presentation phase — can, however, be applied to any formalisms that have certain properties:

- The desired focus is not known definitely in advance.
- There is an algorithm that can produce all patterns from a class of potentially interesting patterns.
- The time requirement for discovering all potentially interesting patterns is not considerably longer than if the discovery was focused to a small subset of the potentially interesting patterns.

A somewhat similar approach has been presented in [ST95] and [ST96]. Their approach, however, emphasizes automatic filtering while in our methodology the user has total, explicit control over the operations on the rule set.

The approach of discovering all patterns can be contrasted with numerous methods, e.g., in machine learning, which aim directly at more focused discovery and produce one or at most few patterns that match the given problem specification. These methods usually require that the searched or learned subject is described quite carefully in advance. The advantage of these systems — none of which supports the methodology proposed here — is that the patterns they find are more expressive than the relatively simple association and episode rules; so, focusing the pattern discovery is thus more important. Explora [HK91, Klo95] finds interesting instances of statistical patterns. The patterns discovered by 49er [ZZ96] are contingency tables, equations, and logical equivalences. The Key Finding Reporter (Kefir) [MPSM96, PSM94], tailored with a lot of domain knowledge, discovers and explains deviations, and gives recommendations for corrective actions. In these systems the user can interactively change the focus, and require for a new pattern discovery.

The KDD process and methodology have been discussed in [BA94, FPSS96b, FPSS96a], which also underline the importance of interaction and iteration. In these articles, however, the iteration concerns the whole process or the pattern discovery and presentation phases.

5.2 Discovery Steps in the Methodology

5.2.1 Preprocessing and Transformation

In the first two phases of the whole process, common to both the general approach and our approach, the raw data is collected and prepared into a suitable form for the knowledge discovery. In particular, the preprocessing phase is crucial to the efficiency of the process, since preprocessing can require as much as 80 per cent of the total effort [Man96]. In addition to the messiness of the data, this is because in most of the technical domains there is plenty of background knowledge that can and must be used. This knowledge has to be applied both in the preprocessing and postprocessing phases in order to discard irrelevant fields, or to remove patterns that are impossible.

As with any data analysis method, whether it is statistics, machine learning, or data mining, the following aspects of data have to be taken care of [FPSM91]:

- *Dynamic data.* The contents of the database are changing all the time, thus affecting the validity of the results.
- *Irrelevant fields.* Not all the fields in the database are relevant with respect to the discovery goal. However, sometimes really surprising connections between some events can remain undiscovered, if too many fields are discarded from the analysis.
- *Missing values.* One has to detect the missing values and decide what to do with them: They may be replaced with some neutral value, the entire field may be ignored, etc.
- *Noise and uncertainty.* Errors or inaccuracies in measurement devices can cause some faulty or too general data.

- *Missing fields.* If data from different sources, for instance, are grouped together, it can cause some inconsistencies.

In most cases, databases contain some inconsistencies, missing values, and noise. Most of the data mining methods available take care of the above aspects, because they do not try to find only relationships, that are always true, but they also allow more or less uncertain connections; that is also the case with our methods. However, Ragel and Crémilleux show in [RC98b] that the execution of Apriori slows down if missing values are considered as some kind of special values. On the other hand, they propose in [RC98a] that association rules can be used in order to predict some of the missing values.

In the sequel, we discuss the preprocessing steps in handling imperfect telecommunication data. Before rule discovery, we always perform a general kind of analysis on the data and check its validity. If there are errors in the analyzed alarms, such as missing alarms, or wrong or missing time stamps, two policies can be adopted. If the errors are known, they can be fixed before episodes are searched for. Alternatively, episodes can be discovered in the original data, in which case, for instance, a confidence close to 100% can suggest that there is an actual causal relation between the alarms, but in some cases alarms were missing.

We also discard clearly irrelevant fields — e.g., seconds field or fields that have only one value — and infer potentially useful new fields. For instance, in the case of telecommunication alarm data, attributes can be derived to contain, e.g., information about the network topology and time fields such as if the current day is a workday or not.

The preprocessing and selection in our methodology does not, however, usually imply a refined process of defining interestingness as in some other methodologies. That is because even for a domain expert, it is often impossible to specify what is interesting before actually seeing what the results are like. After receiving some knowledge the user can better focus the search more interesting areas.

The process of obtaining useful information relies both on preprocessing the data before the discovery phase, and on postprocessing the results after the discovery phase.

Example 5.1 When looking at the sizes of the alarm data sets, one has to remember that the data sets have been cleaned for the experiments; i.e., both some uninformative fields and non-alarms have been discarded. To give some idea of the reduction rate, consider the following case from one of our other real-life data sets not considered here:

First alarm:	20.08.96 at 12:47:48
Last alarm:	21.10.96 at 16:35:41
Original file size:	90 MB
Cleaned file size:	38 MB
Total number of alarms:	334 669
Real alarms:	77 837
Non-alarms:	256 832

In the above case, the non-alarms contain very low-level alarms (such as "disturbances"), notices, etc. When you also remember that some filtering has already been done before the data was logged, the original size of a 100 000 alarm data set can be close to half a million alarms. □

Example 5.2 To prepare the data set, especially for the first time, one has to scan all the attributes and identify the ones that can potentially give us some information; do not try to discard too many attributes, if not necessary and if the data set is not very familiar.

In Table 5.1 there are examples of preprocessing the example alarm data sets. For association rule creation, we analyzed the data sets, and removed attributes which had too many values (time fields etc.) or attributes which had only one value. For instance, the most frequent non-identifier attribute in data set 4 had 55 936 different values, and 27 659 values in data set 6. The data set 4 had originally 61 attributes from which 10 were different time values and 11 had only one value, so the remaining number of attributes was 40. As could be seen in Section 3.4.3, this preprocessing is not sufficient: there are still too many attributes with too many different values for association rule algorithms. □

For alarm data, it is usually rather straightforward to choose the appropriate events and time stamps for episode generation. Usually the main

#	Alarms	Processed attributes	Time etc. attributes	1-value attributes	Considered attributes	Different values	Largest value
1	19 429	16	2	0	14	7184	1 982
2	58 616	15	4	0	11	5912	2 325
3	22 158	16	3	2	11	233	60
4	98 413	61	10	11	40	17594	13 776
5	43 478	45	22	0	23	5503	2 223
6	73 679	29	9	1	19	3411	2 595

Table 5.1: Examples of the preprocessed alarm data sets.

task is to filter out the non-alarms, i.e., notifications etc., because often the amount of notifications is huge compared to the amount of actual alarms. This way we can, however, lose interesting information about the relationships between alarms and the preceding symptoms. This is also true with attributes, i.e., alarm predicates: In preprocessing, sometimes the alarm type can only be detected using the values of the other fields, and sometimes some field acts as a unique identifier for the alarms even though the field originally also has some other purpose.

The situation is not always as easy as for telecommunication data, and preprocessing can require special attention. For instance, there are certain special aspects in the preprocessing of textual data. Next we describe briefly the steps that might be required in the preprocessing of text.

Text consists of words, special characters, and structural information. The preprocessing required depends heavily on the intended use of the results. Typically, the data is homogenized by replacing special characters and structural information (e.g., SGML tags) with symbols. Punctuation marks and structural information often need to be handled separately. Some of them may be ignored entirely, some of them may require special treatment, e.g., to find and process the sentence breaks. The longer distance between words of different sentences may be taken into account by inserting a larger gap in the indexing scheme; similar approach has been used in [LAS97].

Preprocessing of text may involve some amount of natural language analysis. Morphological analysis gives us detailed information of the data which may be included in the *feature vector*; a *feature* in the vector can be a word (e.g., base form, inflected word form, stem), a grammatical feature (e.g., part of speech, case, number), a punctuation mark or other special character, or an SGML structure tag. We may also use this analysis to generalize the data, e.g., by replacing words by their parts of speech, which allows us to identify constructs such as *(preposition, noun)* instead of combinations of specific words.

Filtering of the data is used to focus the discovery phase, to limit the number of results so that we are not overwhelmed by uninteresting rules, and to decrease the processing effort needed in the discovery phase. Pruning may be done either before or after the discovery phase. If we do not have a clear idea of what kind of regularities we are looking for it is often advisable to defer the pruning decisions to the postprocessing phase instead of heavy pruning in the preprocessing phase. On the other hand, with large collections of documents, efficiency has to be taken into account. In cases where we know what features we are not interested in, preprocessing must be preferred, to limit the size of the search space and the time requirement of the discovery phase.

In the preprocessing phase of text, pruning can be used in two distinct ways. We may prune entire feature vectors, i.e., drop uninteresting items such as articles, prepositions, non-informative verbs (e.g., *be*), or punctuation marks, or select only some class of words (e.g., nouns) to be examined. We may also focus on some features of each word, e.g., by leaving only the base form of the word, or only part of the morphological information. An example of the preprocessing of one sentence is given in Example 5.3.

Example 5.3 The sentence

> *Documents are an interesting application field*
> *for data mining techniques.*

is first preprocessed into the following tagged form (`N` is noun, `PL` is plural, `V` is verb, `PRES` is present form, `DET` is determinant, `A` is adjective, `POS` is positive, `SG` is singular, `PP` is preposition):

(*document*_N_PL, 1) (*be*_V_PRES_PL, 2)
(*an*_DET, 3) (*interesting*_A_POS, 4)
(*application*_N_SG, 5) (*field*_N_SG, 6)
(*for*_PP, 7) (*data*_N_SG, 8) (*mining*_N_SG, 9)
(*technique*_N_PL, 10) (STOP, 11)

In the following step, uninteresting parts are pruned, resulting in

(*document*_N_PL, 1) (*interesting*_A_POS, 4)
(*application*_N_SG, 5) (*field*_N_SG, 6)
(*data*_N_SG, 8) (*mining*_N_SG, 9)
(*technique*_N_PL, 10)

The text may also be pruned to contain, e.g., only nouns,

(*document*_N_PL, 1) (*application*_N_SG, 5)
(*field*_N_SG, 6) (*data*_N_SG, 8)
(*mining*_N_SG, 9) (*technique*_N_PL, 10)

or, depending on the goal of the research, the word itself may be discarded and only the morphological information retained,

(N_PL, 1) (A_POS, 4) (N_SG, 5) (N_SG, 6)
(N_SG, 8) (N_SG, 9) (N_PL, 10)

□

A more elaborated method for handling structured and complex data using WARMR is introduced in [DT98]. With this approach, one can create different views about the data thus reducing the needed effort for preprocessing.

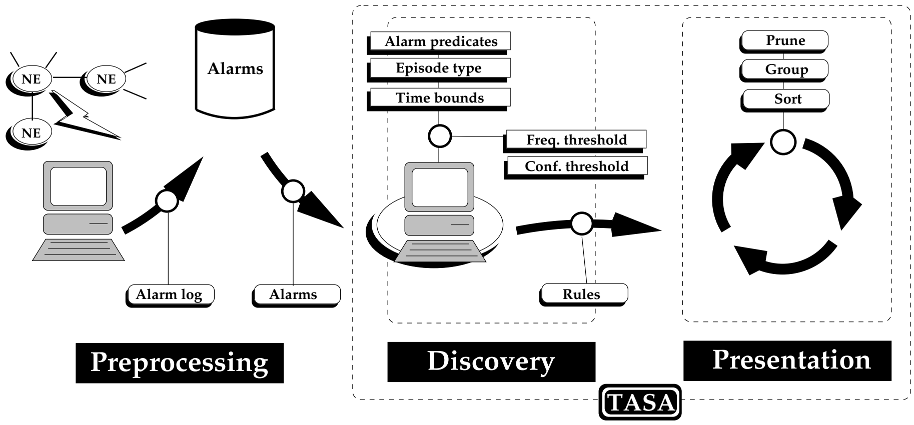


Figure 5.2: TASA system covers discovery and presentation phases of the KDD process.

5.2.2 Discovery of Patterns

In the pattern discovery phase all potentially interesting patterns are generated from the data set. The (rather loose) criteria for interestingness can be, e.g., frequency, prevention of taking rarely occurring phenomena into account, or strength of the patterns. In TASA, the parameters are the following:

- rule type: unordered/ordered association rules/episode rules (required);
- frequency threshold (required);
- confidence threshold (required);
- set of time bounds or maximum time bound (for episode rules, required);
- maximum rule size;
- bounds for the sizes of both rule left-hand and right-hand size.

See Figure 5.2 for an illustration and Figure 5.3 for the user interface of TASA for episode rule creation. The discovery method outputs each episode rule and association rule satisfying the given conditions. The conditions should typically be quite loose, so large amounts of rules are discovered. For each rule, TASA outputs the rule confidence and frequency.

Now we describe the parameters in more detail.

Frequency Threshold. For both episode and association rules, a *frequency threshold* is given by the user. The frequency for an episode rule is

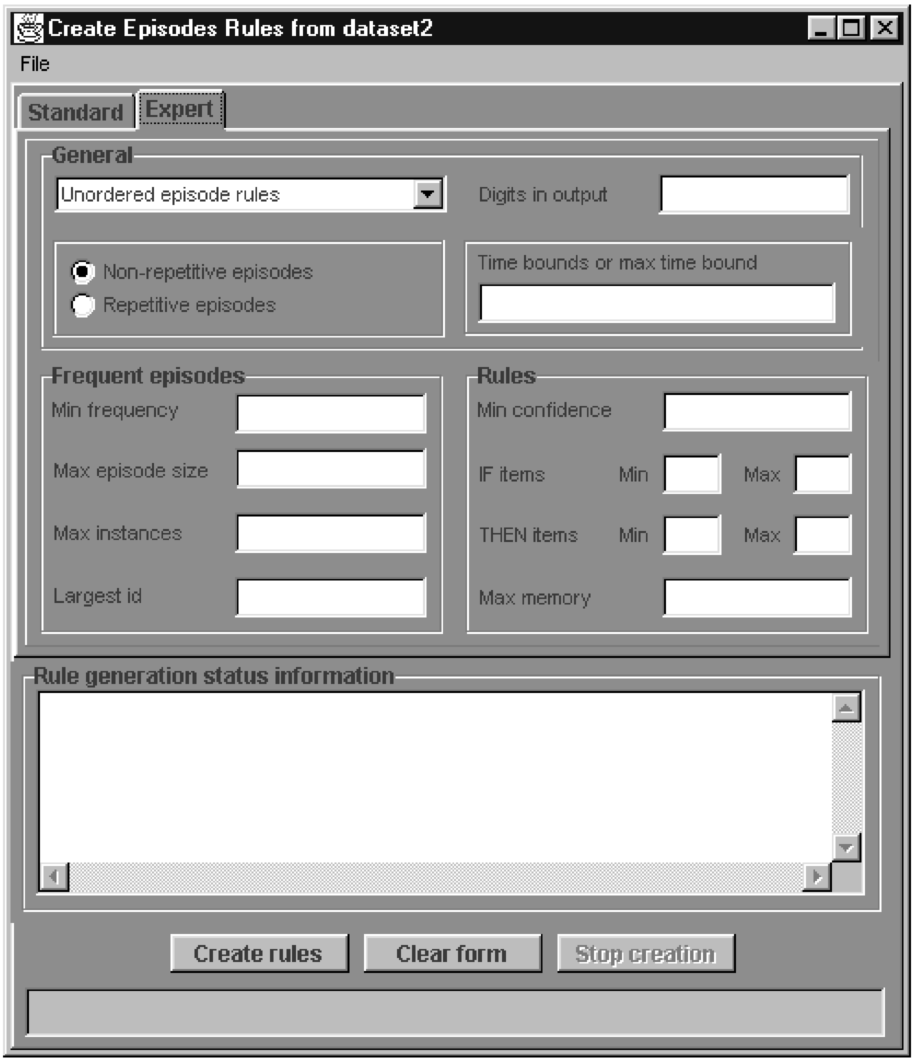


Figure 5.3: Episode rule creation with expert mode in the final version of TASA; standard mode asks only for the required parameters.

defined in Section 3.1 on page 31 and for association rules in Section 3.2 on page 28. The method outputs all episode and association rules specified by the parameters above, whose frequency exceeds the user-specified threshold.

The method is aimed at discovering rules describing relatively large subgroups of the data, not spotting interesting individual cases. With the frequency threshold the user is able to filter out rules that occur too rarely to be trustworthy. For instance, with a frequency threshold of 20 an episode is output only if it appears at least 20 times in the analyzed database. The algorithm is complete in this respect: it is guaranteed to output all episodes that have at least 20 occurrences.

The value of the frequency threshold is crucial for the running time of the algorithm. If the threshold is low, also rules that occur rarely are included in the output, and the discovery time is longer. Suitable values depend heavily on the nature and amount of data. For a database with 100 000 alarms and several hundreds or even thousands of alarm types, thresholds in the range of 50 to 500 have proved to be reasonable. For discussion and experiments on the frequency threshold, see Section 4.2.1.

Confidence Threshold. For both episode and association rules, the user also specifies a *confidence threshold*. The algorithm then outputs all episode and association rules whose confidence is at least the given threshold value.

The confidence threshold allows the user to filter out rules that are too weak to be useful. For instance, a confidence threshold of 80% limits the output to rules that hold with at least 80% certainty. The confidence threshold has no large effect on the running time, so it is useful to specify a low threshold in the rule discovery phase and to prune weak rules later interactively with the postprocessing tools. Note also that even a rule with a low confidence can be interesting. For discussion and experiments on the confidence threshold, see Section 4.2.1.

Set of Time Bounds. For episode rules, the user also supplies a set W of time bounds that are used in the rules. Two aspects guide the setting of W.

1. The maximum time bound in W should be larger than the maximal temporal duration of the phenomena that are searched for.
2. The number of time bounds in W directly affects the temporal granularity at which episode rules are found.

Fault management experts have typically preferred time bounds ranging from 5 seconds to 10 minutes. An example set is the roughly geometrically growing time bounds 5 s, 10 s, 30 s, 1 min, 2 min, 5 min, and 10 min. The higher the number of time bounds is, the higher is the number of rules, too. The effect on the running time is not strong.

In the pattern discovery phase, the maximum time bound has to be adjusted properly. For example, in the case of telecommunication network alarm data there usually is a roughly known window size within which interesting relationships exist. If the window size is not set properly, relationships can remain hidden if they do not fit into the window, or they can be buried under noise, if the window is too large. For experiments on episode rules with different time bounds, see Section 3.4.

Example 5.4 In the course enrollment database, there are 2 010 association rules that match at least 11 students (i.e., the frequency threshold is 0.01). There are 6 715 rules that match at least 6 students.

In the alarm data set 5 there are 5 498 serial MINEPI episode rules that hold with a frequency threshold of 10 (absolute value) with a maximum time bound of 60 seconds. Note that the number of rules equals the number of distinct rules as described in Section 3.3.2 on page 39.

In the same data set, when considering associations between attribute values of individual alarms, there are 1 497 association rules that match at least 435 alarms (i.e., frequency threshold is 0.1). □

The goal of the methodology is that returning to the pattern discovery phase is needed as seldom as possible. It is realistic to expect, however, that the rule discovery algorithms need to be run several times. Parameter settings may need adjustment, or errors in the data cleaning phase may be revealed so that returning to the first phase of the process is necessary.

5.2.3 Postprocessing

The presentation and postprocessing of discovered knowledge is a main part of our methodology. In this phase the relevant patterns should be located from large collections of potentially interesting patterns. Efficient postprocessing requires tools to enable sorting and filtering the episodes and rules produced by the discovery phase. The role of the KDD system is to provide appropriate tools for displaying and browsing the discovered knowledge, the methods for expressing the desired focus, and to support iterative specification of the focus.

For the most part, what is interesting depends on the case, and is highly based on the user's personal aims and perspective. As argued already, iteration is needed. With iteration, it is possible to refine or change the focus after seeing what the results are like. Iteration serves the explorative task of becoming familiar with the data. Iteration also has an important role in recalling passive knowledge, i.e., knowledge that can not be expressed without seeing examples of data and relationships in it. Looking at and focusing into the discovered patterns iteratively simplifies the task considerably. Interestingness and related issues were discussed in detail in Chapter 4.

Stat.	Phrases		Co-occ. terms		Parts of speech		Morphology		Structure	
	Epis.	Rules	Epis.(1)	Epis.(2)	Epis.	Rules	Epis.	Rules	Epis.	Rules
1	39	3	37	118	964	2 234	600	807	266	775
2	428	130	437	2 362	7 756	23 759	25 164	75 303	1 678	2 837
3	29	5	28	61	531	1 121	871	2 317	857	1 958
4	276	83	345	1 532	4 746	13 768	11 579	32 322	1 354	2 870
5	89	11	74	194	1 512	3 758	2 213	4 977	999	2 010
6	812	371	1 090	4 911	12 014	39 035	61 816	206 237	3 148	5 800
7	80	14	90	311	1 790	4 576	1 656	3 397	727	1 553
8	321	217	343	1 601	4 316	12 316	12 051	35 971	1 219	2 222
9	43	10	47	113	869	2 025	523	707	887	2 042
10	56	9	57	186	1 221	2 950	2 209	5 905	946	2 059
11	122	66	114	376	1 291	3 133	4 475	13 843	1 279	2 387
12	32	4	35	124	947	2 279	727	1 255	671	1 394
13	62	12	70	191	1 116	2 688	757	1 038	939	2 060
14	48	9	61	110	1 132	2 721	816	1 303	729	1 519

Table 5.2: The test material and results. The number of co-occurring terms is taken from the test with a gap between sentences; without a gap the figures are about 20% higher. With co-occurring terms, figures (1) and (2) refer to the two different experiments described in Table 5.3.

Example 5.5 Text documents are a good example of a case where one can easily produce huge amounts of rules. Consequently, postprocessing is essential to permit focusing the study after we have seen what kind of rules and episodes are found in the data. This is due to the fact that text is usually rich and very heterogeneous, i.e., words do not in general appear in the documents as often as events in some other data sets, and one has to use very low thresholds. See Tables 5.2 and 5.3 for parameters and results of the experiments with our document collection set.

Typically, the user may be interested in a single word or a selected set of words, and only the rules that contain these words are included, while all others are filtered out. If discrimination between several documents is needed, the frequency of phrases is compared to the others in the collection to evaluate, if the phrase is considerably more frequent in this document than in general.

Many of the pruning decisions may be done either before or after the discovery phase. It is often advisable to defer these to the postprocessing phase instead of heavy pruning in the preprocessing phase, if we do not have a clear idea of what kind of regularities we are looking for. Instead of pruning entire feature vectors during preprocessing (e.g., dropping all prepositions) we can in the postprocessing phase ignore all episodes and rules containing those feature vectors. However, during the postprocessing we cannot efficiently prune the single features included in the feature vector. Likewise, if we end up with a collection of very specialized rules instead of

Discovery of	**Episode type**	**Selected parts of speech**	**Distinct symbols/ document**	**Gap between sentences**	**Window sizes**
Phrases	serial	N, PROP, A	365..2693	3(+1)	1..3
Co-occurring Terms	parallel	(1) N, PROP (2) N, PROP, V, A	293..1904 457..3109	(a) 10(+1) (b) 1	1..10 —
Parts of Speech	serial	all (no words)	17..21	5(+1)	1..5
Morphology	serial	all (no words)	352..1245	5(+1)	1..5
Structure	serial	— (tags only)	34..62	10(+1)	1..10

Table 5.3: The test parameter values. Note the abbreviations for nouns (N), proper nouns (PROP), adjectives (A), and verbs (V). For "parts of speech" and "morphology", "all" means that only special characters have been filtered out. The threshold for episode support was 10 and for episode rule confidence 0.2, when applicable.

more general ones, we need to have a way of combining these using some predefined hierarchies of the features. On the other hand, with large collections of documents, efficiency has to be taken into account. In cases where we know what features we are not interested in, preprocessing must be preferred, to limit the size of the search space and the time requirement of the discovery phase.

As each feature vector is handled as one entity in the discovery phase, it is not possible to study the relation of distinct features in the co-occurring words, e.g., *noun* followed by *plural*. Additionally, co-occurrence of, e.g., two parts of speech may not become visible, if this need is not taken into account during preprocessing. The phenomenon itself might be frequent enough, but if the parts of speech are combined with, e.g., word stems, any distinct combinations may not occur frequently enough to be considered. On the other hand, if the support threshold is set very low, the result contains mostly uninteresting rules. Therefore, choosing a clear focus affects the result remarkably. □

The parameter selection of the pattern discovery phase can sometimes have a very big impact on the number of rules and thus also on the post-processing phase. With alarm data, for example, even a small change in the frequency threshold in the discovery phase can dramatically increase or decrease the number of frequent episodes and episode rules. For instance, consider the following case. With alarm data set 1, the difference between the number of frequent parallel Winepi episodes with window 120 s and shift 12 s using thresholds 0.004 and 0.0045 is huge; see Table 5.4. The total number of windows is 16 390, so threshold 0.004 equals to 66 windows and threshold 0.0045 to 74 windows. If one uses the lower threshold, problems

Winepi: Parallel episodes with window size 120 s and shift 12 s				
	0.004		0.0045	
Iteration #	Candidates	Frequent episodes	Candidates	Frequent episodes
1	1.982	325	1.982	293
2	52.650	2.302	42.778	1.791
3	18.804	6.161	12.880	2.433
4	23.518	21.567	3.067	2.324
5	76.081	76.040	2.649	2.631
6	214.165	214.165	2.466	2.464
tot	387.200	320.560	65.822	11.936

Table 5.4: An example of a dramatical increase on the number of parallel Winepi episodes from alarm data set 1 with window 120 s and shift 12 s using thresholds 0.004 and 0.0045.

will certainly occur in trying to cope with the resulting rule set of several millions of rules.

In Example 5.6, we introduce a KDD process scenario of telecommunication networks fault analysis using association rules. It is a simplified problem of knowledge discovery to support off-line network surveillance, where a network manager tries to identify and correct faults based on sent alarms. A comprehensive discussion on this application is available in [KMT99b].

Example 5.6 The network manager decides to get some insight into the behavior of the network during the current month. So, he/she selects the data of the valid time period for closer examination, and focuses on the association rules derived from the data set. Before the manager goes any further, he/she "tunes" some parameters for the search by pruning out all rules that have a confidence under 5% or a frequency under 0.05% or a size of more than 10 items in one rule. This pruning is based on his personal intuition that weaker rules can not be of interest and bigger rules are too hard to understand. The network manager then recalls that in this case some attributes in the database are not interesting. Thus, he/she decides to exclude fields "alarm text" and "time".

The number of rules in the resulting rule set is, however, quite large, and hard to handle. The manager decides to focus on the rules from week 30 and to restrict the maximum amount of items in the rule to 5. After this procedure, the amount of rules is acceptable. While browsing the collection of rules, the network manager sees that a lot of rules concern the network element *E*. That reminds him/her of maintenance undertaken at the beginning of the week which explains those rules. So, he/she decides to remove all rules that contain network element E or its subcomponent.

At first glance, the resulting set of rules seems not to show anything special. Just to make sure the network manager wants to compare the behavior of the network to the preceding similar period (week 29) and find out possible differences between the rule sets. The network manager then picks up one rule, that looks interesting and is very strong (confidence is close to 1), and he/she wants to find all exceptions to this rule; i.e. rows, where the rule does not hold.

By browsing the changed information he/she sees that during week 30 the critical component of the rule does not emit so many severe alarms anymore. He/she prints the resulting rules as a report so that he/she can look at them with time later on.

It should be noted that different kinds of patterns can be used in a single scenario in order to get a better picture of the considered problem. A real-life analysis consists of several subtasks that require iteration inside the considered rule type collection, between different rule type collections and data (see Figure 5.4). In our case, for instance, one might first look at the source data, then filter the association rules and based on the resulting rule set select episode rules that contain interesting alarms. This is more complex than the more simple case of, e.g., tuning parameters, filtering uninteresting rules, and presenting the rules within some rule type.

As this scenario shows, postprocessing can be viewed as normal SQL-like queries over rule bases and databases, i.e., a query activity over collections of frequent patterns and data. Both this and also, e.g., the amount of rules, data and efficiency, would require an advanced database system. □

5.2.4 Utilization of Discovered Knowledge

Knowledge Acquisition for Alarm Correlation. After postprocessing, the knowledge found has to be understood and applied (Figure 5.5). The patterns can be useful in several ways.

- A network manager can use recurrent patterns or rules as correlation patterns, after possibly editing them and assigning them appropriate correlation actions. Discovered patterns can be used for filtering uninformative alarms, for combining alarms to construct hypotheses of faults, or for prediction of faults.
- Recurrent patterns or rules can reveal previously unknown relationships between alarms. An unknown relationship can stem, e.g., from

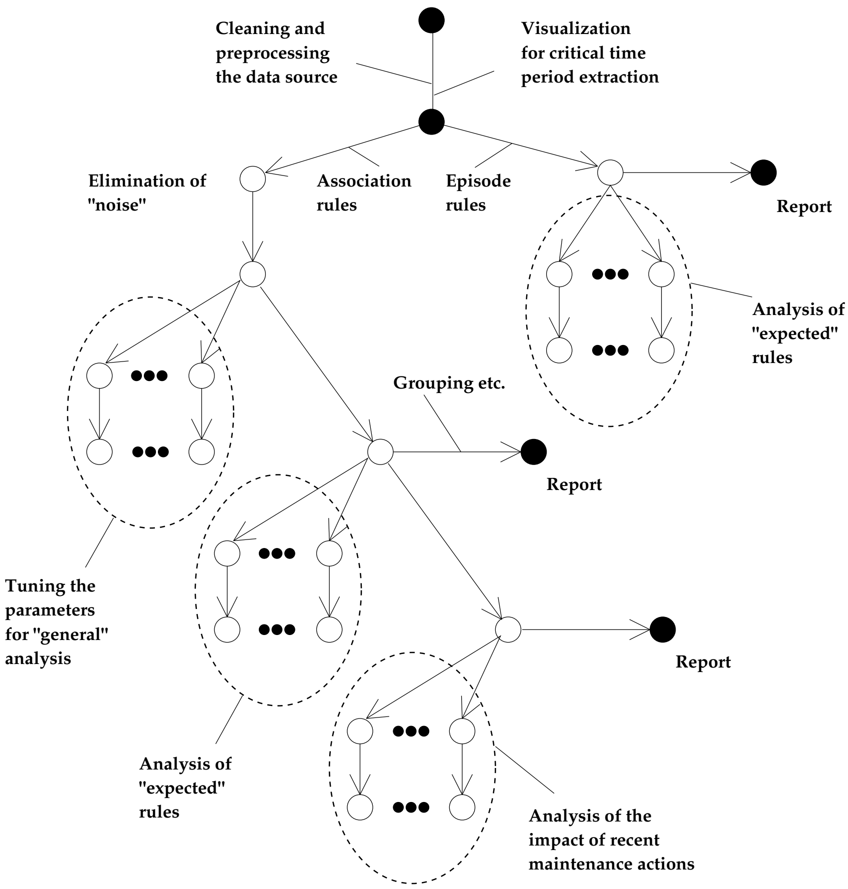


Figure 5.4: An abstraction of an analysis tree of a real-life data mining process.

a fault or a set of faults, or an anomaly in the network, such as an unusual or erroneous configuration.

- Recurrent patterns or rules are also useful for providing different views to the analyzed alarms. Our discovery methods, although they are not directly applicable for on-line analysis themselves, have turned out to be useful also in network surveillance. The analysis can be rerun or augmented, e.g., every day or every hour. Recent patterns may point to yet unnoticed problems in the network.

Next we give examples of the use of discovered rules. Knowledge acquisition for alarm correlation is explained in Example 5.7 and a scenario for network surveillance is given in Section 6.2.

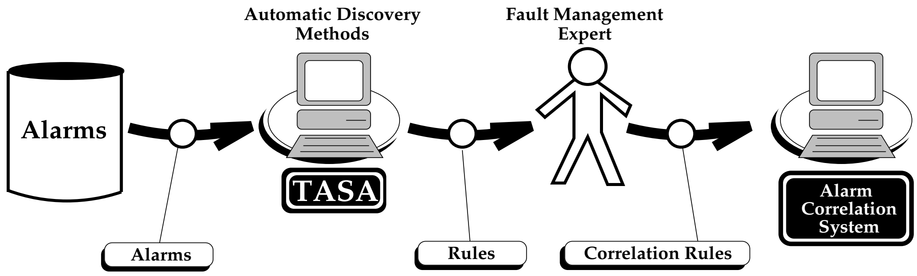


Figure 5.5: Rules discovered in an alarm database can be used in the construction of alarm correlation systems.

Example 5.7 Consider an alarm correlation system which operates in real time and is also able to handle delayed alarms and slightly inaccurate time stamps. In the correlation patterns, delays are handled with a special *wait* function. Episode and association rules can be applied in this system in a rather straightforward way. The serial episode rule

```
IF    link alarm
      link failure
THEN  high fault rate
WITH  [5]  [60]  conf(0.70)  freq(700/1000)
```

discovered by TASA can be coded in the system as at the top of Figure 5.6. That is, if a "link alarm" occurs and a "link failure" follows within 5 seconds, the rule THEN part information is sent and the original alarms are suppressed. If a "link failure" does not follow within 5 seconds, the original "link alarm" is forwarded.

The correlation procedure can be enhanced using association rules. For example, assume that we have detected that if the alarm type is "link alarm", the time of the day is office hours, and alarm severity is 1, then with probability of 95% the alarming element type is BS. After expert evaluation of the rule we know that such alarms from network elements of type BS can be ignored. However, if an element of any other type sends an alarm of type "link alarm" with the same properties, it is an interesting one. The correlation function at the bottom of Figure 5.6 removes the above alarms sent by network elements of type BS. □

Application Experience with TASA. The fault management experts in the telecommunication companies have found TASA useful in

- finding long-term, rather frequently occurring dependencies,

```
if "alarm type = link alarm" then
    start time;
    wait until "alarm type = link failure" or "time = 5 sec";
    if "alarm type = link failure" then
        display warning "high fault rate with 70% probability
        in 60 sec"
    else forward the original alarm;

if "alarm type = link alarm"
and "office hours" and "severity = 1"
and "element type = BS" then
    exit
 else forward the original alarm;
```

Figure 5.6: Correlation functions.

- creating an overview of a short-term alarm sequence, and
- evaluating the alarm data base consistency and correctness.

Recurrent patterns or frequent rules can reveal previously unknown relationships between alarms. An unknown relationship can stem, e.g., from a fault or a set of faults, or an anomaly in the network, such as an unusual or erroneous configuration. Recurrent patterns are also useful for providing different views of the analyzed alarms. Although the discovery algorithms are not directly applicable for on-line analysis, TASA has turned out to be useful also in network surveillance. The analysis can be rerun or augmented, e.g., every day or every hour. Recent patterns may point to yet unnoticed problems in the network.

Unexpected dependencies have been found for example between network elements which are not closely connected in the network topology. An example of such a dependency is that when a remote device sends alarms, the fault influences another corner of the network through several devices, and not necessarily via the same routes and devices. So, just analyzing the neighboring devices might not reveal any strong relationships. However, when a larger region is analyzed, such a relationship can be detected. Beginning from the first tests, discovered rules have been integrated into alarm correlation systems.

On the other hand, many of the rules discovered by TASA are deemed trivial by the network managers. Some of the rules correspond to the knowledge that the network managers have about the behavior of the network, and some other rules reflect the assumed functioning of network devices.

An example of the former is, for instance, that it is known that if an alarm of type A occurs, it is always followed by an alarm of type B within 20 seconds; so, finding a rule like this gives no new information. Luckily, much of the background knowledge can be expressed using templates and hence the trivial rules can be pruned. Templates are also useful since the knowledge trivial to one expert may not be trivial to another, and with templates each expert may filter the rule collection based on his/her personal background knowledge.

Experience has indicated that the algorithms are not well suited for analyzing event sequences that contain bursty periods. This is because during the short periods some bursty alarms can have apparent relationships with many other alarms only due to their own volume. These bursty periods can then affect the rule characteristics of the whole sequence so much that the real relationships can not be found or too many faulty ones are discovered. Thus, it is sometimes more useful to cut off and analyze bursty periods separately.

Overall, TASA has been considered useful in telecommunication companies. Episode rules are being used as first drafts of correlation rules, whereas association rules are more typically used for creating short-term overviews in off-line network surveillance. Telecommunication operators are integrating these methods to their alarm analysis and surveillance systems, and the TASA system itself will reach a product status presumably within one or two years.

Telecommunication networks usually change and evolve quickly over time. It should be noted that even small changes can sometimes affect the behavior of the network substantially. In practice, this means that old rules that hold at a certain moment do not necessarily hold in a changing environment, whether because of actual topological changes or changes in the devices, software etc. One of the strengths of our approach is that reruns to get valid rules can be done efficiently. However, human evaluation is needed in deciding whether the rerun requires modifications to the rule-base of the correlation expert system.

5.3 Discussion

The goal of knowledge discovery is to find useful patterns in data. Success in this task requires not only that the KDD tools and the knowledge representation formalisms are such that useful patterns can be found and comprehended, but it also requires understanding all steps in the discovery process.

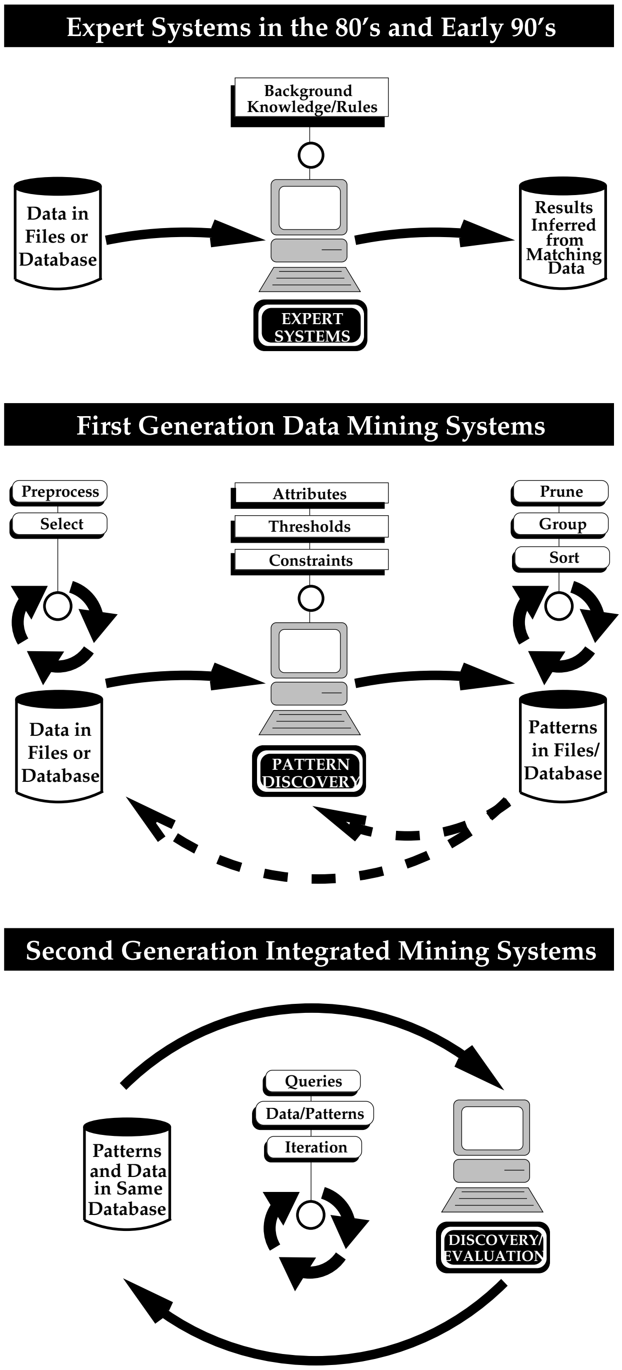


Figure 5.7: The evolution from expert systems to and within data mining systems.

We have described a knowledge discovery methodology that can be used to automate knowledge acquisition. The methodology has the following distinctive characteristics: (1) a large collection of potentially relevant rules is discovered at once, and (2) different views can be formed on the discovered rules iteratively and interactively. The motivation is that after the discovery of all potentially interesting rules, iteration is very efficient. Our experience in this field with both association and episode rules, and especially with telecommunication network alarm data, supports the claim that the methodology is useful in practice. For formalisms stronger than association and episode rules, strong focus in the pattern discovery phase may be essential for keeping the computation tractable.

However, iterativity and interactivity, as well as very large data sets and their postprocessing, pose serious requirements for the data mining system. That is why we would need a really efficient integrated data mining environment on top of a database system and data warehouses, while the current systems are still rather separate modules and produce much unnecessary load in all phases of the KDD process (see Figure 5.7). Nowadays, data warehouses provide means to store huge amounts of preprocessed and unified data. We also have sophisticated database query languages that enable querying efficiently the data in databases. So, a significant step is to combine powerful query languages and advanced data mining methods to be used efficiently in data warehouses. For instance, [GdBV98] shows how to use OQL in order to perform "structural rule covering" and "rule covering" [TKR+95], two global filtering methods over a collection of association rules stored in a nested relation. Moreover, Meo, Psaila and Ceri [MPC96, MPC98] propose the `MINE RULE` architecture as a concrete SQL-aware approach dedicated to association rule mining.

Chapter 6

Modeling KDD Processes within Inductive Databases

As was pointed out in Chapter 5, discovering knowledge from data is a process containing several steps: understanding the domain, preparing the data set, discovering patterns, postprocessing of discovered patterns, and putting the results into use. This is an interactive and iterative process for which many related sets of generalizations — for instance, association and episode rules — have to be computed: different selection predicates and also classes of patterns must be used. Note that sophisticated postprocessing is often needed in order to reduce the number of rules and that it can also be considered as a query on the current collection of rules.

In a state-of-the-art data mining system (see Figure 5.7 on page 99) the user defines a data mining task with different thresholds etc., starts the pattern generation by giving a batch run command, and queries the generated patterns. Our special interest is in combining the pattern discovery and postprocessing phases, and replacing commands and various kinds of queries with a query language.

For KDD we need a query language that enables the user to select subsets of data, to specify and select patterns. The user must also be able to cross the boundary between data and patterns, e.g., when exceptions to a pattern are to be analyzed. Thus a *closure property* of the query language is desirable: the result of a KDD query should be an object of a similar type to its arguments. This principle gives rise to the concept of *inductive databases*, i.e., databases that contain inductive generalizations of the data, in addition to the usual data. The KDD process can then be described as a sequence of queries on an inductive database. During the whole process, the user does not care whether he/she is looking at raw data, aggregations from the data, or "discovered" knowledge: he/she just sends queries. This concept has been suggested in [IM96], and developed further in [Man97] and [BKM98a].

Our basic approach is very simple:

- An inductive database consists of a normal database associated with a subset of patterns from a class of patterns, and an evaluation function that tells how the patterns occur in the data.
- An inductive database can be queried (in principle) using normal relational algebra or SQL, with the added property of being able to refer to the values of the evaluation function on the patterns.
- Modeling KDD processes as sequences of queries on an inductive database enables reasoning about and optimizing these processes.

In this chapter we show how the process view of knowledge discovery can be described in terms of querying inductive databases, i.e., databases that contain rules or patterns in addition to normal data. In Section 6.1, we present our framework for inductive databases [BKM98a], which uses the process approach and in a sense takes it a step further by describing an efficient data mining environment on top of a database system and data warehouses (see also Figure 5.7).

In Section 6.2 we return to the scenario about telecommunication networks introduced in Section 5.2.3, this time using the framework of inductive databases. General ideas about implementing inductive databases are considered in Section 6.3.

6.1 Inductive Databases

The main design issues of the inductive database framework are the following.

- **Patterns are an intensionally defined collection of queries.** It is elegant to consider patterns as queries. For instance, association rules are queries whose answers are their frequency and their confidence in the considered data set. In the general setting, the collection of patterns 1, 2, ... is being determined by the queries $Q_1, Q_2, \ldots$ The space of patterns/queries is structured by the definition of a language of sentences (or patterns), and the schema of the database part while the answers of the patterns/queries are determined also by the data. *This kind of query should not be confused with the queries over an inductive database during a KDD process.*

- **Closure property: mining processes as sequences of queries.** The *schema of an inductive database* is a pair that associates a database schema and the specification of a collection of patterns/queries over this database schema. An inductive database instance $(\mathbf{r}, \mathbf{s})$ consists of a database $\mathbf{r}$ plus a nested relation $\mathbf{s}$ that contains as many rows as there are patterns/queries in the instance. Each row defines a pattern/query and its answer in the data set. If $(\mathbf{r}, \mathbf{s}_0)$ denotes the instance of the whole intensionally defined collection of patterns, each operation, i.e., each query, returns an instance (r, s) where $r \subseteq \mathbf{r}$ and the patterns in s also belong to $\mathbf{s}_0$.
- **Simplicity and orthogonality with respect to pattern classes.** Simplicity is a major design issue of our framework. Orthogonality is one mean to provide simplicity. It means that the definition of patterns, their syntax, and their semantics is not "known" by the database and is defined independently.

It is worth relating the concepts of inductive and deductive databases. A deductive database uses some simple form of deduction to augment fact databases with a potentially infinite set of derived/deduced facts. An inductive database, on the other hand, contains a potentially infinite set of induced rules or some other form of more general information in addition to the usual facts.

Next we give formal definitions of the inductive database concepts.

Formal Definitions. Our goal is to describe a data model that makes it possible to view the whole KDD process as querying a database structured according to the model. Thus the database has to contain both data and generalizations about that data. This motivates the following definition that is slightly different from the one given in [Man97].

The *schema of an inductive database* is a pair $\mathcal{R} = (\mathbf{R}, (\mathcal{Q}_\mathbf{R}, e, \mathcal{V}))$, where $\mathbf{R}$ is a database schema, $\mathcal{Q}_\mathbf{R}$ is a collection of patterns, $\mathcal{V}$ is a set of *result values*, and e is the *evaluation function* that defines pattern semantics. This function maps each pair $(\mathbf{r}, \theta_i)$ to an element of $\mathcal{V}$, where $\mathbf{r}$ is a database over $\mathbf{R}$ and θ_i is a pattern.

An instance of the schema, an *inductive database* $\hat{\mathbf{r}} = (\mathbf{r}, s)$ over the schema $\mathcal{R}$ consists of a database $\mathbf{r}$ over the schema $\mathbf{R}$ and a subset $s \subseteq \mathcal{Q}_\mathbf{R}$.

Example 6.1 If the patterns are boolean formulae about the database, the set of result values $\mathcal{V}$ is $\{$`true, false`$\}$, and the evaluation function $e(\mathbf{r}, \theta)$ has value true iff the formula θ is true about $\mathbf{r}$. Typical data mining tasks of this kind are the discovery of functional or inclusion dependencies that hold in a relational database. □

Let us now recall the definition of association rules given in Section 3.1 and adapt it to the context of inductive databases. Given a schema $R = \{A_1, \ldots, A_n\}$ of attributes with domain $\{0, 1\}$, and a relation r over R, an *association rule* [AIS93] about r is an expression of the form $X \Rightarrow B$, where $\emptyset \neq X \subseteq R$ and $B \in R \setminus X$. The intuitive meaning of the rule is that if a row of the matrix r has a 1 in each column of X, then the row tends to have a 1 also in column B. Given $W \subseteq R$, let $s(W, r)$ denote the fraction of rows of r that have a 1 in each column of W. The frequency of $X \Rightarrow B$ in r is defined to be $s(X \cup \{B\}, r)$ while its *confidence* is $s(X \cup \{B\}, r)/s(X, r)$. In this case, $\mathcal{V}$ is the set $[0, 1]^2$, and $e(\mathbf{r}, \theta) = (f(\mathbf{r}, \theta), c(\mathbf{r}, \theta))$, where $f(\mathbf{r}, \theta)$ and $c(\mathbf{r}, \theta)$ are the frequency and the confidence, respectively, of the association rule θ in the database $\mathbf{r}$.

A typical KDD process operates on both of the components of an inductive database, the database and the patterns $s \subseteq \mathcal{Q}_{\mathbf{R}}$. The user can select a subset of the rows. The user can also select some subset of the patterns, and in the result the data component remains the same as before. At each stage of manipulating the inductive database $(\mathbf{r}, s)$, the user can think that the value of $e(\mathbf{r}, \theta)$ is available for each pattern θ which is present in the set s. Obviously, if the pattern class is large, an implementation will not compute all the values of the evaluation function beforehand; rather, only those values $e(\mathbf{r}, \theta)$ should be computed that the user's queries require to be computed. The situation can be compared with deductive databases where some form of deduction is used to augment fact databases with a potentially infinite set of derived facts. The user can query these facts while they are computed only at evaluation time.

Query Languages. Using the above definition for inductive databases it is easy to formulate query languages for them. For example, we can write relational algebra queries, where in addition to the normal operations we can also refer to the patterns and the value of the evaluation function on the patterns. To refer to values of $e(\mathbf{r}, \theta)$ for any $\theta \in s$, we can then think in terms of object-oriented databases: the evaluation function e is simply a single method that encodes the semantics of the patterns. In fact, any query language, such as SQL3, that enables to query complex objects is suitable for this task. We describe the approach by means of the example of simple association rules and use self-explanatory notations when using the simple extension of the relational algebra that fits our needs. Selection of tuple and patterns are respectively denoted by σ and τ. As it is always clear from the

s_0	Frequency	Confidence
$A \Rightarrow B$	0.25	0.33
$A \Rightarrow C$	0.50	0.66
$B \Rightarrow A$	0.25	0.50
$B \Rightarrow C$	0.50	1.00
$C \Rightarrow A$	0.50	0.66
$C \Rightarrow B$	0.50	0.66
$AB \Rightarrow C$	0.25	1.00
$AC \Rightarrow B$	0.25	0.50
$BC \Rightarrow A$	0.25	0.50

(a)

s_1	Frequency	Confidence
$A \Rightarrow B$	0.33	0.33
$A \Rightarrow C$	0.66	0.66
$B \Rightarrow A$	0.33	1.00
$B \Rightarrow C$	0.33	1.00
$C \Rightarrow A$	0.66	1.00
$C \Rightarrow B$	0.33	0.50
$AB \Rightarrow C$	0.33	1.00
$AC \Rightarrow B$	0.33	0.50
$BC \Rightarrow A$	0.33	1.00

(b)

s_2	Frequency	Confidence
$B \Rightarrow C$	0.50	1.00

(c)

Instance r_0

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

(d)

Table 6.1: Patterns in three instances of an inductive database.

context, the operations can also be applied on inductive database instances while formally, we should introduce new notations for them.

Example 6.2 Mining association rules as defined above is now considered as querying inductive database instances of schema $\mathcal{R}_{\mathcal{A}} = (R, \mathcal{Q}_R, e, [0,1]^2)$. Let us consider the data set is the instance r_0 in Table 6.1(d) of the relational schema $R = \{A, B, C\}$. The inductive database $idb_1 = (r_0, s_0)$ associates with r_0 the association rules in Table 6.1(a). First, we illustrate the selection on tuples (1) and the selection on patterns (2) in the typical situation where the user defines some thresholds for frequency and confidence.

1. $\sigma_{A \neq 0}(idb_1) = (r_1, s_1)$ where $r_1 = \sigma_{A \neq 0}(r_0)$ and s_1 contains the association rules in Table 6.1(b).
2. $\tau_{e(r_0,\theta).f \geq 0.5 \wedge e(r_0,\theta).c \geq 0.7}\ (idb_1) = (r_2, s_2)$ where $r_2 = r_0$ and s_2 contains the association rules in Table 6.1(c).

□

The concept of exceptional data with respect to a pattern or a set of patterns is interesting in practice. Let us introduce the so-called *apply operation*, denoted by α, that enables to cross the boundaries between data and patterns by removing the tuples in the data set such that all the patterns are true in the new collection of tuples. In the case of association rules, assume that the truth value of a pattern θ is false in the tuple t if its left-hand side holds while its right-hand side does not hold, and true in the other cases. In terms of our current framework, it means that an association rule θ is true in a tuple $t \in r$ if and only if $e(\{t\}, \theta).f = e(\{t\}, \theta).c = 1$.

We define the result $\alpha((r, s))$ of applying α to the inductive database (r, s) to be (r', s), where r' is the greatest subset of r such that for all $\theta \in s$, we have $e(r', \theta).c = 1$. Note that $r' \setminus r$ is the collection of tuples that are exceptions to some pattern in s.

Example 6.3 Assume the instance (r_0, s_3) where s_3 only contains the rule $AC \Rightarrow B$ with frequency 0.25 and confidence 0.5. Let $(r_3, s_4) = \alpha((r_0, s_3))$. Only the tuple $\langle 1, 0, 1 \rangle$ is removed from r_0 giving r_3 since the rule $AC \Rightarrow B$ is true for the other rows. The pattern $AC \Rightarrow B$ remains the unique pattern in s_4 but its frequency and confidence in r_2 are now 0.33 and 1, respectively. □

6.2 Inductive Databases and KDD Processes

Already in the case of a single class of patterns, real-life mining processes are complex. This is due to the dynamic nature of knowledge acquisition, where gathered knowledge often affects the search process, giving rise to new goals in addition to the original ones [BA96]. In the following, we return to the scenario about telecommunication networks introduced in Example 5.6 in Section 5.2.3, this time using the framework of inductive databases.

Assume that the schema for the data part of the relevant inductive database is $R =$ *(alarm type, alarming element, element type, date, time, week, alarm severity, alarm text)*. By (r, s) we denote an inductive database for association rules where r refers to the data part and s refers to the pattern part. Here association rules are slightly different from the simple case defined in Example 6.1: items are equalities between attributes and values while right-hand side are sets of items; some example rules are given in Table 6.2. In the selection conditions we also use expressions that concern subcomponents of the rules. Typically, one wants to select rules with a given attribute on the left-hand side (LHS) or on its right-hand side (RHS), or give bounds to the number of occurring items. Self-explanatory notations are used for this purpose. A sample of an instance of this schema is given in Table 6.2.

Scenario. Let us briefly review the scenario given in Section 5.2.3 in our new context and divide it into phases. The network manager decides to look at association rules derived from r_0, the data set for the current month. He/she tunes parameters for the search by pruning out all rules that have

alarm type	alarming element	element type	date	time	week	alarm severity	alarm text
1111	E1.1	ABC	980119	233605	4	1	LINK FAILURE
2222	E2	CDE	980119	233611	4	3	HIGH ERROR RATE
3333	A	EFG	980119	233627	4	1	CONNECTION NOT ESTABLISHED
4444	B2.1	GHI	980119	233628	4	2	LINK FAILURE
...	...	...	...	...	...	...	...

s_0	Frequency	Confidence
alarm_type=1111 ⇒ element_type=ABC	0.25	1.00
alarm_type=2222 ⇒ alarming_element=E2, element_type=CDE	0.25	1.00
alarm_type=1111, element_type=ABC ⇒ alarm_severity=1, alarm_text=LINK_FAILURE	0.25	1.00
alarm_type=5555 ⇒ alarm_severity=1	0.00	0.00
...	...	...

Table 6.2: Part of an inductive database consisting of data part r_0 (upper table) and rule part s_0 (lower table).

confidence under 5% or frequency under 0.05% or more than 10 items (phase 1 in Table 6.3). The network manager then decides that attributes "alarm text" and "time" are not interesting, and projects them away (phase 2). The number of rules in the resulting rule set, s_2, is still quite large. The network manager decides to focus on the rules from week 30 and to restrict to 5 the maximum amount of items in the rule (phase 3). While browsing the collection of rules s_3, the network manager sees that a lot of rules concern the network element E. That reminds him/her of the maintenance operation and he/she decides to remove all rules that contain "alarming element = E or its subcomponent" (phase 4). (We omit the explanation of dealing with the taxonomy of components). The resulting set of rules seems not to show anything special. So, the network manager decides to compare the behavior of the network to the preceding similar period (week 29) and find out possible differences (phases 5–6). The network manager then picks up one rule (we omit the selection operation), s_8, that looks interesting and is very strong (confidence is close to 1), and he/she wants to find all exceptions to this rule; i.e. rows, where the rule does not hold (phases 7–8).

Except for the last phases, the operations are quite straightforward. In the comparison operation, however, we must first replay the phases 3–4. This is because we have to remove the field "week" from the schema we used in creating rules for week 30, so that we can compare these rules with the rules from week 29. Then we create for week 29 the same query (except for the week information), take the intersection from these two rule sets, and calculate the frequencies and confidences of the rules in the intersection. The search for exceptions is performed using the *apply operation* introduced in

Phase	Operation	Query and conditions
1	*Selection*	$\sigma_{F_1}((r_0, s_0)) = (r_0, s_1)$ $F_1 = e(\theta).f \geq 0.005 \wedge e(\theta).c \geq 0.05 \wedge \lvert LHS \rvert \leq 10$
2	*Projection*	$\Pi_T((r_0, s_1)) = (r_1, s_2)$ $T = R \setminus \{alarm\ text, time\}$
3	*Selection*	$\sigma_{F_2}(\Sigma_{C_1}((r_1, s_2)))) = (r_2, s_3)$ $C_1 = (week = 30)$ and $F_2 = \lvert LHS \cup RHS \rvert \leq 5$
4	*Selection*	$\sigma_{F_3}((r_2, s_3)) = (r_2, s_4)$ $F_3 =$ (alarming element $= E*) \notin \{LHS \cup RHS\}$
5	*Replay 3–4 (week 30)*	$\sigma_{F_3}(\sigma_{F_2}(\Pi_U(\Sigma_{C_1}((r_1, s_2)))))) = (r_3, s_5)$ $U = T \setminus \{week\}$, other conditions as in 3–4
6	*Replay 3–4 (week 29)*	$\sigma_{F_3}(\sigma_{F_2}(\Pi_U(\Sigma_{C_2}((r_1, s_2)))))) = (r_4, s_6)$ $C_2 = (week = 29)$, other conditions as in 5
7	*Intersection*	$\cap((r_3, s_5), (\emptyset, s_6)) = (r_3, s_7)$
8	*Apply*	$\alpha(r_3, s_8) = (r_5, s_9)$
9	*Optimization for phase 5*	$\Pi_V(\Sigma_{C_1}(\sigma_{F_2}(\sigma_{F_1}((r_0, s_0)))))$, $F_1 = (\lvert LHS \cup RHS \rvert \leq 5 \wedge$ (alarming element $= E*) \notin \{LHS \cup RHS\})$, $F_2 = (e(\theta).f \geq 0.005 \wedge e(\theta).c \geq 0.05)$, $C_1 = (week = 30)$, and $V = R \setminus$ {alarm text,time,week}

Table 6.3: Summary of the phases of the experiment.

Section 6.1.

Optimization Schemes. This simple scenario illustrates a typical real-life data mining task and the use of queries over an inductive database to model the analysis process. From our experience, it appears that data mining processes need frequent replays of operations and that these combinations give rise to good chances for optimization. Optimization issues become essential in the replays, since many operations require very expensive operations either on the data or the pattern part of the inductive database. Within query sequences, some of these costly operations can be avoided or the costs can be reduced significantly.

One possibility is to push some operations closer to the beginning (or to the end) of the chain, as in normal query optimization. For example, if we consider restrictions on the maximum/minimum size of rules and their attributes as soon as possible, we can reduce the number of relevant frequent item sets and thus avoid unnecessary computation of frequency and confidence values. In case of successive selections, we can combine them and possibly remove redundant phases. Furthermore, in some cases, we can choose to perform some operations or chains of operations only on data (patterns), if the original operations have been performed on patterns (data). For an example, in the query 5 in Table 6.3, we can push restriction $F_2 = |LHS \cup RHS| \leq 5$ and $F_3 = (\text{alarming element} = E*) \notin \{LHS \cup RHS\}$ as close to the beginning as possible, remove the redundant restriction $|LHS| \leq 10$ of condition F_1, move the other restrictions of F_1 to be done later, join the two projections with schemas T and U, etc. So, the resulting, more efficient replay of the same sequence of operations could be something like the optimization (phase 9) in Table 6.3.

Beyond optimization purposes, we can also use concise formal description of KDD processes to support the documentation of the process, the comparison of different processes, the reuse of previous scenarios, etc.

6.3 Discussion

Understanding the nature of the data mining process is essential also in developing useful and efficient inductive database systems. Dealing with possibly multiple pattern classes introduces new problems but also new possibilities. For instance, the network manager might first look at the source data of the alarm database, then filter association rules and based on the resulting rule set select episode rules that contain information on temporal relationships between the analyzed events (recall Figure 5.4 in Section 5.2.3).

Considering the inductive database framework from the practical point of view, one must first decide how to specify an inductive database and also evaluate queries efficiently.

Consider an inductive database for association rules. An instance consists of a current data set and a current collection of rules. Both can be materialized as SQL3 tables as in the following table.

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

body	*head*	*lbody*	e(r_0).f	e(r_0).c
$\{A\}$	$\{B\}$	1	0.25	0.33
$\{A\}$	$\{C\}$	1	0.50	0.66
$\{A, B\}$	$\{C\}$	2	0.25	1
$\{A, C\}$	$\{B\}$	2	0.25	0.5

Modifying the current data set implies a new computation of the evaluation function for the current collection of rules. The only way to modify the current collection of rules is to apply selections or set-oriented operations on it. However, a fundamental feature of inductive databases is that these selections can refer to arbitrary rules, e.g., rules whose frequencies and confidences have not been computed beforehand. The challenge for implementing an inductive database is then to use available materializations and constraints in order to compute efficiently the desired rules.

As a starting point, we have studied the `MINE RULE` operator proposed by Meo, Psaila and Ceri [MPC96, MPC98] as a concrete SQL-aware approach dedicated to association rule mining and considered its connections with our framework. Using `MINE RULE` architecture, it is possible to define the current data set, and it is possible to select patterns that hold in it and that satisfy a selection condition like a frequency or a confidence threshold, or other mining conditions. Other typical operations (e.g., union of patterns) can be simulated by SQL3 queries over the output tables and a "degenerate" `MINE RULE` statement that would just recompute frequency and confidence values for the current collection of rules using the current data set. So, the `MINE RULE` architecture can be used to simulate the closure property we defined on inductive databases. This closure property is not cosmetic: it enables the description of complex mining processes as sequences of queries. This view is an essential basis for an efficient query compilation when replays occur, a frequent situation according to our experience.

From an object-oriented viewpoint, the specific property of inductive databases is that we use some very expensive methods that can lead to database scans. A framework for query optimization has been studied for this case [Hel98] and can serve as a basis for optimization strategies. Note also the interesting work [NLHP98] that starts a systematic study of those constraints that can be used actively in order to speed up association rule mining.

Another major issue is to consider the generality of the approach. There are two aspects: specifying a new inductive database and implementing it. Indeed, an inductive database can easily be defined for other kinds of patterns than association rules: episode rules, integrity constraints in databases, classifiers, etc. The feasibility of implementation in a general case is an open problem. The most challenging problem comes from the pattern selection and can be formulated as follows: given a data set and a potentially infinite collection of patterns, how can we use the selection criteria in order to optimize the generation/evaluation of the relevant patterns.

Generic mining algorithms should also be defined. Interesting ideas come from recent generalizations of APRIORI. In [DR97] it is generalized in the context of frequent atom sets. In the article, the authors provide an inductive logic programming tool that mines the so-called frequent Datalog queries [DT98]. Article [TUA+98] consider query flocks that are parameterized Datalog (or SQL) queries for which filter condition is related to the number of parameter values. The authors propose an optimizing scheme that provides subqueries for eliminating parameter values at a cheaper price. However, a general scheme for optimization that could be orthogonal to the specification of an inductive database is still to be found.

Chapter 7

Concluding Remarks

Telecommunication networks provide both fixed and wireless connections for numerous uses. Customers require high-quality services, but at the same time network management is becoming more difficult due to the growing size and complexity of the networks. In this thesis, we have considered fault management in telecommunication networks as an example of KDD.

The networks produce large amounts of alarm information which must be analyzed and interpreted before faults can be located. Alarm correlation systems are expert systems for processing of alarm data. They are quite popular, but acquiring the knowledge necessary for constructing an alarm correlation system for a network and its elements is difficult. The complexity and diversity of network elements and the large variation in the patterns of alarm occurrences pose serious problems for network management experts trying to build a correlation model. Moreover, characteristics of the alarm flow change often, as new equipment and software releases are added to the network. On the other hand, large amounts of alarms are available, so the area is potentially good for knowledge discovery application. In this thesis we described a process from understanding the domain and preparing the data set to the utilization of the discovery results.

We proposed two methods, episode and association rules, for semi-automatic knowledge acquisition from alarm data. Episode rules describe temporal proximity and temporal ordering of recurrent combinations of alarms in a given alarm database. Association rules describe, in turn, the properties of individual alarms without taking the temporal relationships of the alarms into account. Given such rules holding in an alarm database, a fault management expert is able to verify whether the rules are useful or not. Some of the rules may reflect known causal connections, and some may be irrelevant, while some rules give new insight to the behavior of the network elements. Selected rules can be used as a basis of correlation patterns for alarm correlation systems.

Based on our experience, simple-minded use of discovery algorithms poses problems with the amount of generated rules and their relevance. Throughout this thesis, we have considered the problem of finding interesting rules from the collection of all association or episode rules. By experiments with real-life data we have demonstrated that pruning based on objective measures such as support, confidence, and J-measure thresholds is quite effective with respect to the number of rules. Still, large amounts of rules can remain even if proper thresholds for objective measures have been set. Part of these rules are redundant or nearly redundant, and can be pruned using, e.g., rule covers. Nevertheless, in the end the selection of interesting subsets from discovered rules is in the hands of the human user performing the task according to his/her subjective needs, which can be expressed using template-like mechanisms. Thus, data mining requires not only efficient algorithms but also environments, which can store and handle large data sets and rule sets efficiently, and provide appropriate tools for the user.

As one possible response to these challenges, we presented a formalization for inductive databases considering that the whole process of data mining can be viewed as a querying activity. It appears that without introducing any additional concepts, object-oriented and relational database terminology enables to carry out inductive database querying. However, the query optimization techniques for expensive selections still need much work.

A significant question is whether the inductive database framework is interesting for a reasonable collection of data mining problems. We currently study other interesting pattern classes and data mining processes that need to mine different kinds of patterns. We consider, e.g., association and episode rules in the context of alarm database analysis, and functional and inclusion dependencies in the context of database maintenance. An equally important, and partially overlapping question is to investigate how the operation set supported by the presented algebra fits in more complex scenarios. From the architectural point of view, there is clearly a challenge in identifying the border between mining systems and relational processing that should be given to SQL servers.

Clearly, the need for an inductive database like environment in the context of telecommunication databases is essential. There are lots of data in data warehouses, not only alarms and customer data but also measurement data used for monitoring and controlling the network. At the moment, measurement and alarm data are still analyzed separately. In the future, a combined analyzing system could be used to improve the analysis.

References

[AHKV97] H. Ahonen, O. Heinonen, M. Klemettinen, and A. I. Verkamo. Mining in the phrasal frontier. In J. Komorowski and J. Zytkow, editors, *Proceedings of the First European Symposium on Principles of Data Mining and Knowledge Discovery (PKDD'97)*, number 1263 in Lecture Notes in Artificial Intelligence, pages 343 – 350, Trondheim, Norway, 1997. Springer-Verlag.

[AHKV98] H. Ahonen, O. Heinonen, M. Klemettinen, and A. I. Verkamo. Applying data mining techniques for descriptive phrase extraction in digital document collections. In *Advances in Digital Libraries (ADL'98)*, pages 2 – 11, Santa Barbara, California, USA, 1998. IEEE Computer Society Press.

[AIS93] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In P. Buneman and S. Jajodia, editors, *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'93)*, pages 207 – 216, Washington, D.C., USA, 1993. ACM.

[AMS+96] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 307 – 328. AAAI Press, Menlo Park, CA, 1996.

[BA94] R. J. Brachman and T. Anand. The process of knowledge discovery in databases: A first sketch. In U. M. Fayyad and R. Uthurusamy, editors, *Knowledge Discovery in Databases, Papers from the 1994 AAAI Workshop (KDD'94)*, pages 1 – 12, Seattle, Washington, USA, 1994. AAAI Press.

[BA96] R. J. Brachman and T. Anand. The process of knowledge discovery in databases: A human-centered approach. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 37 – 57. AAAI Press, Menlo Park, CA, 1996.

[BCFK95] A. T. Bouloutas, S. B. Calo, A. Finkel, and I. Katzela. Distributed fault identification in telecommunication networks. *Journal of Network and Systems Management*, 3(3):295 – 312, 1995.

[Bia91] R. Bias. Walkthroughs: Efficient collaborative testing. *IEEE Software*, 8(5):94 – 95, 1991.

[BKM98a] J.-F. Boulicaut, M. Klemettinen, and H. Mannila. Modeling KDD processes within the inductive database framework. Technical Report C-1998-29, Department of Computer Science, P.O. Box 26, FIN-00014 University of Helsinki, Finland, 1998.

[BKM98b] J.-F. Boulicaut, M. Klemettinen, and H. Mannila. Querying inductive databases: A case study on the MINE RULE operator. In J. M. Zytkow and M. Quafafou, editors, *Proceedings of the Second European Symposium on Principles of Data Mining and Knowledge Discovery (PKDD'98)*, volume 1510 of *Lecture Notes in Artificial Intelligence*, pages 194 – 202, Nantes, France, 1998. Springer-Verlag.

[CN89] P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3:261 – 283, 1989.

[DLM+98] G. Das, K.-I. Lin, H. Mannila, G. Renganathan, and P. Smyth. Rule discovery from time series. In R. Agrawal, P. Stolorz, and G. Piatetsky-Shapiro, editors, *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD'98)*, pages 16 – 22, New York, NY, USA, 1998. AAAI Press.

[DR97] L. Dehaspe and L. D. Raedt. Mining association rules in multiple relations. In N. Lavrač and S. Džeroski, editors, *Proceedings 7th Int. Workshop on Inductive Logic Programming*, volume 1297 of *Lecture Notes in Artificial Intelligence*, pages 125–132. Springer-Verlag, 1997.

[DT98] L. Dehaspe and H. Toivonen. Frequent query discovery: A unifying ILP approach to association rule mining. Technical Report CW-258, Department of Computer Science, Katholieke Universiteit Leuven, Belgium, March 1998. Available at http://www.cs.kuleuven.ac.be/~publicaties/rapporten/CW1998.html.

[FPSM91] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus. Knowledge discovery in databases: An overview. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 1 – 27. AAAI Press, Menlo Park, CA, 1991.

[FPSS96a] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. The KDD process for extracting useful knowledge from volumes of data. *Communications of the ACM*, 39(11):27 – 34, 1996.

[FPSS96b] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery: An overview. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 1 – 34. AAAI Press, Menlo Park, CA, 1996.

[FPSSU96] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. AAAI Press, Menlo Park, CA, 1996.

[GALU95] R. M. Goodman, B. E. Ambrose, H. W. Latin, and C. T. Ulmer. NOAA – An expert system managing the telephone network. In *Integrated Network Management IV*, pages 316 – 327. Chapman & Hall, London, 1995.

[GdBV98] B. Goethals, J. V. den Bussche, and K. Vanhoof. Decision support queries for the interpretation of data mining results. Manuscript, University of Limburg (Belgium), 1998. Available at http://www.luc.ac.be/~vdbuss, 1998.

[GH98] R. D. Gardner and D. A. Harle. Methods and systems for alarm correlation. Manuscript, 1998. Available at http://voyager.eee.strath.ac.uk/~rgardner/globecom96.ps, 1998.

[GHJ+89] J. Goldman, P. Hong, C. Jeromnimon, G. Louit, J. Min, and P. Sen. Integrated fault management in interconnected networks. In B. Meandzija and J. Westcott, editors, *Integrated Network Management, I*, pages 333 – 344. Elsevier, 1989.

[Git73] I. Gitman. An algorithm for nonsupervised pattern classification. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-3:6674 –, 1973.

[GNV88] E. R. Gansner, S. C. North, and K. P. Vo. DAG — A program that draws directed graphs. *Software—Practice and Experience*, 18(11):1047 – 1062, 1988.

[Haj96] S. Hajela. HP OEMF: Alarm management in telecommunications networks. *Hewlett-Packard Journal*, 47(5), 1996. Available at http://hpcc940.external.hp.com/hpj/oct96/oc96a3.htm.

[HCC92] J. Han, Y. Cai, and N. Cercone. Knowledge discovery in databases: An attribute-oriented approach. In L.-Y. Yuan, editor, *Proceedings of the Eightteenth International Conference on Very Large Data Bases (VLDB'92)*, pages 547 – 559, Vancouver, British Columbia, Canada, 1992. Morgan Kaufmann.

[Hel98] J. M. Hellerstein. Optimization techniques for queries with expensive methods. *ACM Transaction on Database Systems*, 1998. Available at http://www.cs.berkeley.edu/~jmh/miscpapers/todsxfunc.ps.

[HF95] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In U. Dayal, P. M. D. Gray, and S. Nishio, editors, *Proceedings of the 21st International Conference on Very Large Data Bases (VLDB'95)*, pages 420 – 431, Zürich, Switzerland, 1995. Morgan Kaufmann.

[HF96] J. Han and Y. Fu. Exploration of the power of attribute-oriented induction in data mining. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 399 – 421. AAAI Press, Menlo Park, CA, 1996.

[HK91] P. Hoschka and W. Kloesgen. A support system for interpreting statistical data. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 325 – 345. AAAI Press, Menlo Park, CA, 1991.

[HKM+96a] K. Hätönen, M. Klemettinen, H. Mannila, P. Ronkainen, and H. Toivonen. Knowledge discovery from telecommunication network alarm databases. In S. Y. Su, editor, *Proceedings of the 12th International Conference on Data Engineering (ICDE'96)*, pages 115 – 122, New Orleans, Louisiana, USA, 1996. IEEE Computer Society Press.

[HKM+96b] K. Hätönen, M. Klemettinen, H. Mannila, P. Ronkainen, and H. Toivonen. TASA: Telecommunication alarm sequence analyzer, or "How to enjoy faults in your network". In *Proceedings of the 1996 IEEE Network Operations and Management Symposium (NOMS'96)*, pages 520 – 529, Kyoto, Japan, 1996. IEEE.

[HKMT95] M. Holsheimer, M. Kersten, H. Mannila, and H. Toivonen. A perspective on databases and data mining. In U. M. Fayyad and R. Uthurusamy, editors, *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD'95)*, pages 150 – 155, Montreal, Canada, 1995. AAAI Press.

[Hui78] D. H. Huizinga. A natural or mode seeking cluster analysis algorithm. Technical Report 78-1, Behavioral Research Institute, 2305 Canyon Blvd., Boulder, Colorado 80302, USA, 1978.

[IM96] T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Communications of the ACM*, 39(11):58 – 64, 1996.

[JP93] J. F. Jordaan and M. E. Paterok. Event correlation in heterogenous networks using OSI management framework. In H.-G. Hegering and Y. Yemini, editors, *Integrated Network Management, III*, pages 683 – 695. Elsevier Science Publishers B.V (North-Holland), Amsterdam, The Netherlands, 1993.

[JW93] G. Jakobson and M. D. Weissman. Alarm correlation. *IEEE Network*, 7(6):52 – 59, 1993.

[JW95] G. Jakobson and M. Weissman. Real-time telecommunication network management: Extending event correlation with temporal constraints. In *Integrated Network Management IV*, pages 290 – 301. Chapman & Hall, London, 1995.

[KK73] K. W. Kolence and P. J. Kiviat. Software unit profiles and Kiviat figures. *Performance Evaluation Review*, 2(3):2 – 12, 1973.

[Klo95] W. Kloesgen. Efficient discovery of interesting statements in databases. *Journal of Intelligent Information Systems*, 4(1):53 – 69, 1995.

[Klo96] W. Kloesgen. Explora: A multipattern and multistrategy discovery assistant. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 249 – 271. AAAI Press, Menlo Park, CA, 1996.

[KMR+94] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A. I. Verkamo. Finding interesting rules from large sets of discovered association rules. In *Proceedings of the Third International Conference on Information and Knowledge Management (CIKM'94)*, pages 401 – 407, Gaithersburg, MD, USA, 1994. ACM.

[KMT96] M. Klemettinen, H. Mannila, and H. Toivonen. Interactive exploration of discovered knowledge: A methodology for interaction, and usability studies. Technical Report C-1996-3, University of Helsinki, Department of Computer Science, P.O. Box 26, FIN-00014 University of Helsinki, Finland, 1996.

[KMT97] M. Klemettinen, H. Mannila, and H. Toivonen. A data mining methodology and its application to semi-automatic knowledge acquisition. In R. R. Wagner, editor, *Proceedings of the 8th International Workshop on Database and Expert Systems Applications (DEXA'97)*, pages 670 – 677, Toulouse, France, 1997. IEEE Computer Society Press.

[KMT99a] M. Klemettinen, H. Mannila, and H. Toivonen. Interactive exploration of interesting findings in TASA. *Information and Software Technology, Special Issue on Knowledge Discovery and Data Mining*, 1999. To appear.

[KMT99b] M. Klemettinen, H. Mannila, and H. Toivonen. Rule discovery in telecommunication alarm data. *Journal of Network and Systems Management*, 1999. To appear.

[Kol73] K. W. Kolence. The software empiricist. *Performance Evaluation Review*, 2(2):31 – 36, 1973.

[LAS97] B. Lent, R. Agrawal, and R. Srikant. Discovering trends in text databases. In D. Heckerman, H. Mannila, D. Pregibon, and R. Uthurusamy, editors, *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD'97)*, pages 227 – 230, Newport Beach, California, USA, 1997. AAAI Press.

[Man96] H. Mannila. Data mining: machine learning, statistics, and databases. In P. Svensson and J. C. French, editors, *Proceedings of the 8th International Conference on Scientific and Statistical Database Management, Stockholm*, pages 1 – 6, Stockholm, Sweden, 1996. IEEE Computer Society Press.

[Man97] H. Mannila. Inductive databases and condensed representations for data mining. In J. Maluszynski, editor, *Proceedings of the International Logic Programming Symposium (ILPS'97)*, pages 21 – 30, Port Jefferson, Long Island N.Y., USA, October 13-16 1997. MIT Press.

[MMHL86a] R. S. Michalski, I. Mozetic, J. Hong, and N. Lavrac. The AQ15 inductive learning system: An overview and experiments. Technical Report 86-20, Department of Computer Science, University of Illinois at Urbana-Champaign, USA, 1986.

[MMHL86b] R. S. Michalski, I. Mozetic, J. Hong, and N. Lavrac. The multipurpose incremental learning system AQ15 and its testing application to three medical domains. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI'86)*, pages 1041 – 1047, Philadelphia, PA, USA, 1986. Morgan Kaufmann.

[MPC96] R. Meo, G. Psaila, and S. Ceri. A new SQL-like operator for mining association rules. In T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, editors, *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB'96)*, pages 122 – 133, Mumbay, India, 1996. Morgan Kaufmann.

[MPC98] R. Meo, G. Psaila, and S. Ceri. A tightly-coupled architecture for data mining. In *Proceedings of the Fourteenth International Conference on Data Engineering (ICDE'98)*, pages 316 – 322, Orlando, Florida, USA, 1998. IEEE Computer Society Press.

[MPSM96] C. J. Matheus, G. Piatetsky-Shapiro, and D. McNeill. Selecting and reporting what is interesting. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 495 – 515. AAAI Press, Menlo Park, CA, 1996.

[MT96] H. Mannila and H. Toivonen. Discovering generalized episodes using minimal occurrences. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96)*, pages 146 – 151, Portland, Oregon, USA, 1996. AAAI Press.

[MTV95] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovering frequent episodes in sequences. In U. M. Fayyad and R. Uthurusamy, editors, *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD'95)*, pages 210 – 215, Montreal, Canada, 1995. AAAI Press.

[MTV97] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3):259 – 289, 1997.

[NLHP98] R. Ng, L. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In L. M. Haas and A. Tiwary, editors, *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'98)*, pages 13 – 24, Seattle, Washington, USA, 1998. ACM Press.

[Nyg95] Y. A. Nygate. Event correlation using rule and object based techniques. In *Integrated Network Management IV*, pages 278 – 289. Chapman & Hall, London, 1995.

[PS91] G. Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 229 – 248. AAAI Press, Menlo Park, CA, 1991.

[PSF91] G. Piatetsky-Shapiro and W. J. Frawley, editors. *Knowledge Discovery in Databases.* AAAI Press, Menlo Park, CA, 1991.

[PSM94] G. Piatetsky-Shapiro and C. J. Matheus. The interestingness of deviations. In U. M. Fayyad and R. Uthurusamy, editors, *Knowledge Discovery in Databases, Papers from the 1994 AAAI Workshop (KDD'94)*, pages 25 – 36, Seattle, Washington, USA, 1994. AAAI Press.

[PT90] F. N. Paulisch and W. F. Tichy. EDGE: An extendible graph editor. *Software—Practice and Experience*, 20(S1):63 – 88, 1990.

[RC98a] A. Ragel and B. Crémilleux. Preprocessing of missing values using robust association rules. In *Proceedings of the Second European Symposium on Principles of Data Mining and Knowledge Discovery (PKDD'98)*, volume 1510 of *Lecture Notes in Artificial Intelligence*, pages 414 – 422, Nantes, France, 1998. Springer-Verlag.

[RC98b] A. Ragel and B. Crémilleux. Treatment of missing values for association rules. In X. Wu, K. Ramamohanarao, and K. B. Korb, editors, *Research and Development in Knowledge Discovery and Data Mining, Second Pacific-Asia Conference, PAKDD-98*, volume 1394 of *Lecture Notes in Artificial Intelligence*, pages 258 – 270, Melbourne, Australia, 1998. Springer-Verlag.

[RDM+87] L. A. Rowe, M. Davis, E. Messinger, C. Meyer, C. Spirakis, and A. Tuan. A browser for directed graphs. *Software—Practice and Experience*, 17(1):61 – 76, 1987.

[SA95] R. Srikant and R. Agrawal. Mining generalized association rules. In U. Dayal, P. M. D. Gray, and S. Nishio, editors, *Proceedings of the 21st International Conference on Very Large Data Bases (VLDB'95)*, pages 407 – 419, Zürich, Switzerland, 1995. Morgan Kaufmann.

[SG92] P. Smyth and R. M. Goodman. An information theoretic approach to rule induction from databases. *IEEE Transactions on Knowledge and Data Engineering*, 4(4):301 – 316, 1992.

[ST95] A. Silberschatz and A. Tuzhilin. On subjective measures of interestingness in knowledge discovery. In U. M. Fayyad and R. Uthurusamy, editors, *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD'95)*, pages 275 – 281, Montreal, Canada, 1995. AAAI Press.

[ST96] A. Silberschatz and A. Tuzhilin. What makes patterns interesting in knowledge discovery systems. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):970 – 974, 1996.

[TKR+95] H. Toivonen, M. Klemettinen, P. Ronkainen, K. Hätönen, and H. Mannila. Pruning and grouping of discovered association rules. In G. N. Yves Kodratoff and C. Taylor, editors, *Workshop Notes of the ECML-95 Workshop on Statistics, Machine Learning, and Knowledge Discovery in Databases*, pages 47 – 52, Heraklion, Crete, Greece, 1995. MLnet.

[TUA+98] D. Tsur, J. D. Ullman, S. Abiteboul, C. Clifton, R. Motwani, S. Nestorov, and A. Rosenthal. Query flocks: A generalization of association-rule mining. In L. M. Haas and A. Tiwary, editors, *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'98)*, pages 1 – 12, Seattle, Washington, USA, 1998. ACM Press.

[Ull88] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume I. Computer Science Press, Rockville, MD, USA, 1988.

[WBES98] P.-T. P. Wu, R. Bhatnager, L. Epstein, and Z. Shi. Alarm correlation engine (ACE). In *Proceedings of the 1998 IEEE Network Operations and Management Symposium (NOMS'98)*, pages 733 – 742, New Orleans, Louisiana, USA, 1998. IEEE.

[ZZ96] R. Zembowicz and J. M. Zytkow. From contingency tables to various forms of knowledge in databases. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 329 – 349. AAAI Press, Menlo Park, CA, 1996.

Appendix A

Additional Experiments for Chapter 3

This Appendix presents some experiments with MINEPI and WINEPI to complement the ones introduced in Chapter 3.

Table A.1 summarizes the performance for serial and parallel episodes with repetition from data set 4.

Table A.2 reveals the iteration examples from experiments behind Tables 3.4 and A.1.

Figures A.1 and A.2 presents a comparison between the number of parallel and serial candidate episodes, frequent episodes and rules with window sizes 30 and 60 s, and shifts 3 and 6 s. Results in Figure A.1 are from data sets 1 and 2, and results in Figure A.2 are from data sets 5 and 6.

Figure A.3 gives the performance for serial episodes with window sizes 30 and 60 s, and shifts 3 and 6 s. Furthermore, it presents a comparison between the time used in frequent parallel and serial episode calculation.

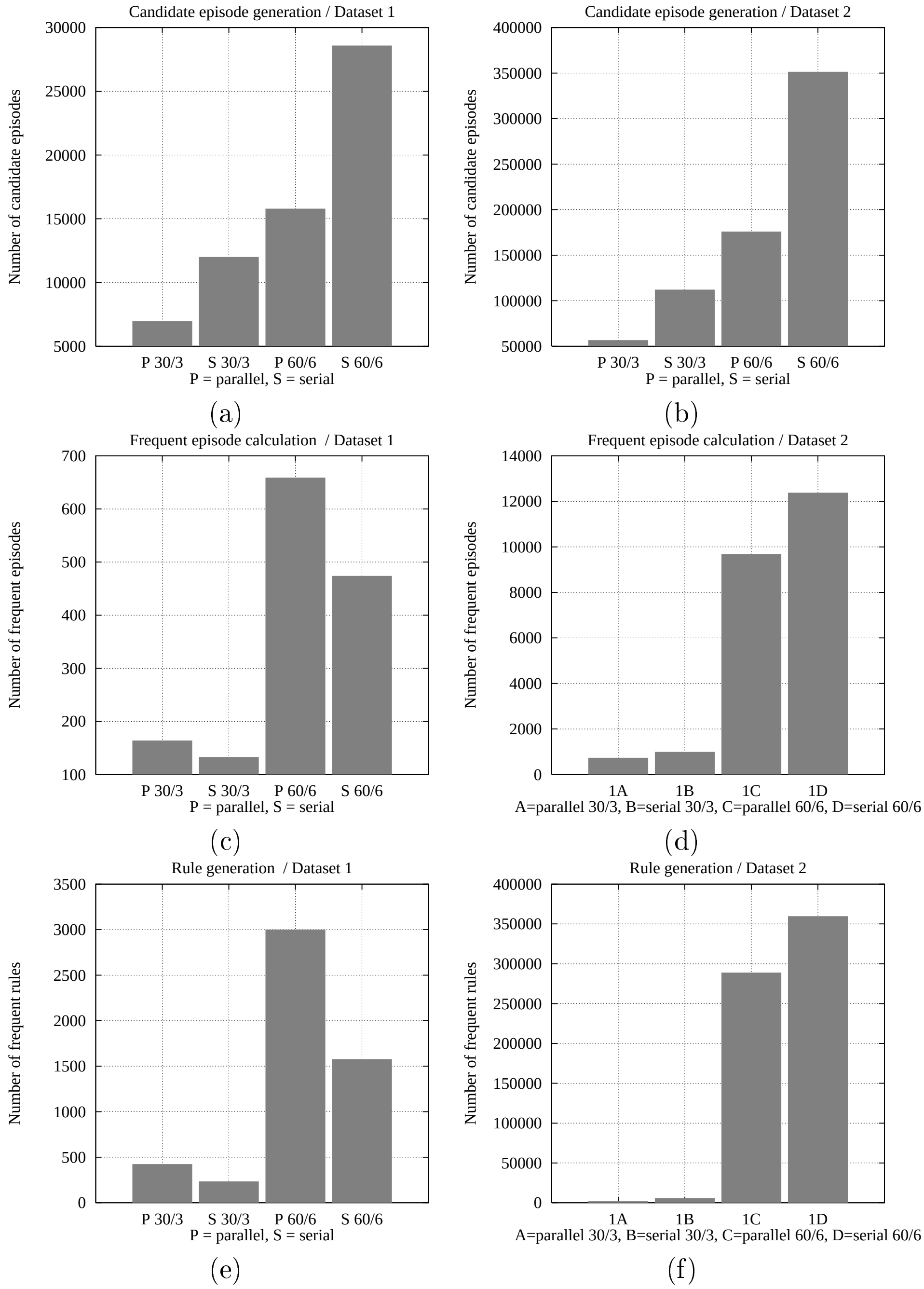


Figure A.1: A comparison between the number of parallel and serial (a-b) candidate episodes, (c-d) frequent episodes and (e-f) rules with window sizes 30 and 60 s, and shifts 3 and 6 s. Results on the left (a,c,e) are from data set 1 and on the right (b,d,f) from data set 2.

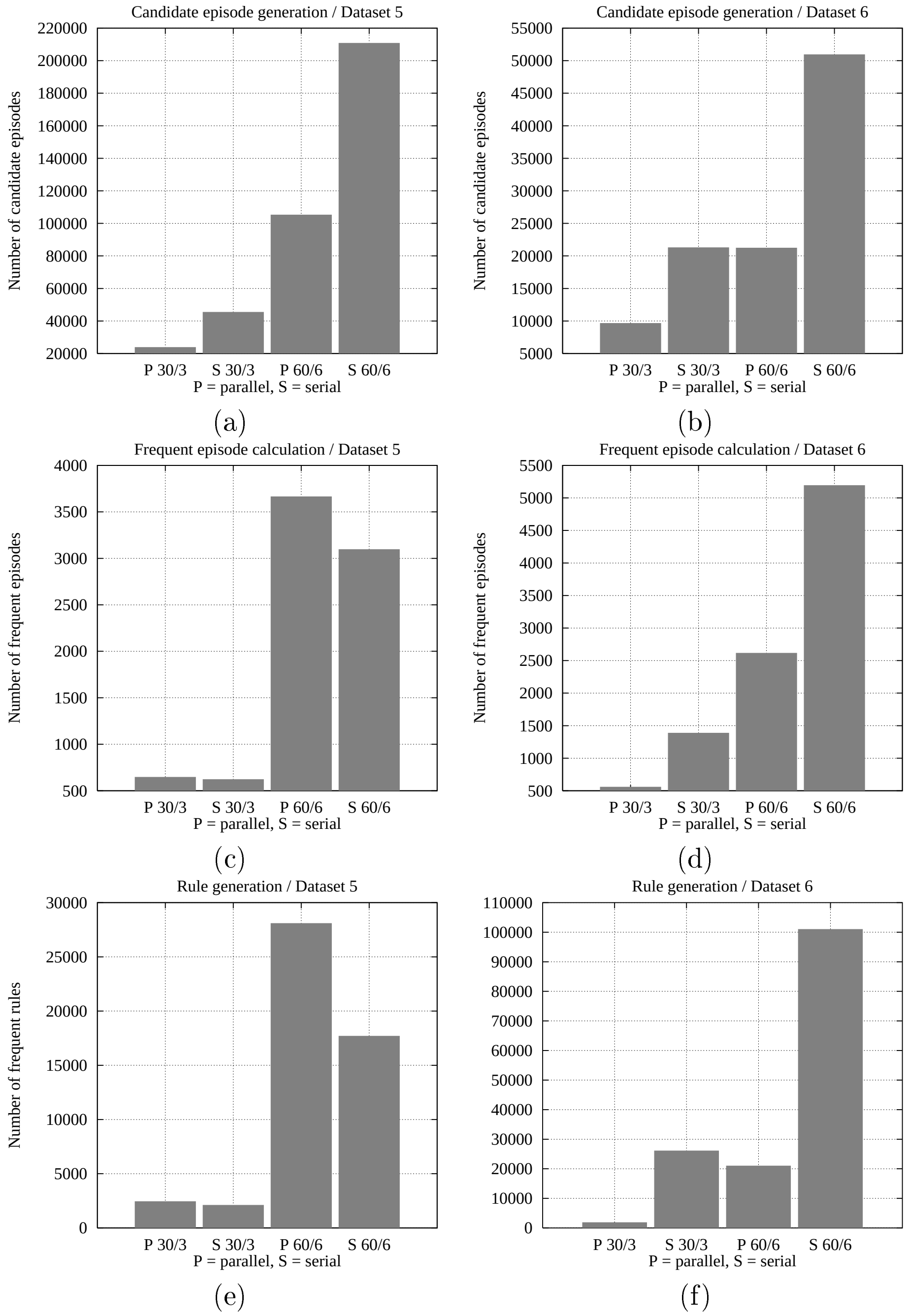


Figure A.2: A comparison between the number of parallel and serial (a-b) candidate episodes, (c-d) frequent episodes and (e-f) rules with window sizes 30 and 60 s, and shifts 3 and 6 s. Results on the left (a,c,e) are from data set 5 and on the right (b,d,f) from data set 6.

Minepi: Serial episodes with repetition, $win^* = 60$ s					
Frequency threshold	Candidates	Time (s)	Frequent Episodes	Time (s)	Iterations
10	865487	400.79	473020	192.06	11(*
20	246572	74.97	87283	114.93	100
30	139876	36.46	43416	95.03	100
40	95044	22.39	28583	83.45	100
50	67858	15.62	20684	75.84	100
100	25650	4.61	7433	57.44	100
250	7151	1.25	2018	42.17	100
500	2985	0.76	752	35.78	100
1000	1052	0.24	273	29.04	62
2000	451	0.03	72	25.95	9

(a)

Minepi: Parallel episodes with repetition, $win^* = 60$ s					
Frequency threshold	Candidates	Time (s)	Frequent Episodes	Time (s)	Iterations
10	425373	299.38	377345	185.71	100
20	148034	66.43	124964	126.03	100
30	86838	33.26	71831	103.10	100
40	58940	21.15	48195	91.71	100
50	44367	15.18	35897	84.61	100
100	17898	4.77	14192	64.78	100
250	5415	1.51	3910	47.23	100
500	2312	0.88	1465	37.95	100
1000	939	0.27	542	30.75	62
2000	436	0.03	153	26.52	9

(b)

Table A.1: Performance of the Minepi algorithm for serial (a) and parallel (b) episodes with repetition from data set 4; maximum time bound $win^* = 60$ s. (* = virtual memory exceeded.

Minepi: Serial/without/10				
It.#	Cand.	Time (s)	Freq.	Time (s)
1 :	212	0.00	100	21.80
2 :	9900	0.14	1403	12.78
3 :	22403	2.50	8028	12.97
4 :	48254	9.42	20023	14.96
5 :	60249	16.80	21462	9.72
6 :	24877	11.29	7042	2.59
...	...	...	...	...

(a)

Minepi: Serial/with/10				
It.#	Cand.	Time (s)	Freq.	Time (s)
1 :	212	0.00	100	21.56
2 :	10000	0.15	1457	13.04
3 :	25936	2.60	9762	18.94
4 :	81449	14.39	34639	30.17
5 :	174816	47.26	69243	34.78
6 :	207808	85.62	65382	25.51
...	...	...	...	...

(b)

Minepi: Serial/without/500				
It.#	Cand.	Time (s)	Freq.	Time (s)
1 :	212	0.00	27	22.58
2 :	702	0.02	75	3.89
3 :	306	0.02	165	1.82
4 :	318	0.05	28	0.92
No more frequent episodes.				

(c)

Minepi: Serial/with/500				
It.#	Cand.	Time (s)	Freq.	Time (s)
1 :	212	0.00	27	22.62
2 :	729	0.02	88	4.22
3 :	509	0.03	267	3.54
4 :	953	0.09	211	3.50
5 :	446	0.07	41	1.21
6 :	19	0.01	10	0.09
No more frequent episodes.				

(d)

Minepi: Parallel/without/10				
It.#	Cand.	Time (s)	Freq.	Time (s)
1 :	212	0.00	100	22.66
2 :	4950	0.08	1011	6.72
3 :	6828	0.71	3870	8.00
4 :	9319	1.55	7788	7.91
5 :	10579	2.61	9776	5.63
6 :	9123	2.63	8823	3.47
...	...	...	...	...

(e)

Minepi: Parallel/with/10				
It.#	Cand.	Time (s)	Freq.	Time (s)
1 :	212	0.00	100	21.40
2 :	5050	0.08	1065	6.99
3 :	8523	0.68	4892	11.92
4 :	17911	2.51	14044	17.43
5 :	34363	7.62	28830	21.09
6 :	52538	16.09	45745	22.60
...	...	...	...	...

(f)

Minepi: Parallel/without/500				
It.#	Cand.	Time (s)	Freq.	Time (s)
1 :	212	0.00	27	22.48
2 :	351	0.01	89	2.52
3 :	167	0.02	133	1.92
4 :	107	0.02	104	1.05
5 :	46	0.03	46	0.38
6 :	13	0.01	13	0.09
7 :	2	0.00	2	0.01

(g)

Minepi: Parallel/with/500				
It.#	Cand.	Time (s)	Freq.	Time (s)
1 :	212	0.00	27	21.75
2 :	378	0.01	102	2.81
3 :	305	0.03	220	3.59
4 :	411	0.05	302	3.75
5 :	381	0.07	311	2.80
6 :	294	0.08	241	1.68
...	...	...	...	...

(h)

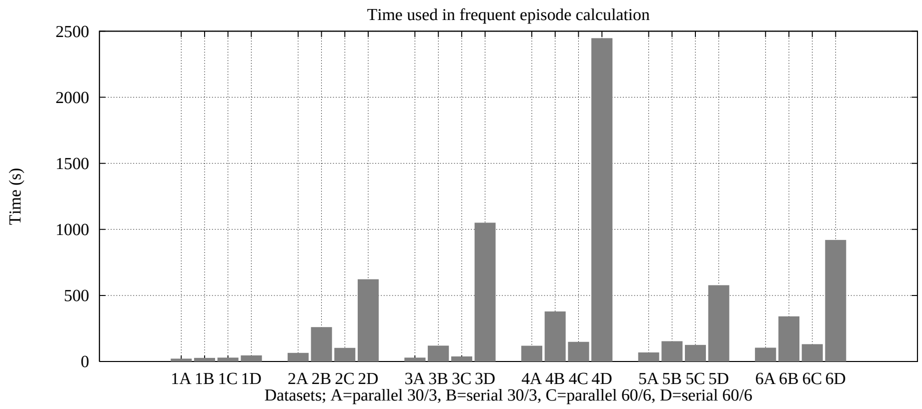
Table A.2: Iteration examples from experiments behind Tables 3.4 and A.1.

WINEPI: SERIAL EPISODES WITH WINDOW SIZE 30 S AND SHIFT 3 S							
#	Frequency threshold	Candidate episodes	Time (s)	Frequent episodes	Time (s)	Rules (distinct)	Time (s)
1	0.0045	12003	0.18	133	27.35	235	19.33
2	0.0005	112244	1.62	996	260.31	5792	20.30
3	0.0001	3672	0.29	1588	120.23	50044	29.60
4	0.0001	4170	0.18	537	378.82	3565	19.48
5	0.0005	45546	0.71	624	153.43	2126	19.72
6	0.0001	21305	0.53	1390	341.86	26168	24.51

(a)

WINEPI: SERIAL EPISODES WITH WINDOW SIZE 60 S AND SHIFT 6 S							
#	Frequency threshold	Candidate episodes	Time (s)	Frequent episodes	Time (s)	Rules (distinct)	Time (s)
1	0.0045	28587	0.50	474	46.39	1578	19.63
2	0.0005	351436	7.32	12380	622.52	359662	96.17
3	0.0001	32497	4.15	23492	1050.78	997388	253.63
4	0.0001	22412	2.02	4885	2447.68	73279	33.18
5	0.0005	210902	5.04	3098	577.90	17716	21.88
6	0.0001	50974	2.11	5195	920.41	101039	38.47

(b)



(c)

Table A.3: Performance of the WINEPI algorithm for serial episodes: (a) window size 30 s, shift 3 s, and (b) window size 60 s, shift 6 s, and (c) a comparison between the time used in frequent parallel and serial episode calculation.

Appendix B

Additional Experiments for Chapter 4

This Appendix presents some experiments with Apriori to be compared with the results with Winepi and Minepi.

Figure B.1 illustrates the effect of rule frequency, confidence, and J-measure thresholds to the number of parallel association rules with Apriori in the example alarm data sets. Due to the rule set sizes of data sets 5–6, we present the results from data sets 1–3 and 5–6 separately. Furthermore, due to the deliberately faulty preprocessing, we here omit the results from data set 4.

Figures B.2, B.3 and B.4 present the relationships between confidence and J-measure, frequency and J-measure, and confidence and frequency in parallel association rules with Apriori.

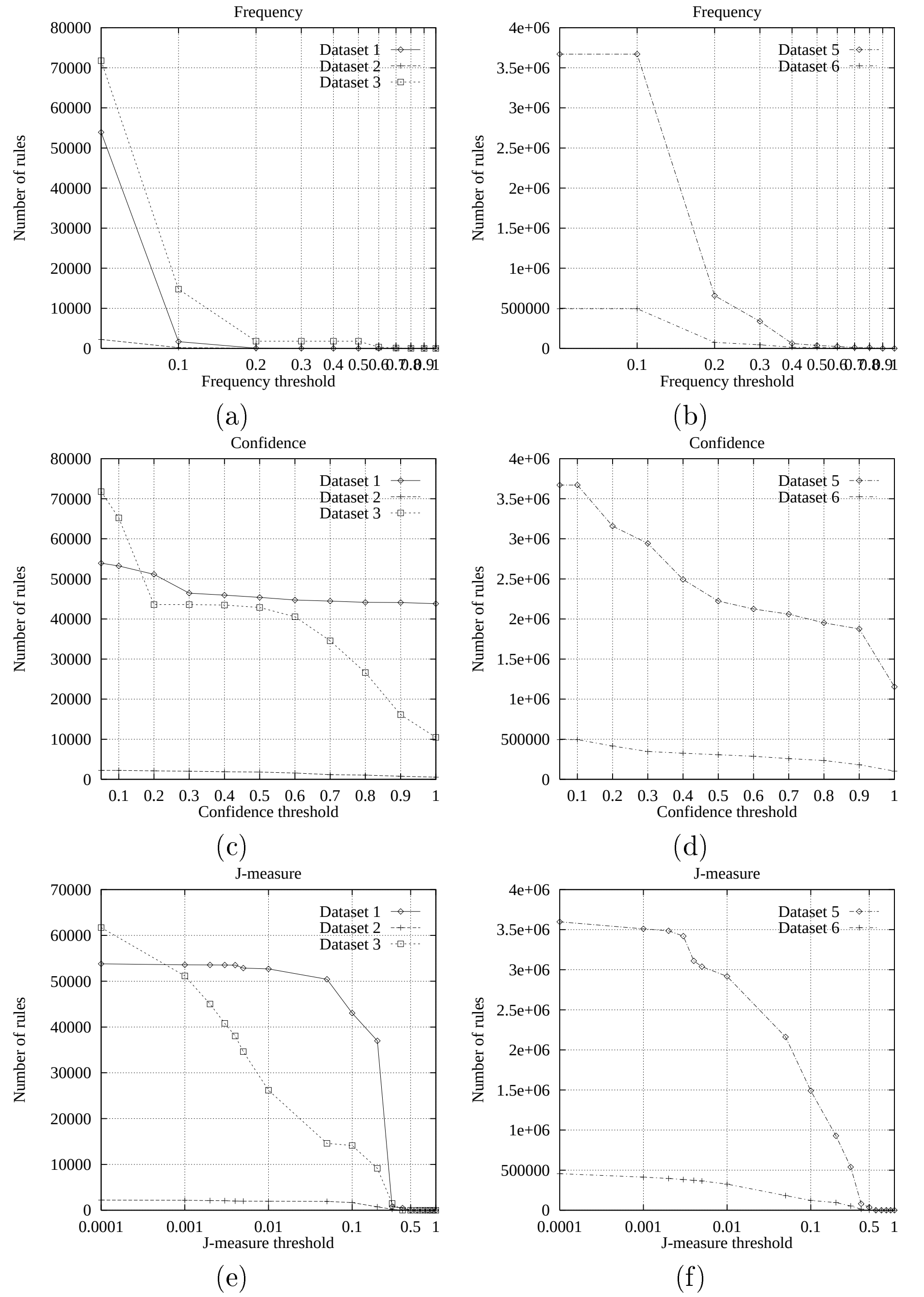
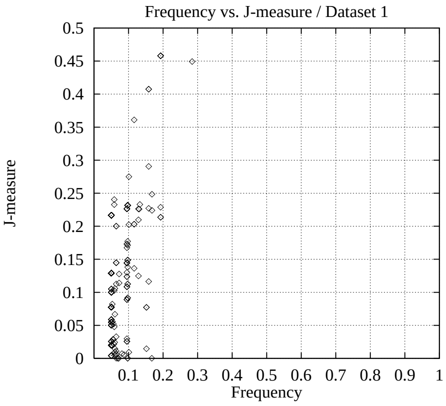
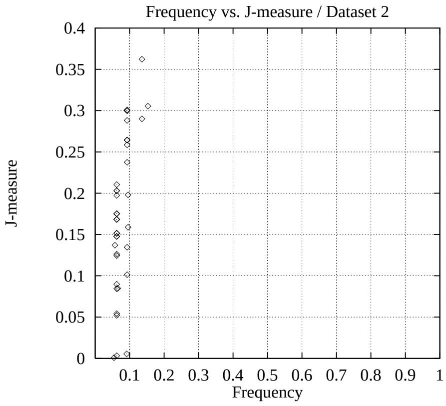


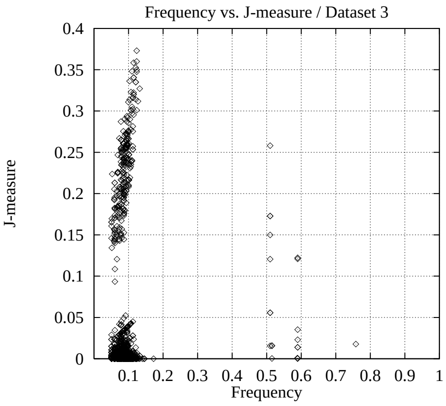
Figure B.1: Effect of (a-b) rule frequency, (c-d) confidence, and (e-f) J-measure thresholds to the number of parallel association rules with APRIORI in the example alarm data sets. Results on the left (a,c,e) are from data sets 1–3 and on the right (b,d,f) from data sets 5–6.



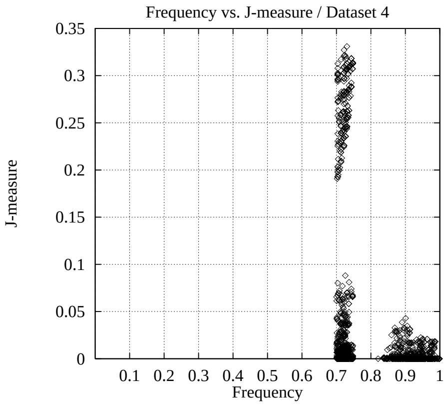
(a)



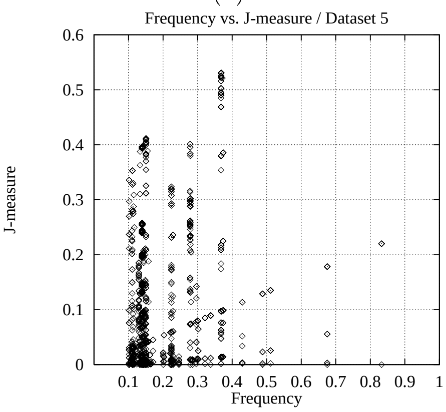
(b)



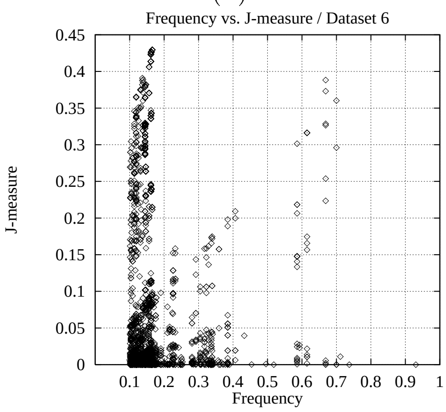
(c)



(d)



(e)



(f)

Figure B.2: A comparison between frequency and J-measure in parallel APRIORI rules from alarm data sets 1–6 with window 60 s and shift 6 s.

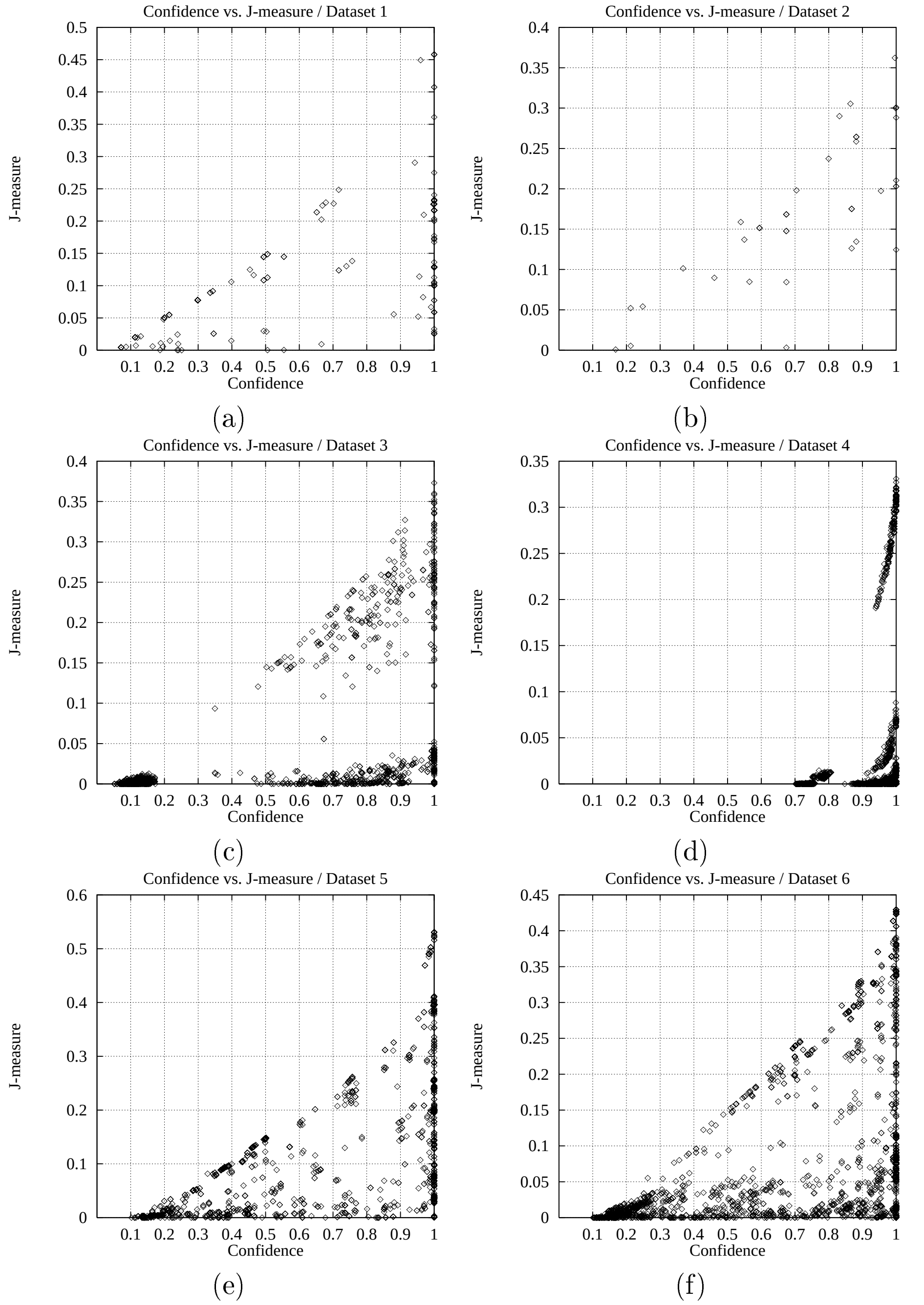


Figure B.3: A comparison between confidence and J-measure in parallel APRIORI rules from alarm data sets 1–6 with window 60 s and shift 6 s.

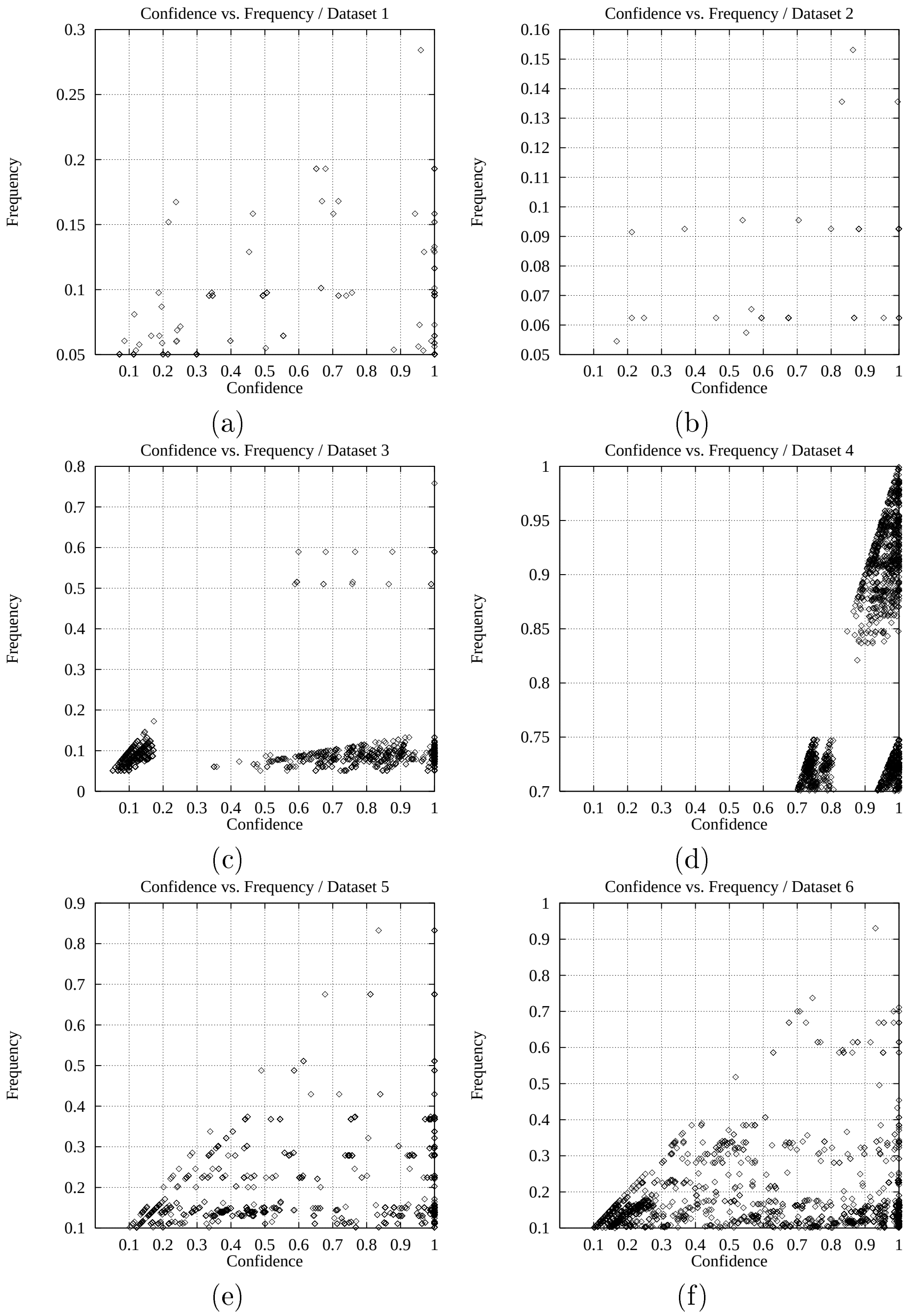


Figure B.4: A comparison between confidence and frequency in parallel APRIORI rules from alarm data sets 1–6 with window 60 s and shift 6 s.

Epilogue

Mikä on yhteistä kalalle
trumpetille ja piipulle?
entäpä mitä on hyvyys ja kampela?
asioitapa tietenkin ovat ne

Mikä kumma on munakas
tuo munista tehty hassu
asiahan se on ja hiukan litteä
ja pehmeä asia taas on töyhtö

Asioita ovat esimerkiksi juusto
ja pelikaanikin on asia
muita asioita ovat tursas ja viisumi
maailma on asioita pullollaan

Laulu asioista, © Janne Saarikivi

ISSN 1238-8645
ISBN 951-45-8465-1
Helsinki 1999
Helsinki University Printing House

Mika Klemettinen: A Knowledge Discovery Methodology for Telecommunication Network Alarm Databases A-1999-1