ACTIVE AND MOBILE DATA MANAGEMENT THROUGH EVENT HISTORY MINING

A DISSERTATION SUBMITTED TO

THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BİLKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

By

Yücel Saygın

August 2001

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Assoc. Prof. Özgür Ulusoy (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Prof. Adnan Yazıcı

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Asst. Prof. Attila Gürsoy

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Prof. Cevdet Aykanat

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Asst. Prof. Nail Akar

Approved for the Institute of Engineering and Science:

Prof. Mehmet Baray
Director of the Institute

ABSTRACT

ACTIVE AND MOBILE DATA MANAGEMENT THROUGH EVENT HISTORY MINING

Yücel Saygın
Ph.D.in Computer Engineering
Supervisor: Assoc. Prof. Özgür Ulusoy
August 2001

An event history is a collection of events that have occurred in an event-based system over a period of time. There can be various types of events, among which are temperature changes and power demands in a power management system, client requests for data items in a data broadcast system, price increase of a stock in a stock market, and so on. There is a lot of interesting information that can be extracted from an event history via data mining techniques. Our purpose in this thesis is to propose methods for extracting this useful information in the form of event sequences and event associations from a single or two correlated event histories. We also aim to show how the results of the mining process can be used for active and mobile data management. The results of the mining process demonstrate the relationships among the events which are generally captured as associations or sequences. The relationships among the events are shown to be a useful tool to enhance an event-based system via event organization, predictive event detection, and proactive rule execution.

We consider the mining of both a single event history and two correlated event histories. We first propose a method for mining binary relationships from a single event history. The binary relationships among events are used to organize the events into related groups of event. This organization is important because the number of events in an event-driven system may become very high and unmanageable. The groups of related events and the relationships among the events are exploited for predictive event detection and proactive rule execution in active database systems. We also consider the mining of two correlated event histories which are disjoint and the events in one history are related to the events in the other history. We describe how we can efficiently extract associations among the

events spanning different event histories, which we call *cross associations*.

We have chosen data broadcast in mobile computing environments as a case study for active data management. One of the important facts in mobile computing environments with wireless communication medium is that the server-to-client (downlink) communication bandwidth is much higher than the client-to-server (uplink) communication bandwidth. This asymmetry makes the dissemination of data to client machines a desirable approach. However, the dissemination of data by broadcasting may induce high access latency in case the number of broadcast data items is large. Our purpose is to show how the features of active data management can be used to improve mobile data management through broadcast data organization and prefetching from the broadcast medium. In order to achieve this, the client requests of data items at the server are considered as events and the chronological sequence of items that have been requested by clients is considered as an event history. An event history in broadcast medium is called a broadcast history. The first step in this work is to analyze the broadcast history to discover sequential patterns describing the client access behavior. The sequential patterns are used to organize the data items in the broadcast disk in such a way that the items requested subsequently are placed close to each other. Then, we utilize predictive event detection techniques to improve the cache hit ratio to be able to decrease the access latency. Prediction of future client access behavior enables clients to prefetch the data from the broadcast disk based on the rules extracted from the broadcast history.

Keywords: Data mining, event history mining, correlated histories, cross associations, active database systems, predictive event detection, proactive rule execution, fuzzy event sets, fuzzy triggers, fuzzy rule execution, similarity based event detection, broadcast histories, mobile databases, prefetching, broadcast organization.

ÖZET

AKTİF VE HAREKETLİ VERİLERİN OLAY GEÇMİŞLERİNİN ANALİZİ YOLUYLA YÖNETİMİ

Yücel Saygın
Bilgisayar Mühendisliği, Doktora
Tez Yöneticisi: Doç. Dr. Özgür Ulusoy
Ağustos 2001

Olay geçmişleri, olay tabanlı sistemlerde görülen olayların bir zaman dilimi içerisinde toplu olarak tutulduğu yerdir. Bir sistemde birçok olay bulunabilir, bunlara örnek olarak, güç yönetimi sistemlerinde sıcaklık değişimleri ve elektrik talepleri, veri yayın sisteminde verilere erişim işlemleri, borsada belirli hisse senetlerinin artışı gösterilebilir. Veri analizi teknikleriyle olay geçmişleri analiz edilerek birçok faydalı bilgi elde edilebilir. Bu tezde amacımız, tek ya da birbiriyle ilgili birden fazla olay geçmişinin analiz edilmesiyle olay dizilerinin ve olay ilintilerinin çıkarılmasıdır. Bunun yanında diğer bir hedefimiz de olay geçmişi analizi yoluyla elde edilen olay dizi ve ilintilerinin aktif ve hareketli veri yönetiminde nasıl kullanılabileceğini de gösterilmesidir. Olay geçmişi analizi sonucu elde ettiğimiz bilgiler, genelde ilinti veya dizi şeklinde gösterilebilir. Olaylar arası ilişkilerin, olay-tabanlı sistemlerin olaylarının organize edilmesi, olayların önceden tahmin edilmesi ve kuralların önceden aktive edilmesi yoluyla geliştirilmesinde çok yararlı olduğu çalışmalarımızda tesbit edilmiştir.

Bu tezde, hem tek hem de birden fazla birbiriyle ilişkili olay geçmişlerinin analizi üzerinde çalışılmıştır. Öncelikle tek bir olay geçmişinin analizi konusu ele alınmış ve böyle bir olay geçmişinden ikili olay ilişkilerinin çıkarılması için bir yöntem geliştirilmiştir. Olayların arasındaki bu ikili ilişkiler, olayların gruplandırılması için kullanılmıştır. Bir sistemdeki olayların gruplandırılması önemlidir, çünkü sistemdeki olay sayısı çok artabilir ve bu durum olayların yönetimini zorlaştırır. Çalışmamızda, birbiriyle ilişkili olay grupları ve olaylar arasındaki ilişkiler aktif veri tabanı sistemlerinde tahmini olay saptaması ve kuralların önceden aktive edilmesi için kullanılmıştır. Birbiriyle ilişkili iki olay geçmişinin analizi üzerinde de çalışmalar yapılmıştır. Birbiriyle ilişkili olay geçmişlerinde iki

ayrık olay kümesi vardır ve bir olay geçmişindeki olaylar, öteki olay geçmişindeki olaylarla ilgilidir. Tezimizde, iki olay geçmişini kapsayan olaylar *çapraz ilintiler* olarak adlandırılmış ve çapraz ilintilerin etkili bir şekilde nasıl çıkarılabileceği anlatılmıştır.

Çalışmamızda, aktif veri yönetimi için, hareketli (mobil) sistem ortamlarında veri yayını konusu araştırma alanı olarak seçilmiştir. Hareketli sistem ortamlarında önemli bir gerçek, sunucudan-istemciye, yani aşağı-hat, iletişim kapasitesinin, istemciden-sunucuya, yani yukarı-hat, kapasitesinden çok daha yüksek olmasıdır. Bu asimetri, verilerin yayınının arzu edilen bir veri iletme yöntemi olmasını sağlar. Öte yandan, verilerin yayın yoluyla iletilmesi, verilerin çok olması durumunda uzun bekleme sürelerine sebep olur. Bu tezde amaçlarımızdan birisi, hareketli verilerin yönetiminin, yayınlanan verilerin organize edilmesi ve verilerin önceden istemci hafızasına alınması yoluyla iyileştirilmesidir. Bunu sağlamak için, istemciden sunucaya yapılan veri istekleri birer olay olarak düşünülmüş ve bu şekilde istemciden gelen isteklerin diziminden bir olay geçmişi oluşturulmuştur. Yayın ortamında istemciden gelen veri isteklerinin tutulduğu olay geçmişleri, çalışmamızda yayın geçmişleri olarak adlandırılmıştır. İlk olarak, yayın geçmişi analiz edilerek istemcilerin veri ulaşım davranışlarını gösteren sıralı paternler keşfedilmiştir. Elde edilen sıralı paternler, yayın diskindeki verilerin organize edilmesi amacıyla kullanılmıştır. Burada önemli olan, yayın diskindeki verilerin aynı anda sıralı olarak istenen veriler birbirine yakın olacak şekilde yerleştirilmesidir. Daha sonra, tahmini olay saptaması teknikleri kullanılarak istemci hafızasına yerinde ulaşım oranının iyileştirilmesi yoluyla istemcilerin veri ulaşım zamanının azaltılmasına çalışılmıştır. Olayların tahmini saptanması, istemcilerin yayın diskinden verileri önceden hafızalarına olay geçmişinden çıkarılan kurallar yardımıyla yüklemeleri için kullanılmıştır.

Anahtar sözcükler: veri analizi, olay geçmişi analizi, birbiriyle ilişkili geçmişler, çapraz ilintiler, aktif veri tabanı sistemleri, tahmini olay saptaması, kuralların önceden aktive edilmesi, bulanık olay kümeleri, bulanık tetikler, bulanık kural uygulaması, yakınlık derecesine göre olay saptaması, yayın geçmisleri, hareketli veri tabanları, önceden hafızaya yükleme, yayın organizasyonu.

Acknowledgment

First, I would like to thank my supervisor Özgür Ulusoy for his priceless guidance over the 7 years of my graduate studies. I would also like to thank the committee members, Prof. Cevdet Aykanat, Prof. Adnan Yazıcı, Asst. Prof. Nail Akar, and Asst. Prof. Attila Gürsoy for reading and commenting on my thesis. Our discussions with Bora Uçar and Prof. Cevdet Aykanat on the hypergraph models and the algorithms we used were especially very helpful. A course taught by Prof. Adnan Yazıcı established the necessary fuzzy database background which was very important for my thesis. Asst. Prof. Nail Akar was very kind to accept being in my committee and had very useful comments. During her short trip to Ankara, Özlem Öğüt read the initial version of my thesis (i.e., the abstract) and provided very constructive feedback.

Part of my research was conducted at Purdue University under the supervision of Prof. Ahmed Elmagarmid. The endless experience of Prof. Ahmed Elmagarmid in research contributed a lot to my academic vision. I was very lucky to meet and conduct joint research with Prof. Elisa Bertino from University of Milan, during her visit to Purdue. During my visit to the University of Florida at Gainesville, I had a chance to have very helpful discussions with Assoc. Prof. Abdelsalam Helal on my thesis.

Last but not least I would like to thank my family and all my friends including the Olympos Tree-house Community for their moral support.

To my family, friends, and of course Sheeba

Contents

List of Figures

List of Tables

Chapter 1

Introduction

An *event history* is a collection of events that have occurred in an event-based system over a period of time. Events in a history are recorded in the order of occurrence together with timestamps that capture the day, hour or a finer granularity time unit supported by the system. Relationships among the events can be captured from event histories and they can be used to enhance the performance of event based systems. In this thesis, we demonstrate how event histories and the relationships among the events can be used for event organization, predictive event detection, and proactive rule execution in active database systems. We further show how the concepts of event organization, predictive event detection and proactive rule execution can be used for broadcast data organization and prefetching in mobile computing environments.

1.1 Event History Analysis

Event histories may consist of events from the same domain like weather events, or there may be multiple correlated event histories where the events in each history correspond to different domains like weather events and power demand events of companies. A single event history can be divided into multiple correlated event histories when domain knowledge is available and able to specify what group of

events belong to what domain.

There might be different types of event patterns (relationships among events) that can be extracted from an event history, namely *associations, cross associations*, or *sequences*. In a single event history, an *association* between two events implies that their occurrences in the event history are somehow related. For example, a price increase of a stock in a stock market may be associated with a price increase of another stock. We propose a sliding window algorithm to obtain the similarities among events in a single history. We also describe how a broadcast event history viewed as a single event history can be analyzed to extract client request event patterns. Client request event patterns are expressed as *sequences*. A sequence is a special case of an event pattern where the events have an ordering in the pattern. We define the relationship among two events as a *cross association* when the associated events belong to different event histories. Cross associations exist when there are two or more histories, and events in one of those histories are related to the events in another history. We call such types of histories *correlated event histories*. As an example, it can be claimed that weather and power demand histories in a power management system are correlated, because the weather events such as temperature changes can affect the power demand of households or companies. In this thesis, we provide a framework for cross associations and multiple correlated event histories. We also propose two data mining algorithms to extract cross associations from correlated event histories which exploit early pruning strategies. The first algorithm is based on merging and then mining the correlated event histories. The second algorithm analyses the two histories concurrently. It uses early support and confidence pruning strategies to reduce the number of candidates and the number of database scans. We also evaluate the performance of the proposed algorithms on synthetic data.

1.2 Organization of Events into Fuzzy Sets

Modularization of rules is an important research issue in active database systems. In systems where thousands of events may occur and thousands of rules may

be fired by these events, it is really very difficult to keep track of the rules. Therefore, partitioning the whole event space into smaller groups would be helpful in controlling the rules fired by those events. Partitioning the event space enables the users of the system to concentrate on a smaller group of related events, increasing the efficiency and effectiveness of the system.

Consider the demand side management of an electricity producing and selling company where the power consumption information of individual companies is stored in a history. Consider also that there are hundreds of companies, which is usually the case, and each company has more than two power consumption events, like low, medium, and high power consumption. The number of events in the system can easily become unmanageable. If there were a way to divide the whole event space into subsets and deal with each subset individually, the job of the power dealers would be much easier.

Classification of rules by using semantic constraints in an active database system was considered before by Baralis et al. in [BCP96]. This process is called *stratification*. Event-based stratification techniques aim to modularize the rules by classifying the event space which is done by human interaction and can not be used when semantic knowledge of events is not available. The rule modularization approach of Baralis et al. is based on semantic techniques and it is not automated. However, it is always desirable to automate this process whenever possible. In this thesis, we deal with the problem of automated rule modularization by partitioning the events which trigger the rules. Our model for modularizing rules is based on the analysis of past occurrences of events which are stored in a time based event history.

We use graph partitioning techniques for dividing the event space using event relationship matrix into subsets. These subsets are modeled as fuzzy sets, therefore we call them fuzzy event sets. Each event has a degree of membership to a fuzzy event set. Similarity based event detection was proposed as a useful method in an event based system [SUY99]. Fuzzy event sets are considered to be the basis of similarity based event detection and fuzzy rule execution in active

database systems. In this thesis, we propose a method for extracting the similarities from event histories without the domain knowledge of experts. By utilizing fuzzy event sets, we introduce the benefits of fuzzy set theory to active database rule modularization and rule execution. We also describe a new rule execution model based on fuzzy event sets which gives more flexibility to rule execution.

1.3 Proactive Rules for Data Broadcast

Cross associations, or sequences extracted from event histories are used for *predictive event detection* which enables us to handle events prior to their occurrence. Knowing that two events are related (by association, cross association, or sequence) and one of the events' occurrence can be predicted, it is possible to take an immediate action for the other event which is also expected to occur. As an example, suppose that a high temperature is cross-associated with a high electricity demand of households, and also weather events like temperature changes are predictable. So, if a high temperature is predicted for the next day, the power generation company can take actions to regulate the electricity demand by using intelligent power meters that can make selling and buying decisions. Predictive event detection invokes the notions of *proactive event handling* and *proactive rule execution*. We provide a mechanism to anticipate future events and handle these events prior to their occurrence. Database systems that are characterized by proactive event handling can be called *proactive database systems*.

Broadcast data management in mobile computing environments is a very good application of event organization, predictive event detection and proactive rule execution. Broadcasting has become a very popular way of information dissemination with the advances in mobile computer and wireless technology. However, unlike on-demand data service, broadcast environments introduce high access latency for clients. This is due to the fact that broadcast forms a uni-directional stream of data in the air like a tape, and clients need to wait for the particular data of their interest to appear in the broadcast. With intelligent broadcast data organization and prediction of future client requests, we aim to decrease this

latency.

The broadcast requests issued over time in a mobile database environment can be stored in a special type of event history called *broadcast history* where the events are client requests for broadcast data items. In the broadcast history, a lot of useful information about the broadcast request patterns and their relative issuing times are hidden. Our methods for broadcast organization and predictive detection of future client request events are based on analyzing the broadcast history using data mining techniques to extract client request event patterns. We are interested in the extraction of sequences of client data accesses, as well as clustering the data items. We discuss how we can exploit sequential rules for organizing data broadcast for efficient data access by mobile clients. The broadcast data is viewed as a set of events and it is organized by using the techniques proposed for event organization. Hypergraph partitioning methods based on sequential patterns are used for clustering data items. Predictive event detection in the context of data broadcast is supported by the sequential rules obtained from sequential patterns. Predictive event detection is used for predicting the future client request events. The rules are made available to mobile clients through broadcasting and they establish a rule base for predictive prefetching. Prefetching is performed by proactive rule execution. The data items that are going to be accessed in the near future are loaded to the cache proactively, before they are actually requested. It is shown through performance experiments that event based broadcast organization and prefetching in mobile computing environments improve client waiting time better than the state of the art prefetching methods.

1.4 Summary of Contributions

The main contributions of this thesis can be summarized as follows:

- Analysis of correlated event histories and development of efficient algorithms for extracting cross associations from correlated event histories;
- Organization of events in an event based system into fuzzy event sets using

graph partitioning techniques;

- Development of a framework for predictive event detection, and proactive rule execution in active database systems;
- Supporting predictive event detection and proactive rule execution in active database systems with event patterns extracted from a single event history or correlated event histories; and
- Application of event organization, predictive event detection, and proactive rule execution to broadcast data management to improve the client waiting times.

1.5 Organization of The Thesis

We provide the background and related work in Chapter 2 where the concepts we deal with in this thesis, such as data mining, active database systems, broadcast environments, and fuzzy set theory are introduced. The work done in mining a single event history is described in Chapter 3. In Chapter 4, we extend the event history concept to multiple correlated event histories and develop a framework for mining two correlated event histories. The results of an extensive performance study conducted to evaluate the effectiveness of the proposed methods are provided in the same chapter. Fuzzy and proactive rule execution as an extension to the standard active rule execution is introduced in Chapter 5. In Chapter 6, we present how predictive event detection and proactive rule execution techniques can be used in broadcast data management in mobile computing environments. The performance impact of loading data items to client cache by predictive detection of client data request events and proactively loading the items to client cache is evaluated through experiments on a web log. Finally, Chapter 7 summarizes the conclusions drawn from this thesis and proposes some future research directions in the context of this work.

Chapter 2

Background and Related Work

Data mining is the most relevant research area to our work. We introduce the related work performed in the context of data mining in Section 2.1. Event histories are introduced in Section 2.2. We use the patterns extracted by mining the event histories for active and mobile data management. Active data management is achieved by utilizing event patterns for event organization and fuzzy rule execution as well as proactive event handling. Mobile data management is performed by using the patterns of client request events to data items for organizing the broadcast data and loading the data items to client cache proactively. We describe the fuzzy set theory and fuzzy inferencing methods in Section 2.3, and the basic concepts in s are explained in Section 2.4. Proactive event handling and event organization techniques are considered to be a useful tool in broadcast data management in mobile computing environments. We describe the basics of broadcast data management in Section 2.5.

2.1 Data Mining

Concerning the area of data mining, the research issues related to our work include association mining, sequence mining, and proactive databases. The problem of finding association rules among items is formally defined by Agrawal et al.

in [AS94] as follows:

Definition 2.1.1 *Let $I = \{i_1, i_2, ..., i_m\}$ be a set of literals, called items and D be a set of transactions, i.e., $\forall T \in D, T \subseteq I$. An association rule is denoted by an implication of the form $X \Rightarrow Y$, where $X \subset I$, $Y \subset I$, and $X \cap Y = \emptyset$. A rule $X \Rightarrow Y$ is said to hold in the transaction set D with confidence c if $c\%$ of the transactions in D that contain X also contain Y. The rule $X \Rightarrow Y$ has support s in the transaction set D if $s\%$ of transactions in D contain $X \cup Y$.*

The process of finding association rules is organized into two steps:

1. finding the large itemsets (i.e., the set of items with support greater than the user specified minimum support threshold),
2. finding the association rules using the large itemsets.

The second step is trivial compared to the first step, therefore association rule finding algorithms concentrate on finding the large itemsets. *Apriori*, introduced by Agrawal et al. in [AS94], is the fundamental algorithm for mining associations from a database of customer transactions. Apriori is based on the fact that all the subsets of a large itemset are also large. Using this principle, large itemsets of size k are found only by using the large itemsets of size $k-1$.

Dynamic Itemset Counting (DIC) algorithm proposed by Brin et al. is another algorithm for finding large itemsets [BMUT97]. The work presented by these authors also includes efficient ways for generating implication rules out of the large itemsets. An online association rule finding algorithm proposed by Hidber allows the user to change the support threshold during the mining process [Hid99]. Mining of multiple-level association rules was studied by Han and Fu [HF99]. Multiple-level association rules are obtained by analyzing the data at multiple levels of abstraction. A recent method for finding large itemsets without candidate generation was proposed by Han et al. [HPY00]. An index

structure was provided in that work for storing large itemsets and for counting their supports. Mining frequent patterns using support constraints was described in [WHH00]. Mining of quantitative association rules which is based on mining numerical attributes in large relational databases was studied by Srikant and Agrawal [SA96]. The approach proposed in that work is based on fine partitioning the values and then combining the adjacent partitions. In our case, we are also dealing with quantitative attributes like temperature, humidity, and power consumption. However, our main focus is on mining two correlated histories.

Sequences are obtained when the events are recorded together with their occurrence time. In general, a sequence s of size n consists of n elements, $s_1, s_2, ..., s_n$ where s_i occurs before s_j in the history if $i < j$. Mining sequences in customer transactions was investigated by Agrawal and Srikant [AS95]. Sequences in their work are in the form $(s_1, s_2, ...s_n)$ where s_i, $i = 1, \ldots, n$, is an itemset. The authors proposed an algorithm that first finds the large itemsets in the data and then generates sequences by trying to combine these large itemsets. Properties of cross associations are similar to those of sequences discussed in Agrawal and Srikant's work, however due to the nature of the problem we deal with (i.e., mining correlated event histories), we can utilize more efficient algorithms and early pruning strategies. Mannila et al. in [MTV95] developed an algorithm based on the Apriori technique to find out which episodes in a given class occur frequently in a sequence. Their algorithm assumes a single event sequence while we assume two independent event histories in our algorithms. Mining of co-evolving time sequences is explained in [YSJ+00]. In that work, the authors tackle the problem of online mining of time sequences and proposed methods for predicting the missing values in real-time. They deal with quantitative data while we are interested in the events obtained by converting the sensor values. Analysis of event histories for organizing events is described in [SU01] by Saygin and Ulusoy where events are partitioned using a single event history.

Mining of large volumes of data for useful information is currently a hot research topic. Data mining issues can be categorized as:

- *classifications*, where we try to partition the data into disjoint groups [AGI+92],
- *associations*, where some correlations among data items are sought [SA95], and
- *sequences* where we try to find sequences among data items [AS95].

Discovery of event patterns from event histories (sequences) is very similar to finding association rules among a set of items. Informally, we have a set of items and a database consisting of transactions where each transaction contains some items bought by a customer at a time in a market basket database. The problem is to find rules like "if somebody buys diapers then he/she buys a bottle of beer as well", using the database of customer transactions.

Discovery of event patterns from event sequences is discussed by Bettini et al. in [BWJL98] and Mannila et al. in [MTV95]. Mannila et al. proposed efficient algorithms for finding event patterns (they call it *frequent episodes*) by analyzing event sequences. The application area used in their work is telecommunication alarm management. They use a sliding window approach, where the window size is specified by the user. Bettini et al. provided algorithms based on Mannila et al.'s sliding window approach for discovering event patterns. They tackled a more complex problem of finding event patterns where events have multiple granularities.

Bettini et al. used a sliding window approach for the discovery of event patterns from event histories [BWJL98]. In this approach, each window can be viewed as a customer transaction where transaction identifiers are just the window numbers in the event history. The set of events recorded in the history can be mapped to a set of items. The problem of finding frequent episodes is the same as finding large item sets with a given support (i.e., frequency) in the problem of finding associations. The problem of finding large item sets is a subproblem of finding associations where an item set is large if its support (i.e., the number of transactions that contain that item set) is greater than or equal to the minimum support value provided by the user. However, the problem of finding frequent

episodes becomes more complex when the user is allowed to specify complex episode structures.

2.2 Event Histories and Relationships Among Events

Event histories consist of the sequence of past signaled events. Each event has a different label. An event occurrence is stored with its timestamp.

Alarms in a telecommunication management system [MTV95], price changes in a stock exchange market [BWJL98], service requests in event-driven web applications [LPT99], weather conditions and electricity demand changes in a power management system are among many possible event types. Although the events in any system can have different characteristics, the relationships among the events can still be captured and exploited in that system. There can be three types of relationships between two events, which are *association, cross association,* and *sequence.* An *association* between two events implies that the occurrences of the events are somehow related. For example a price increase of a stock may be associated with a price increase of another stock in a stock exchange market.

The relationship among two events is a *cross association* when the associated events belong to different event histories. Cross associations exist when there are two or more histories, and events in one of those histories are related to the events in another history. Such types of histories can be called *correlated event histories.* As an example, it can be claimed that weather and power demand histories are correlated in the context of a power management system, because the weather events such as temperature changes can affect the power demand of households or companies.

Sequence relationship between two events describes an association among events together with the relative sequence of the event occurrence. A good example for sequence is the sequential access patterns of clients to data items in a

server. We are particularly interested in cross associations and sequence relationships among the events in event histories.

Suppose that we have a set of events $\xi = \{e_1, e_2, ..., e_n\}$. An event occurrence is denoted by a pair (e, t) where $e \in \xi$ and t is the timestamp denoting the time of the occurrence of e. The occurrences of events in a system are recorded in a basic event history (basic history, for short) which is defined below.

Definition 2.2.1 *Let $\xi = \{e_1, e_2, ..., e_n\}$ be a set of events. A basic history H is the set of past events recorded with their timestamps:*
$H = \{(e_{i_1}, t_{i_1}), (e_{i_2}, t_{i_2}), ..., (e_{i_n}, t_{i_n})\}$, *where t_{i_j}, $j = 1, \ldots, n$, is a timestamp, $e_{i_j} \in \xi$, $j = 1, \ldots, n$, and n is the size of the basic history.*

Example 2.2.1 *Consider the temperature and humidity measurements in Table 2.1 from August 2 to August 6 in year 1997 taken from a weather station near Chicago. These measurements can be converted to events by categorizing the values. For the sake of simplicity, we have chosen a simple categorization with two categories, one for low, and one for high values. By looking at the maximum and minimum values, we first find the midpoint. Values below the midpoint belong to the first category, and others belong to the second category. For example, the maximum and minimum temperatures are* 80.4 *and* 64.9*, respectively. The mid point of these values is* $(80.4 + 64.9)/2$*, which is equal to* 72.65*. Temperature values above* 72.65 *are considered to belong to category "high temperature" and the others are assumed to be of category "low temperature". Table 2.2 shows the weather event history, denoted by H_w, with events from $\xi_w = \{HT, LT, HH, LH\}$ which are derived using the weather data shown in Table 2.1. Event labels used are explained in Table 2.3.*

At each time tick, the events signaled at that tick form a set called *eventset*, similar to the notion of itemsets in market basket data. The eventset corresponding to the time tick represented by timestamp "1" in Example 2.2.1 is {HT,LH}. Note that the collection of eventsets signaled over time can be viewed as market basket data. Therefore, mining associated events in a history corresponds to mining frequent (large) itemsets in market basket data.

Date	Temperature	Humidity
08/02/1997	80.4	55.9
08/03/1997	76.8	67.2
08/04/1997	70.0	77.4
08/05/1997	64.9	59.2
08/06/1997	66.8	58.7

Table 2.1: Daily temperature (in Fahrenheit), and relative humidity measures from Chicago area

Time Stamp Stamp	Events
1	{HT, LH}
2	{HT, HH}
3	{LT, HH}
4	{LT, LH}
5	{LT, LH}

Table 2.2: A weather event history for the weather data

Event Label	Meaning
HT	**H**igh **T**emperature
LT	**L**ow **T**emperature
HH	**H**igh **H**umidity
LH	**L**ow **H**umidity

Table 2.3: Meaning of weather event labels used

2.3 Fuzzy Sets and Fuzzy Rules

The theory of fuzzy sets was introduced by Zadeh [Zad65]. For a crisp set (an ordinary set that we are familiar with) S, which is a subset of the universal set U, for any element $e \in U$, either $e \in S$ or $e \notin S$ whereas for a fuzzy set there is a degree of membership in the range $[0,1]$ for each element belonging to the universal set. Crisp set theory is a special case of the fuzzy set theory where the membership degrees of any element belonging to the set is either 0 or 1. A fuzzy set is characterized by its membership function. This membership function gives us the degree of membership of each element in the universal set to the fuzzy set. Membership function of a fuzzy set F on the universal set U is generally denoted by μ_F and maps each element $x \in U$ to a real number in the range $[0,1]$, i.e.,

$$\mu_F(x) : U \rightarrow [0,1].$$

The fuzzy set theory is best understood with real life examples. Assume that we have a universal set U for all the ages a human being can have. We can define a fuzzy set *young* denoted by Y on U, and assign a membership function μ_Y to Y. A sample membership function can be defined as:

$$\mu_Y(x) = \left\{ \begin{array}{ll} 0, & x < 10, \\ \frac{x}{10} - 1, & 10 \leq x < 20, \\ 1, & 20 \leq x < 30, \\ \frac{-x}{10} + 4, & 40 \leq x. \end{array} \right\} \tag{2.1}$$

Membership function μ_Y is shown graphically in Figure 2.1. According to that, a person with age 15 is young with a membership degree of 0.5. Calculation of the membership functions of the union, intersection, and difference of two fuzzy sets is explained in [KF88].

Fuzzy logic can be viewed as an application area of fuzzy set theory [KCY97]. We may define the degree of truth of the fuzzy proposition "x is a member of

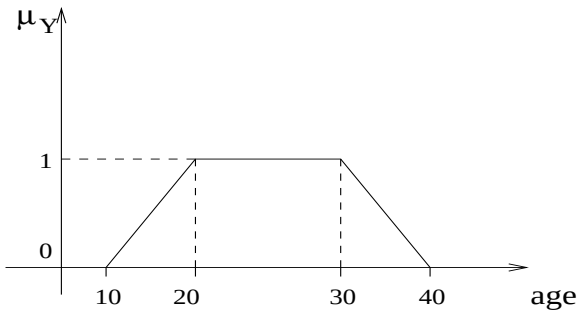

Figure 2.1: Membership function of the fuzzy set *young*

A" as the membership degree of x in A. This can be generalized to arbitrary propositions, like P :"x is F" where $x \in A$ and F is a linguistic expression such as, low, high, old, young. The degree of truth of P can be interpreted as the membership degree $\mu_A(x)$ where A is characterized by the linguistic expression F [KCY97]. So, using fuzzy logic, we can reason about the degree of truth of imprecise propositions. Fuzzy logic allows the use of [KF88]:

- fuzzy predicates like *old, expensive, high*,
- fuzzy quantifiers like *many, few, usually*,
- fuzzy truth values like *very true, mostly false*, and
- fuzzy modifiers like *almost, likely, extremely.*

Some examples of imprecise propositions are "John is tall is true", and "Mary is short is fairly false".

Fuzzy inference rules are the basic building blocks of a fuzzy controller (Mamdani type of control which is the most popular fuzzy control approach). In this approach, fuzzy control is performed in 4 steps [KCY97]:

1. Fuzzification,
2. Fuzzy inferencing,
3. Calculation of the overall conclusion, and

4. Defuzzification.

At predefined times, the measured values of input variables are received by the controller and in the first step, the matching rules are determined. In the second step, an inference is performed by each rule that is selected. In the third step, the overall conclusion is calculated, and finally in the last step, the overall conclusion is defuzzified, i.e., converted to a real value. For more details, see [KCY97].

2.4 Active Databases with Fuzzy Extensions

In this section, we introduce the basic aspects of active database systems and the previous work on fuzzy rules in active databases systems.

2.4.1 Aspects of an Active Database System

Active database systems can be considered as an extension of conventional database management systems, in such a way that the database system can respond to the state changes in database by automatically executing some actions. The production rule concept in Artificial Intelligence was modified for the active database context so that rules can respond to state changes caused by database operations [HW92]. Expert systems and active database systems are very much related in that they are both based on the concept of rules although their rule structures are different. In a typical ADBMS, system responses are declaratively expressed using Event-Condition-Action (ECA) rules [Day88]. An ECA rule is composed of an *event* that triggers the rule, a *condition* describing a given situation, and an *action* to be performed if the condition is satisfied. Primitive events can be combined to form composite events. Composition of primitive events can be done with various event constructors, like conjunction, disjunction, or closure. Coupling modes between event and condition, and between condition and action determine when the condition should be evaluated relative to the occurrence of the event, and when the action should be executed relative to the satisfaction of

the condition, respectively. Rules can be executed sequentially or concurrently depending on the underlying application [SUC98]. An abstract ECA rule for electricity producing and selling company is given below:

Event : Power consumption of company A is increased by 20%
Condition: If the temperature and humidity has increased 30%
Action: Increase the production and the price of electricity by 10%

2.4.2 Fuzzy Rules In Active Database Systems

Fuzzy concepts were integrated into expert systems and database systems [BP82, GSPB96, BP83, YBP99, YSBP99], and it was previously shown that incorporation of fuzzy concepts into databases is desirable and does solve the problems of uncertainty and inherent fuzziness of acquired data [YG99, Pet96]. The use of fuzziness in active database context, which extends the standard databases by rules, was also shown to be useful [WB98, BW97, SUY99]. Although incorporation of fuzziness to active databases introduces much flexibility, not much attention has been paid so far to this issue. To the best of our knowledge, only a research group in VTT (Finland) worked on fuzzy triggers [BW96, BW97, BKPW97, WB98]. In [BKPW97], a *Condition-Action (CA)* fuzzy trigger was proposed which means that fuzziness was introduced to the CA part of an *Event-Condition-Action (ECA)* rule. In a later work [BW97], the concept of CA trigger was extended to a fuzzy ECA rule by introducing the notion of fuzzy events. A CA fuzzy trigger consists of a fuzzy predicate (i.e., a predicate that has linguistic terms) on the database as its condition, and a fuzzy action which is an overall conclusion obtained after evaluating fuzzy conditions. Wolski and Bouaziz compiled their previous work on fuzzy ECA rules and based their contributions on a sound theoretical background in [WB98]. A rule with a fuzzy condition and a crisp action is called a *C-fuzzy* trigger. The C-fuzzy trigger model is based on linguistic terms. The *max-min* inference method is applied to the rule set to determine the truth value of the fuzzy predicates. In fuzzy ECA rules, an

event may fire a set of rules. Fuzzy events are defined as fuzzy sets and use linguistic terms like *high, low,* and *strong* [BW97]. Formally a primitive fuzzy event is represented as a tuple $< e_c, e_f >$ where e_c is a crisp event, and e_f is a fuzzy event predicate. When a crisp event is signaled (such as a database update), the current value v produced upon the operation causing the crisp event is fed into the membership function of e_f. The output of the membership function is called the *event match factor*, and the fuzzy event is signaled only if the event match factor is greater than zero [BW97]. Upon the occurrence of the fuzzy event, the corresponding rules are fired and their conditions (which are fuzzy predicates on the database) are checked. The action of a rule may be started to execute depending on the result of condition evaluation.

We may define a fuzzy ECA rule as in the following example:

Event : Power consumption of company A is *high*
Condition: If the temperature and humidity are *high* and A is a major customer
Action: Increase the production and the price of electricity

where the linguistic terms like *high*, *low*, and *major* increase the understandability of the rule. It is very difficult to give exact numbers for the temperature. Instead, predefined linguistic terms, which are in fact fuzzy sets, can be used without difficulty by the people who define the rules.

In the above rule, the event "*high* temperature" is actually a fuzzy event. When a temperature event is signaled, which is a crisp event, its current value, say 40^oC, is fed into the membership function of the corresponding fuzzy event (i.e., "*high* temperature"). Assuming that the membership function of the fuzzy event is μ_f, then the value $\mu_f(40)$ is called the event match factor.

Another possible application area is the stock exchange market. Price changes of certain stocks can mimic the price changes of some other related stocks, and dealers can take actions accordingly. A rule for a stock exchange market control system can be defined as:

Event: On a *considerable* price reduction in stock A
Condition: $TRUE$
Action: Sell a *considerable* amount of stock A and stock B

In the above rule, the event, *considerable* price reduction in stock A, is a fuzzy event. When a price reduction event is signaled, its value, say 12% is fed into the membership function of the corresponding fuzzy event and the result is used in fuzzy rule execution. Such a rule is very useful since it automatically issues an action upon the occurrence of an event, therefore reducing the time required to take an action.

In this thesis, we extended the formal description of the fuzzy events provided by Wolski and Bouaziz [WB98], to composite fuzzy events. We also studied fuzzy coupling modes and introduced the concepts of similarity based event detection and fuzzy rule execution via scenarios.

2.5 Data Broadcast

We have chosen data broadcast environments for a case study of predictive event detection and proactive rule execution. The continuous broadcast of data items from the server to a number of clients can be considered as simulating a rotating storage medium, or a broadcast disk as proposed in [ZFAA94]. There are two main approaches for data dissemination through broadcast [AFZ97]:

1. *Push based approach* where data is broadcast according to predefined user profiles. This approach does not consider the current client requests.

2. *Pull based approach* where data is broadcast according to user requests. This approach is the same as the client-server paradigm.

Each approach has its own benefits and drawbacks. In the push based approach, the server sends the data regardless of the current user requests, where users may end up with receiving unrequired data. In the pull based approach, on the other hand, the server load is very high since the server has to listen to client requests. To overcome these drawbacks and make use of the benefits of both approaches, a hybrid approach was developed by Acharya et al. that combines the two approaches and is called *interleaved push and pull* [AFZ97]. In this approach, there exist both a broadcast channel and a backchannel for the user requests. A hybrid data delivery model that combines push and pull[1] was also proposed by Sthathatos et al. [SRB97].

In our work, we assume a hybrid broadcast scheme where the user requests sent by the back channel are logged in the broadcast history. User requests are processed as in the hybrid scheme of [AFZ97], but the organization of the data to be broadcast is determined by sequential patterns obtained by mining the broadcast history.

In broadcast disks, the set of items that are broadcast is called the *broadcast set*. The sending sequence of the items in the broadcast set is called the *broadcast schedule*. Construction of the broadcast schedule is crucial for the performance of the broadcast disk. The broadcast schedule affects the waiting times of mobile clients for the item of their interest to arrive. This problem is similar to the problem of scheduling disk requests, since we are dealing with a "disk on air" in a sense [IVB97]. The difference is in that the disk on air is uni-directional and single-dimensional, since we cannot go back and forth and do random access on it. Therefore, it is more appropriate to call this new type of storage medium "one-way tape on air".

The problem of scheduling the broadcast requests is to determine the sequence of items in the broadcast schedule. We need to determine what should be put in the schedule and in what sequence, by taking into account the previous request event patterns of the clients (i.e., the broadcast history). Organizing the data on air is similar in a sense to organizing the events in an event based system

[1] The authors call them broadcast and unicast, respectively.

based on the proximity of event signaling. In an event based system, it is very logical to organize the events in a way that events signaled very close in time are placed in the same set. Similarly, the broadcast items that are requested together frequently should be broadcast close to each other in time.

When we are dealing with broadcasting in mobile environments, caching and prefetching of broadcast items also turn out to be very important from the performance viewpoint of the mobile system. The impact of caching in terms of communication cost in mobile environments was studied by Sistla and Wolfson [SW98]. It was shown that caching is an important factor for minimizing the communication cost which is of great importance in mobile computing environments due to bandwidth limitations. Caching in mobile computing environments has different characteristics than caching in a traditional client-server environment. This is due to the fact that, in mobile computing environments data items that are not cached are not equidistant to the client since the broadcast disk is single dimensional. Some caching strategies were proposed by Acharya et al. considering this aspect of broadcast disks [AAFZ95]. These strategies take into account the access probabilities of the cached items together with the frequency of broadcast. A prefetching technique for broadcast disks was proposed by Acharya et al. [AFZ96] which loads the data items to cache before an actual request is made to these data items. This technique uses a heuristic that calculates a value for each data page by multiplying the probability of access for that page by the time that will elapse before that page appears next on the broadcast disk. The decision whether a data page on broadcast is going to be replaced by one of the data pages in the mobile client cache is based on the values calculated.

Prefetching in broadcast environments can be achieved by analyzing the broadcast history of client request events. The resulting client request event patterns can further be used for predictive event detection and prefetching these data items. In our work, prefetching is implemented as proactive rule execution based on predictive event detection.

Chapter 3

Analysis of a Single Event History

In this chapter, we propose methods to extract the relationships among events from a single event history. We are particularly interested in finding the similarities of the events in an event history and finding client request event patterns from a broadcast history. In Section 3.1, we present a method for extracting the similarities between event pairs in an event based system. Event similarities extracted from an event history are further shown to be useful for the construction of fuzzy event sets and fuzzy rule execution in active database systems as to be explained in Chapter 5. In Section 3.2, we describe the analysis of a broadcast history to obtain client request event patterns. Client request event patterns are used for broadcast data organization and for loading the data items to mobile client cache proactively in broadcast environments as to be explained in Chapter 6.

3.1 Extracting Event Similarities from an Event History

Assume that we have a set of events $E = \{e_1, e_2, ..., e_n\}$, and an event history H where the events are stored in the form: $< e, p, t >$ where $e \in E$, p is the event parameter (i.e., the value returned by the real event such as the power consumption), and t is the time-stamp of the event occurrence. Our problem is to find similarities among the events in E using event history H. There might be different approaches for obtaining the similarities among events. Our approach uses the proximity of event occurrences in the event history. If two events are frequently being fired together in a short period of time, then it is likely that these events belong to the same context; therefore, they are *similar* in that sense. We use a sliding window approach to analyze the event history and obtain the similarities among the events. In what follows, we explain our approaches in more detail.

3.1.1 Preliminary Definitions

Definition 3.1.1 *A window, win, of size m is a collection of m consecutive time stamps and the events belonging to these time stamps. win[i] denotes the events in the i^{th} timestamp of win, where $1 \leq i \leq m$.*

Definition 3.1.2 *Pivot events are the events that belong to the first time stamp of the current window, i.e., pivot events = win[1].*

Example 3.1.1 *In Figure 3.1, a sample event history is depicted where the letters a, b, and c correspond to power consumption of three companies and the subscripts l, and h correspond to low and high power consumption respectively. In Figure 3.1, w_1 is a window that covers timestamps, t_1 and t_2. Pivot events of w_1 are a_h, b_h, and c_l.*

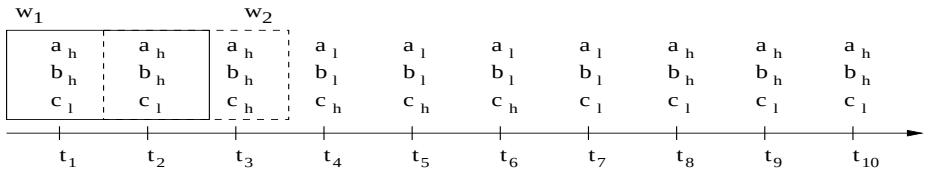

Figure 3.1: Sliding windows, illustrated on the history of power consumption events

Definition 3.1.3 *Inverse distance between two event occurrences,* (e_i, t_l) *and* (e_j, t_k) *mimics the distance of occurrence of* e_i *and* e_j*. Inverse distance is a value in the range* $(0, 1]$ *and we have chosen the formula* $\frac{1}{|t_k - t_l|+1}$ *to calculate the inverse distance.*

Example 3.1.2 *In Figure 3.1, the inverse distance between* (a_h, t_1) *and* (b_h, t_2) *is* $\frac{1}{|2-1|+1} = 0.5$*. The inverse distance between two concurrent event occurrences* (a_h, t_1) *and* (c_l, t_1) *is* $\frac{1}{|1-1|+1} = 1$*.*

The inverse distance of concurrent event occurrences is 1, which is the maximum possible value and the inverse distance value between (e_i, t_l) and (e_j, t_k) goes to 0 as $t_k - t_l$ goes to infinity.

Definition 3.1.4 *Event relationship matrix is an* $N \times N$ *matrix where* N *is the number of possible events that may occur in the system. Each element* $m_{i,j}$ *of the matrix is the cumulative inverse distance of the occurrence of event* e_j *after the occurrence of event* e_i *observed in the history.*

We also take into account the order of event occurrences; i.e., the proximity of event occurrences in the order e_i, e_j (i.e., $e_i \rightarrow e_j$) may be different from that of the same events in the order e_j, e_i (i.e., $e_j \rightarrow e_i$). In that sense our notion of distance has a direction.

```
Begin
    initialize matrix M to 0
    initialize window to first m events
    repeat
    {
        Initialize pivot_events to the events
            in the first time unit of window
        foreach pair of events e_i, e_j in pivot_events
        {
            increment M[e_i, e_j] by one
            increment M[e_j, e_i] by one
        }
        foreach event e_p in pivot_events
        {
            foreach event e_r in rest of window
            {
                increment M[e_p, e_r]
                    by inverse distance of e_r to e_p
            }
        }
    }
    until the end of H is reached
End
```

Figure 3.2: Sliding Window Algorithm

3.1.2 Sliding Window Algorithm

In order to construct the event relationship matrix, we use a sliding window approach similar to the one described by Mannila et al., in [MTV95]. The difference of our approach from that of Mannila et al. is that, we do not find event patterns, but try to find pairwise correlations among events in terms of proximity of event occurrences. We store the results of pairwise correlations in a matrix and use this intermediate structure to construct fuzzy event sets.

Our sliding window algorithm for mining event histories is presented in Figure 3.2. The algorithm traces the whole event history in sequence and considers

	a_h	a_l	b_h	b_l	c_h	c_l
a_h	0.0	0.0	1.5	0.0	0.0	1.5
a_l	0.0	0.0	0.0	0.0	0.0	0.0
b_h	1.5	0.0	0.0	0.0	0.0	1.5
b_l	0.0	0.0	0.0	0.0	0.0	0.0
c_h	0.0	0.0	0.0	0.0	0.0	0.0
c_l	1.5	0.0	1.5	0.0	0.0	0.0

Figure 3.3: Sample Event Relationship Matrix after processing the first window

	a_h	a_l	b_h	b_l	c_h	c_l
a_h	0.0	0.0	3.0	0.0	0.5	2.5
a_l	0.0	0.0	0.0	0.0	0.0	0.0
b_h	3.0	0.0	0.0	0.0	0.5	2.5
b_l	0.0	0.0	0.0	0.0	0.0	0.0
c_h	0.0	0.0	0.0	0.0	0.0	0.0
c_l	3.0	0.0	3.0	0.0	0.5	0.0

Figure 3.4: Sample Event Relationship Matrix after processing the second window

the events in a window of size m. In Figure 3.1, we have set the window size to 2. The first window, w_1, is shown with a rectangle drawn using straight lines. It covers the events at the first two time stamps, t_1 and t_2. The window slides one time unit at each iteration. The second window, w_2, is shown in Figure 3.1 with a rectangle drawn with dashed lines. The second window covers the events with time stamps t_2 and t_3. At each iteration, proximity of the pivot events to the rest of the elements in the window is updated one by one in the matrix. After this process, window slides one event to the right and pivot events are updated again. This process continues until the end of the history is reached, i.e., time stamp of the pivot events is the last time stamp in the event history. The output of the algorithm is the event relationship matrix obtained by tracing all the event history.

The event relationship matrix keeps the proximity of events incrementally. Its elements are initially set to zero[1]. We would like to give an example to facilitate

[1]In case of incremental runs of the algorithm, the contents of the previous matrix are used

	a_h	a_l	b_h	b_l	c_h	c_l
a_h	0.0	0.5	8.0	0.5	2.0	6.5
a_l	0.5	0.0	0.5	5.5	4.0	2.0
b_h	8.0	0.5	0.0	0.5	2.0	6.5
b_l	0.5	5.5	0.5	0.0	4.0	2.0
c_h	1.0	5.0	1.0	5.0	0.0	0.5
c_l	7.5	1.0	7.5	1.0	0.5	0.0

Figure 3.5: Sample Event Relationship Matrix after processing the whole event history

understanding of the sliding window algorithm. Assume that we have an event set $E = \{a_h, a_l, b_h, b_l, c_h, c_l\}$. The event relationship matrix, say M, would be a 6×6 square matrix. A sample event history for the given event set is provided in Figure 3.1 with the first and second event windows shown. Figure 3.1 shows the first event window w_1, of size 2 in the history. Initially, all the entries in the event relationship matrix are 0. During the first iteration, the pivot events are a_h, b_h, and c_l. As the first step, proximity values of the pivot events are updated in the matrix. As a second step, proximity values of the pivot events to the rest of the events in the current window are incremented in matrix M. After the first pass, the matrix is updated as shown in Figure 3.3.

In the second pass, the window is moved one position right, as shown in Figure 3.1, and the new pivot events are set to a_h, b_h and c_l. The same process is repeated for the new pivot events. After the second pass, entries of the matrix are as shown in Figure 3.4. The final proximity values of the events are given in Figure 3.5.

Giving more weight (increased proximity) to the events occurring very close to the pivot event in the window improves the accuracy of the sliding window algorithm.

A nice property of the sliding window algorithm is that, it is incremental. Assume that the time-stamp of the last event processed by the previous run of

as initial values.

the algorithm is t_1. As the event history grows with time, the algorithm can be reapplied to the event history starting from t_1 and the matrix values are updated according to the new event patterns. For some applications, like stock exchange, very old portions of the event history may be obsolete and misleading. Therefore, after a reasonable time it may be necessary to discard the old proximity values and produce them from scratch starting from some point in the history.

The event relationship matrix that is constructed by mining the event history is used for fuzzy rule execution in active database systems. Fuzzy rule execution is performed by similarity based event detection and construction of rule scenarios, which are explained in Section 5.2.

3.1.3 Time Complexity of the Algorithm

Time complexity of the sliding window algorithm is determined by the size of the event history, size of the window and the maximum possible number of concurrent events.

Lemma 3.1.1 *Time complexity of the sliding window algorithm is $O(n \times c^2 \times m)$, where n is the event history size, m is the window size and c is the maximum number of concurrently occurring events in the event history.*

Proof

This result is due to the fact that the whole event history is traced for once and at each slide of the window m comparisons are made. The window is initialized to first m timestamps, and then the window slides $n-1$ times. Therefore, the total number of iterations at the repeat-until loop is n. At each iteration of the loop, first the pivot events are initialized and for each pair of events in the pivot events, the matrix entries are updated. Updating an entry in the matrix takes constant time, and the number of pairs in the pivot events is $\binom{c}{2}$ where c is the maximum number of concurrent events at a time. After that, for each event in the pivot events, we update the proximity of that event with the rest of

the events in the window. Since there are $m-1$ time points in the rest of the window (excluding the time point of pivot events), this operation is done at most $c \times c \times (m-1)$ times. Therefore, the maximum possible number of operations is, $n \times (\binom{c}{2} + c^2 \times (m-1))$ which equals to $n \times (\frac{c \times (c-1)}{2} + c^2(m-1))$. Since we have calculated the maximum possible number of operations with respect to the maximum number of concurrent events, the time complexity of the sliding window algorithm is $O(n \times c^2 \times m)$ ■

The only parameter that we can control in the time complexity formula is the window size. As the window size increases the precision of the result of the mining increases, however the time complexity also increases. There is a tradeoff between the time complexity and precision. The selection of window size should consider this tradeoff.

3.2 Broadcast History Mining

In this section, we demonstrate how the useful information in the broadcast history can be extracted in the form of sequential patterns. We discuss the issues related to the extraction of sequential patterns from broadcast histories and management of the resulting patterns.

3.2.1 Sessions and Sequential Patterns

Client requests are the events of interest to us in a broadcast history. In order to mine for sequential event patterns, we assume that the continuous client requests are organized into discrete sessions. Sessions specify user interest periods, and a session consists of a sequence of client requests for data items ordered with respect to the time of reference. The whole database is considered as a set of sessions. The client requests to data items are considered as events. Ordering among the sessions is not important while the ordering of the data items inside a session is important for extracting the sequential patterns. Formally, we have a

set of items, $I = \{i_1, i_2, ..., i_m\}$ and a set of sessions D, such that $\forall S \in D, S \subseteq I$.

Definition 3.2.1 *A session S supports a sequence of items X if the items in X appear in S in the same order as they appear in X, i.e., X is a subsequence of S.*

Sequential patterns are obtained from the sessions and the items are ordered in each session. Sequential patterns are different from associations in that the ordering of data items is important in a sequential pattern and the order is determined by the ordering of the items in individual sessions.

A sequential pattern p of size k consists of ordered data items, $p_1, p_2, ..., p_k$, and is represented as $p =< p_1, p_2, ..., p_k >$.

Definition 3.2.2 *A sequential pattern p has support s if $s\%$ of sessions in D supports p. A session supports a sequential pattern if that pattern appears in the session.*

Sequential rules are obtained from sequential patterns.

Example 3.2.1 *For a sequential pattern*
$p =< p_1, p_2, ..., p_k >$, the possible sequential rules are:

$< p_1 > \Rightarrow < p_2, p_3, ...p_k >$,

$< p_1, p_2 > \Rightarrow < p_3, p_4, ...p_k >$,

...

$< p_1, p_2, ..., p_{k-1} > \Rightarrow < p_k >$.

Definition 3.2.3 *A sequential rule $< p_1, p_2, ..., p_n > \Rightarrow < p_{n+1}, p_{n+2}, ..., p_k >$, where $0 < n < k$, has confidence c if $c\%$ of the transactions that support $< p_1, p_2, ..., p_n >$ also supports $< p1, p2, ..., p_k >$, i.e., $confidence(p) = \frac{support(<p_1, p_2, ..., p_n>)}{support(<p1, p2, ..., p_k>)}$.*

For a sequential pattern $p =< p_1, p_2, ..., p_k >$, among the possible rules that can be derived from p, we are interested in the rules with the smallest possible

antecedent (i.e., the first part of the rule). This is due to the fact that the rules are used for inferencing, and inferencing should start as early as possible. The rest of the rules trivially meet the confidence requirement. This follows from the formulation of $confidence(p)$ as specified above. Support of a sequential pattern is less than the support of any of its subpattern. Therefore, as the antecedent of the rule grows, the support of the rule shrinks, making the confidence higher.

3.2.2 Finding Sequential Patterns in Data Broadcast Histories

In this subsection, we discuss how broadcast histories can be analyzed to come up with sequential rules describing the sequential order and frequency of occurrence of requests for data items in the data broadcast history.

The data mining problem in the context of broadcast disks is to extract useful information hidden in the broadcast history in the form of sequential rules. Mining of broadcast histories can be performed individually by each data server in a distributed fashion in case the broadcast history is distributed among the servers. Distributed rule mining may result in global sequential rules as discussed in [CNFF96], or each server can maintain its own localized sequential rule set independent of the other sites. The choice of whether we should use a globalized or a localized approach is system dependent. A globalized approach is more beneficial in systems where the profiles of the data items kept in each server are basically the same. However, in case the data items stored in different servers are completely unrelated, a localized approach is more useful. Security issues also come into picture when we deal with the analysis of private user access histories. Autonomous systems may sometimes not be willing to distribute user requests to the others. This situation can be handled by assigning symbolic ids to users and distribute the local broadcast results. However, some sites may not be willing to share valuable information with the other sites. In such a case, using a localized approach of obtaining sequential rules is the only solution.

There can be two basic approaches for mining the broadcast history depending

$user_1$	(win_{1_1}: x z y; win_{1_2}: z y; ...; win_{1_k}: p q u)
$user_2$	(win_{2_1}: x p; win_{2_2}: u w v; ...; win_{2_l}: r s)
...	...
$user_n$	(win_{n_1}: x y; win_{n_2}: u v; ...; win_{n_m}: x y z)

Table 3.1: User-based partitioning of the broadcast history.

on how the history of user requests is partitioned:

1. *Flat* approach.
2. *User-based partitioning* approach.

The flat approach tries to extract data item request patterns regardless of who requested them. The user-based partitioning approach, on the other hand, divides the broadcast history into subsets with respect to the user who requested them as shown in Table 3.1. Table 3.1 shows the corresponding windows of user requests for each user, such as $user_1$ who had k different request windows. The analysis is done for each subset of the broadcast history corresponding to a user independent of the other subsets.

In both approaches, we divide the request set into windows and find the sequential rules using the data mining algorithms we have discussed in the preceding subsection. However, the construction of the windows is different for the two approaches. In the user-based partitioning approach, windows are clusters of items requested according to the times of requests; i.e., if a user has requested some items consecutively with short periods of time in between, then these items should be put in the same window. We define a window as a group of requests where the time delay between two consecutive requests is less than a certain threshold. After requesting a set of items consecutively, if the user waits for a long period of time before starting another session, then the sequence of items requested in the new session can be put into another window. In Table 3.1, the requests of $user_1$, for instance, are divided into k windows. The first window of $user_1$, denoted by win_{1_1}, has three requests, namely the data items x, z, and y. We need to set a

threshold value for the time delay in between two consecutive windows. We may set this threshold to infinity to put all the items requested by the same user into the same window, which might be a reasonable approach when there are a lot of users and each user accesses a reasonable amount of (not too many) items. When the number of users is small and each user accesses a large amount of documents, then we need to set a reasonable threshold value. For the flat approach, we determine a fixed window length and use a sliding window approach to construct the windows. The sliding window approach is based on moving the window one item further and consider the items in the current window for finding the large itemsets.

The advantage of the flat approach is that there is no overhead of partitioning the items into users and clustering the items into windows. But the information about the users is lost in this approach. In user-based partitioning, the rules obtained can be considered to be more reliable, as they are based on finding sequential rules for the same user by partitioning the broadcast history into users.

In order to be able to use the data mining algorithms described in Section 2.1 to obtain sequential rules, we need to map the windows to the *sessions* and the data items to the *items* described in Section 2.1. After that we can execute a sequential pattern mining algorithm similar to the one in [AS95] to obtain the sequential patterns.

3.2.3 Elimination of Obsolete Rules and Incremental Mining

Rules that have been constructed through mining may become obsolete after a certain period of time. The rule set is dynamic in a sense. Therefore, we need to analyze the whole history periodically, eliminate the obsolete rules and add new rules if necessary. Mining the broadcast history very frequently is a waste of server resources, however, using the same set of rules for a long time may affect the system performance negatively since the current rule set may no longer reflect the access patterns of the clients. Therefore, determination of the frequency of

mining the broadcast requests is an important issue that needs to be investigated. We may use the feedback from the users to determine when the rules become obsolete. If the number of user requests is increasing, we can attribute this to obsolete rules and when the number of user requests becomes significantly larger than the initial requests, we may decide to perform the mining process again. We can determine a threshold value for the time period to wait before restarting the mining of the broadcast history and find new sequential rules. For very huge histories, mining the whole history periodically is a waste of resources; therefore, some incremental methods can be applied to reduce the time spent for remining. Efficient methods for incremental rule maintenance are proposed in [CHNW96].

3.3 Summary

In this chapter, we have discussed mining a single event history to extract event similarities and event patterns. We have shown how an event history can be analyzed to obtain the similarities among events in terms of the proximity of event occurrence. We have proposed a sliding window algorithm for mining an event history to construct a structure for storing the event similarities. We have also proposed methods for analyzing a broadcast history where the events are client requests to data items. The broadcast history is mined to obtain sequential patterns of client request events.

Chapter 4

Analysis of Correlated Event Histories

In this chapter, we define a framework for finding associations among events from different histories. Dependencies among the events belonging to the same history are not considered while relating events from different histories. We first provide the definition of some operations that will be used as the basis for the concept of cross associations in Section 4.1. In Section 4.2, we discuss some algorithms for mining cross associations from correlated event histories. In Section 4.3, we provide the implementation details and the experimental results on both synthetic and real data sets.

4.1 Preliminary Definitions

Events in one history, called *triggering event history*, can trigger a set or sequence of events in another history, called *triggered event history*. As an example, if we have two histories, one for weather events and the other for power consumption events of various companies, it is clear that weather events are triggering events and power consumption events are triggered events, as power consumption events cannot alter the weather, while weather can alter power consumption in a region.

Date	Temp	Humid	Power (Resid)	Power (Indus)
08/02/1997	80.4	55.9	1.2	26.8
08/03/1997	76.8	67.2	0.9	24.5
08/04/1997	70.0	77.4	0.9	35.0
08/05/1997	64.9	59.2	0.5	35.7
08/06/1997	66.8	58.7	0.5	35.9

Table 4.1: Temperature, humidity and the corresponding electricity demand measurements (in Kilo watts) of residential and industrial customers

Time Stamp	Weather Events	Power Events
1	{HT, LH}	{HR, LI}
2	{HT, HH}	{HR, LI}
3	{LT, HH}	{HR, HI}
4	{LT, LH}	{LR, HI}
5	{LT, LH}	{LR, HI}

Table 4.2: Two correlated histories, **W**eather and **P**ower demand

The data mining problem in case of two correlated event histories is to find inter history rules that describe the relationship between the events in the triggering and triggered histories with a certain support and confidence.

For market basket analysis, we derive the association rules from a frequent itemset by looking at all possible combinations that may result in an association rule and selecting the ones that meet the minimum confidence requirement. However in case we have two histories, H_1 and H_2, with events from ξ_1 and ξ_2 respectively, and H_1 being the triggering event history, the rules that we are interested in are of the form $A \Rightarrow B$ where $A \subset \xi_1$ and $B \subset \xi_2$. Associations spanning two histories are called *cross associations*.

Example 4.1.1 *Two correlated histories, namely weather (H_w) and power (electricity) demand (H_p) are shown in Table 4.2. The power events are obtained from the power demand data shown in Table 4.1. This data is the average hourly power*

Event Label	Meaning
HR	**H**igh **R**esidential power consumption
LR	**L**ow **R**esidential power consumption
HI	**H**igh **I**ndustrial power consumption
LI	**L**ow **I**ndustrial power consumption

Table 4.3: Meaning of power consumption event labels used

consumption (in kilowatt-hours), averaged over a day of a typical household (residential) and a small scale industrial corporation (industrial) located in Chicago area. The power data is sampled at the same time as the corresponding weather data. Events are derived by using the same categorization technique as weather data. The events of H_w are from the set $\xi_w = \{HT, LT, HH, LH\}$, and the events of H_p are from the set $\xi_p = \{HR, LR, HI, LI\}$. The meanings of the event labels are described in Table 2.3 and Table 4.3.

Definition 4.1.1 *Let ξ be a set of events, H be a basic history with events from ξ, and $e \in \xi$ be an event in H. $timeTicks_H(e)$ denotes the set of time ticks where event e occurred in history H. $|timeTicks_H(e)|$ denotes the cardinality of the set of time ticks of e in H. For a set of events E[1], $timeTicks_H(E) = \cap_{e \in E} timeTicks_H(e)$, and $timeTicks_H(\emptyset)$ is defined to be the set of all time ticks in history H.*

Two time ticks t_1 and t_2 are said to be *corresponding time ticks* in H_1 and H_2 respectively, if t_1 in H_1 semantically corresponds to t_2 in H_2. Corresponding time ticks may have the same value or may have different values. For simplicity, we will assume that corresponding time ticks in two histories are represented by the same label.

Definition 4.1.2 *Let H_1 and H_2 be two histories with two disjoint sets of events, ξ_1 and ξ_2, where $\forall(e_i, t_k) \in H_1, e_i \in \xi_1$ and $\forall(e_j, t_l) \in H_2, e_j \in \xi_2$. Let A and B be*

[1] *Note that ξ denotes the set of all different events that can happen in a history, while E denotes an event set, i.e., a subset of ξ.*

two event sets. A cross association between A and B, from H_1 to H_2 is denoted as $A_{H_1} \stackrel{C}{\Rightarrow} B_{H_2}$. $A_{H_1} \stackrel{C}{\Rightarrow} B_{H_2}$ exists if $\exists t_m, t_n$ such that $\forall a \in A, a \in \xi_1, (a, t_m) \in H_1$ and $\forall b \in B, b \in \xi_2, (b, t_n) \in H_2$, and t_m, t_n are corresponding time ticks in histories H_1 and H_2.

In case of correlated event histories, we are interested in association rules spanning two histories which are defined by the cross associations. Therefore, cross associations will imply cross association rules.

Example 4.1.2 *The cross association, $\{HT, HH\}_{H_w} \stackrel{C}{\Rightarrow} \{HR, LI\}_{H_p}$, exists over weather and power demand event histories given in Table 4.2. A logical explanation of this association is that a high humidity and temperature triggers an increase in the usage of ACs that leads to an increased consumption in residences. However, as the electricity demand increases, the price of electricity also increases leading to a decrease in the electricity consumption in the industry. Electricity companies sell cheap electricity to industry under the condition that they can cut the electricity for some non-vital units in case of peak electricity demands.*

Definition 4.1.3 *Let H be a basic history with events from ξ. Support of an event set E in history H, denoted by $support_H(E)$ is defined as, $support_H(E) = \frac{|timeTicks_H(E)|}{|timeTicks_H(\emptyset)|} * 100$.*

Example 4.1.3 *For the history given in Table 2.2,*
$timeTicks_{H_w}(LT) = \{3, 4, 5\}$, *and*
$timeTicks_{H_w}(\{LT, LH\}) = \{3, 4, 5\} \cap \{1, 4, 5\} = \{4, 5\}$.
$timeTicks_{H_w}(\emptyset) = \{1, 2, 3, 4, 5\}$.
$Support_{H_w}(\{LT, LH\}) = \frac{|timeTicks_{H_w}(\{LT,LH\})|}{|timeTicks_{H_w}(\emptyset)|} * 100 = \frac{2}{5} * 100 = 40\%$

The time ticks in $timeTicks_H(\emptyset)$ correspond to transaction identifiers in the market basket data, and the notion of support in the context of event sets is the same as the support of an itemset in market basket analysis.

Definition 4.1.4 *A cross association* $A_{H_1} \stackrel{C}{\Rightarrow} B_{H_2}$ *has support* $s\%$ *if the percentage of the corresponding time ticks in* $\{timeTicks_{H_1}(A) \cup timeTicks_{H_2}(B)\}$ *is* s. *Support of a cross association* $A_{H_1} \stackrel{C}{\Rightarrow} B_{H_2}$ *is denoted as* $support_{H_1,H_2}(A,B)$, *and* $support_{H_1,H_2}(A,B) = \frac{|timeTicks_{H_1}(A) \cap timeTicks_{H_2}(B)|}{|timeTicks_{H_1}(\emptyset) \cup timeTicks_{H_2}(\emptyset)|}$.

A frequent cross association has a support greater than the minimum support required. Confidence of a cross association $A_{H_1} \stackrel{C}{\Rightarrow} B_{H_2}$ is $\frac{support_{H_1,H_2}(A,B)}{support_{H_1}(A)}$.

Definition 4.1.5 *A frequent cross association* $A_{H_1} \stackrel{C}{\Rightarrow} B_{H_2}$ *is maximal if there is no frequent cross association* $A'_{H_1} \stackrel{C}{\Rightarrow} B'_{H_2}$ *where* $A \subset A'$ *and* $B \cup B'$.

Example 4.1.4 *In Table 4.2, let the minimum support and confidence thresholds be 40% and 70%, respectively. Then, the cross association* $\{LT, LH\}_{H_w} \stackrel{C}{\Rightarrow} \{LR, HI\}_{H_p}$ *will be a frequent cross association since its support is 40%. Confidence of the same cross association is 100% which means that it meets the minimum confidence requirement as well. It is also a maximal cross association.*

4.2 Mining Cross Associations From Correlated Event Histories

Mining for cross associations involves two different event histories and the aim is to find frequent cross associations. Given two histories H_1 and H_2, and two sets of events ξ_1 and ξ_2 where events in H_1 are from ξ_1 and events in H_2 are from ξ_2, and $\xi_1 \cap \xi_2 = \emptyset$. The problem of mining cross associations is to find maximal cross associations from H_1 to H_2. In the following sections, we provide our approach for extracting cross associations.

We first present a base algorithm that merges and then extracts the large itemsets using any standard large itemset extraction algorithm. We then present another algorithm that considers the correlated event histories separately and utilizes some early pruning and concurrent candidate generation schemes.

4.2.1 The Merge Algorithm

The assumption that the two histories have different sets of events enables us to use an algorithm based on merging the histories. Two event histories, H_1 and H_2 with events from ξ_1 and ξ_2, respectively are first merged to form a single history. The history obtained by applying this method to the event histories given in Table 4.2 is shown in Table 4.4. After the merge step, we find the large itemsets and then partition them to obtain cross associations. The basic steps of the algorithm that is based on merging histories are:

1. Merge the two event histories by combining the event sets belonging to corresponding time ticks.
2. Find the frequent event sets, L, using Apriori[2] algorithm. $L = \{L_1, L_2, ..., L_k\}$ where $L_i, i = 1, ..., k$ contains event sets of size i, such that $\neg\exists E_1, E_2 \; where \; E_1 \in L_i$ and $E_2 \in L_j$ with $i < j \; and \; E_1 \subset E_2$.
3. Partition each event set X, into pairs (X_1, X_2), where $X_1 \subset \xi_1$ and $X_2 \subset \xi_2$.
4. Discard the pairs with either $X_1 = \emptyset$ or $X_2 = \emptyset$, and store the rest in the result set.

After applying the last step, the rest of the pairs of event sets gives us the cross associations that we are looking for. This process is shown in Tables 4.5 and 4.6. Table 4.5 lists the large itemsets. The resulting cross associations after partitioning the large itemsets are provided in Table 4.6. Note that the pairs whose left or right hand sides are empty set should be discarded, since they are not cross associations.

We can utilize some pruning strategies to reduce the number of candidate cross associations. These pruning strategies lead us to concurrent cross association mining algorithms.

[2]It is also possible to use another another algorithm to extract the large itemsets.

Time Stamp	**Events (Weather and Power)**
1	{HT, LH, HR, LI }
2	{HT, HH, HR, LI}
3	{LT, HH, HR, HI}
4	{LT, LH, LR, HI}
5	{LT, LH, LR, HI}

Table 4.4: The history obtained by merging the two correlated histories

Frequent events	HT, LT, HH, LH, HR, LR, HI, LI
Frequent event sets of size 2	{HT, HR },{HT, LI }, {LT, HI }, {LT, LH }, {LT, LR }, {LH, LR }, {LH, HI }, {HR, LI}, {HI, LR}, {HH, HR }
Frequent event sets of size 3	{HT, HR, LI }, {LT, LH, LR }, {LT, LH, HI }, {LH, LR, LI }
Frequent event sets of size 4	{LT, LH, LR, HI }

Table 4.5: Frequent event sets with minimum support 40%

Frequent event sets of size 2	{HT}{HR}, {HT}{LI}, {LT}{HI}, {LT,LH}{}, {LT}{LR}, {LH}{LR}, {LH}{HI}, {}{HR, LI}, {}{HI, LR}, {HH}{HR}
Frequent event sets of size 3	{HT}{HR,LI}, {LT, LH} {LR }, {LT, LH}{HI}, {LH}{LR, LI }
Frequent event sets of size 4	{LT, LH}{LR, HI }

Table 4.6: Cross associations after partitioning the frequent event sets

Time Stamp	**Weather Events**	**Power (Residential and Industrial)**
6	{HW, LT, LH}	{HR, LI}
7	{LW, HT, HH}	{HR, HI}
8	{LW, LT, HH}	{LR, LI}
9	{HW, LT, LH}	{LR, HI}

Table 4.7: Modified Weather and Power demand histories

Frequent events	HW, LW, LT, HH, LH, HR, LR, HI, LI
Frequent event sets of size 2	{HW, LT }, {LT, LH }, {LT, LR }, {LT, LI }
Frequent event sets of size 3	{HW, LT, LH }

Table 4.8: Frequent event sets with minimum support 50% obtained after merging the histories

4.2.2 The Concurrent Algorithm with Early Pruning

The merge algorithm is simple but not efficient in terms of the number of database scans and the number of candidates considered. We propose an algorithm that can take the advantage of the fact that there are two different histories with two different sets of events. Consider the two histories given in Table 4.7. These histories include the wind speed which is another important measure due to its cooling effect. In the table, LW corresponds to low wind speed and HW corresponds to high wind speed. Provided that the minimum support is 50%, the large itemsets obtained after merging are shown in Table 4.8. As it can be seen from the table, the longest frequent event set is {HW, LT, LH}, whereas the longest cross associations are $LT \stackrel{C}{\Rightarrow} LR$ and $LT \stackrel{C}{\Rightarrow} LI$, and they are of size 2. Therefore, we spend an extra scan of the database to find associations that are not of interest.

The concurrent algorithm finds the frequent event sets and cross associations at the same time. That is where the name *concurrent* comes from, and the algorithm makes use of the following principles stated as lemmas:

Lemma 4.2.1 *Let* $A_{H_1} \stackrel{C}{\Rightarrow} B_{H_2}$ *be a frequent cross association such that* $support_{H_1,H_2}(A,B) \geq s$. *Then,* $support_{H_1}(A) \geq s$ *and* $support_{H_2}(B) \geq s$ *in histories* H_1 *and* H_2, *respectively.*

This lemma states that the support of the components of a cross association should be higher than the cross association itself. This is obviously true since the cross association is a superset of it components. This principle is going to be used for generating candidate cross associations in the concurrent algorithm. The concurrent algorithm scans for large itemsets concurrently in both histories, since only large itemsets can be the components of cross associations. The large itemsets are then used for generating candidate cross associations with a special candidate generation scheme.

Lemma 4.2.2 *For a cross association* $A_{H_1} \stackrel{C}{\Rightarrow} B_{H_2}$ *to be frequent, all cross associations* $A'_{H_1} \stackrel{C}{\Rightarrow} B'_{H_2}$ *where* $A' \subset A$ *and* $B' \subset B$ *should also be frequent cross associations.*

This principle states that for a cross association to be frequent all its sub-cross-associations should also be frequent. This is very similar to the Apriori principle, and will be used for pruning the cross associations while generating the candidate cross associations.

Lemma 4.2.3 *A cross association* $A_{H_1} \stackrel{C}{\Rightarrow} B_{H_2}$ *has minimum confidence if and only if all cross associations of the form* $A_{H_1} \stackrel{C}{\Rightarrow} B'_{H_2}$ *where* $B' \subset B$ *also have minimum confidence.*

Proof: Confidence of a cross association $A_{H_1} \stackrel{C}{\Rightarrow} B_{H_2}$ is $\frac{support_{H_1,H_2}(A,B)}{support_{H_1}(A)}$.
We know that $support_{H_1,H_2}(A,B) \leq support_{H_1,H_2}(A,B')$ where $B' \subset B$.
Therefore, $\frac{support_{H_1,H_2}(A,B)}{support_{H_1}(A)} \leq \frac{support_{H_1,H_2}(A,B')}{support_{H_1}(A)}$. ■

This principle will be used for pruning candidate cross associations by using the confidence of their sub-cross-associations. It tells us that a cross association

$A_{H_1} \stackrel{C}{\Rightarrow} B_{H_2}$ which does not meet the minimum confidence requirement, can not generate candidate cross associations such that the left component of the new cross association is the same and the right component is a superset of B_{H_2}. This is because the new cross association will not meet the confidence requirement. However, $A_{H_1} \stackrel{C}{\Rightarrow} B_{H_2}$ can generate cross associations of the form $A'_{H_1} \stackrel{C}{\Rightarrow} B'_{H_2}$ where $A_{H_1} \subset A'_{H_1}$ and $B_{H_2} \subseteq B'_{H_2}$.

The pruning strategies provided in Lemma 4.2.1, Lemma 4.2.2, and Lemma 4.2.3 are for pruning cross associations. We can also use cross associations to prune the frequent event sets in histories H_1 and H_2. We eliminate the large itemsets X in H_1 and H_2 at the k^{th} step of the algorithm, provided that there is no frequent cross association $A_{H_1} \stackrel{C}{\Rightarrow} B_{H_2}$ where $A_{H_1} \subset X$ or $B_{H_2} \subset X$. With this pruning strategy, we ensure that no redundant candidate frequent event sets are generated.

Benefits of the concurrent cross event set mining algorithm can be summarized as follows:

- Parallelism; i.e., while we are scanning the database, we can generate event sets and cross event sets concurrently.
- Pruning; i.e., we can utilize all pruning strategies stated in Lemma 4.2.1, Lemma 4.2.2, and Lemma 4.2.3. We can also prune the frequent event sets using cross associations obtained in the intermediate steps.
- The number of database scans is lower than the number required for the merge algorithm.

Concurrent mining of cross associations is performed by generating the cross associations while finding the event sets in both histories. At level k, while candidate event sets of size k are being counted, candidate cross associations $A_{H_1} \stackrel{C}{\Rightarrow} B_{H_2}$ are counted where $|A_{H_1}|, |B_{H_2}| < k$.

Our algorithm is based on the *Apriori Algorithm*. Apriori algorithm finds the candidate large itemsets C_{k+1} using only the large itemsets in L_k that are found

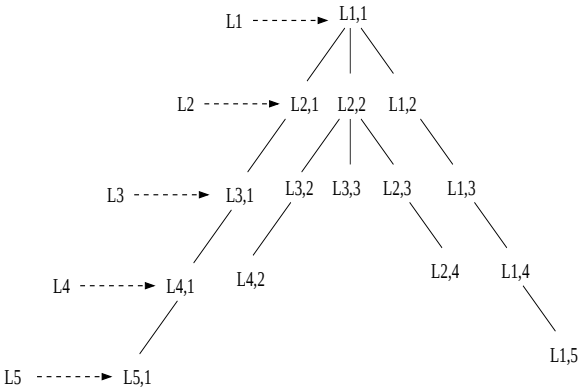

Figure 4.1: Frequent cross association tree

at the k^{th} scan of the database. Basic steps of our algorithm for finding frequent cross event sets can be described as follows:

Candidate generation should consider all of the pruning strategies. A frequent cross association $A_{H_1} \stackrel{C}{\Rightarrow} B_{H_2}$ will be extended in three different ways to generate larger candidate frequent event sets:

1. $(A \cup \{a_i\})_{H_1} \stackrel{C}{\Rightarrow} B_{H_2}$
2. $A_{H_1} \stackrel{C}{\Rightarrow} (B \cup \{b_j\})_{H_2}$
3. $(A \cup \{a_k\})_{H_1} \stackrel{C}{\Rightarrow} (B \cup \{b_l\})_{H_2}$

and all of these event sets should be considered at the same time as candidates. However, the candidate generation process should avoid redundancies. In order to avoid the redundancies, we do a breadth first search on the candidate tree to find the frequent event sets. A sample frequent event set tree is shown in Figure 4.1. At the root of the tree (the first level), L_1 contains only $L_{1,1}$, i.e., pairs of frequent events. In general, at level k, L_k contains $L_{i,j}$ which consists of frequent event sets (A, B), such that $|A| = i$ and $|B| = j$, and at least one of i, j is equal to k. The concurrent algorithm is shown in Figure 4.2. At each iteration of the algorithm, the candidates from $L_{i,j}$ are generated as follows:

if $i = j$, then generate $C_{i,j}, C_{i+1,j}, C_{i,j+1}$
else if $i < j$, then generate $C_{i,j+1}$

Begin
$L_1^{H_1} = \{Frequent\ events\ in\ H_1\}$
$L_1^{H_2} = \{Frequent\ events\ in\ H_2\}$
$k = 1$
while $((L_k^{H_1} \neq \emptyset)$ and $(L_k^{H_2} \neq \emptyset))$ **do**
$C_{k+1}^{H_1} = aprioriGen(L_k^{H_1})$
$hashTreeInsert(HT, C_{k+1}^{H_1})$
$C_{k+1}^{H_2} = aprioriGen(L_k^{H_2})$
if $k = 1$ **then** // this is a special case for $L = \emptyset$
$C_k = \{(x, y) | x \in L_1^{H_1} \wedge y \in L_1^{H_2}\}$
else
$C_k = \emptyset$
for each $L_{ij} \in L_k$ **do**
if $i = j = (k-1)$ **then**
$C_{k,k-1} = multiGenLeft(L_{k-1,k-1}, L_k^{H_1})$
$C_{k-1,k} = multiGenRight(L_{k-1,k-1}, L_k^{H_2})$
$C_{k,k} = multiGenLeftRight(L_{k-1,k-1}, L_k^{H_1}, L_k^{H_2})$
$C_k = C_k \cup \{C_{k,k-1}\} \cup \{C_{k-1,k}\} \cup \{C_{k,k}\}$
else if $j = k$ **then**
$C_{i,k} = multiGenRight(L_{i,k-1}, L_k^{H_2})$
$C_k = C_k \cup \{C_{i,k}\}$
else // the case when $i = k$
$C_{k,j} = multiGenLeft(L_{k-1,j}, L_k^{H_1})$
$C_k = C_k \cup \{C_{k,j}\}$
endfor
$hashTreeInsert(HT, C_k)$
$hashTreeInsert(HT, C_{k+1}^{H_2})$
// count the support of candidate cross associations and frequent event sets
// for each transaction in the order of timestamps
for $i = 1$ **to** N **do**
$concurrentHashCheck(HT, t_i^{H_1}, t_i^{H_2})$
endfor
// Perform confidence pruning in case $i < j$
$CL = \{c \in C_{i,j} | C_{i,j} \in C_k \wedge i < j \wedge c.count \geq minsup \wedge c.confidence \geq minconf\}$
//Check only the support in case $i \geq j$
$SL = \{c \in C_{i,j} | C_{i,j} \in C_k \wedge i \geq j \wedge c.count \geq minsup\}$
$L_k = CL \cup SL$
$L_{k+1}^{H_1} = \{c \in C_{k+1}^{H_1} | c.count \geq minsup \wedge (\exists_{x,i}\ s.t\ x \in L_{k,i} \wedge leftComponent(x) \subset c)\}$
$L_{k+1}^{H_2} = \{c \in C_{k+1}^{H_2} | c.count \geq minsup \wedge (\exists_{x,i}\ s.t\ x \in L_{i,k} \wedge rightComponent(x) \subset c)\}$
k++
endwhile
End

Figure 4.2: Concurrent mining algorithm

Begin
$multiGenLeft(L_{i,j}, L_{i+1}^{H_1})$
select $p.item_1, ..., p.item_i, q.item_i, p.item_{i+1}, ..., p.item_{i+j}$
from $L_{i,j}\ p, L_{i,j}\ q, L_{i+1}^{H_1}\ r$
where $p.item_1 = q.item_1, ..., p.item_{i-1} = q.item_{i-1}$ and $p.item_i \leq q.item_i$
and $p.item_1 = r.item_1, ..., p.item_i = r.item_i, q.item_i = r.item_{i+1}$
End

Figure 4.3: Left candidate generation algorithm

else, generate $C_{i+1,j}$

Note that, if $i = j$, then $i = j = k$ since at least of of i, j should be equal to k. Otherwise if $i < j$ then j should be equal to k, and if $i > j$ then i should be equal to k.

The candidate generation algorithm for cross associations is similar to the Apriori candidate generation. We provide the algorithm for candidate generation of cross associations in Figure 4.3 where the left component of a given cross association is extended. Generation of the right component and generation of both left and right components are similar to the generation of the left component, therefore they are not provided here.

4.2.3 Correctness of the Concurrent Algorithm

With the candidate generation scheme described above, all possible candidates are generated with L_k containing $L_{i,j} \neq \emptyset$, where $i, j \leq k$ and, at least one of i, j is equal to k.

Suppose that there exist two histories, H_1, and H_2, with frequent events $\{a, b, c, d\}$, and $\{p, q, r, s\}$, respectively. Initially L_1 will contain only $L_{1,1}$ and the cross associations in $L_{1,1}$ will be in the form $a \Rightarrow p$. After the second iteration, L_2 will consist of $\{L_{1,2}, L_{2,1}, L_{2,2}\}$. The cross associations in $L_{1,2}$ will be in the form $a \Rightarrow pq$, and $L_{1,2}$ will generate candidates in the form $a \Rightarrow pqr$. Note that $L_{1,2}$ can not generate candidates in the form $ab \Rightarrow pq$ since they are already considered

in the previous iteration as candidates in $C_{2,2}$. Similarly $L_{2,1}$ contains candidate itemsets in the form $ab \Rightarrow p$ and it will generate candidates in the form $abc \Rightarrow p$. It can not generate candidates in the form $ab \Rightarrow pq$ since they are already considered in $C_{2,2}$ during the previous iteration. In general for $i, j < k$, $L_{i,k}$ will only generate $C_{i,k+1}$, and $L_{k,j}$ will only generate $C_{k+1,j}$, because the candidates in $C_{i+1,k}$ and $C_{k,j+1}$ are processed during the previous iterations. Note also that, candidates in $C_{i+1,k+1}$ will be generated by $C_{i+1,k}$ and candidates in $C_{k+1,j+1}$ will be generated by $C_{k,j+1}$. $L_{k,k}$ will generate $C_{k+1,k}$, $C_{k,k+1}$, and $C_{k+1,k+1}$ since none of the candidates are processed in the previous iterations. This candidate generation scheme allows non-redundant generation of candidates.

Data Structures and the Counting Process:
We have used a hash-tree to store the candidate frequent event sets in the merge algorithm which was also used in [AS94]. Similarly, for the concurrent algorithm, a hash-tree is used to store the candidate frequent event sets together with the candidate cross associations. This way the storage of the candidate frequent event sets and candidate cross associations is overlapped. Initially, the candidate frequent event sets in H_1 are inserted to the hash-tree. This is followed by the insertion of the candidate cross associations. Then, the candidate frequent event sets in H_2 are inserted to the hash-tree by traversing the previously inserted cross associations.

The merge algorithm performs the counting efficiently on the hash-tree. Counting of the candidate frequent event sets and counting of the candidate cross associations are overlapped in the concurrent algorithm. At the k^{th} step, candidate frequent event sets of size k, and candidate cross associations $A_{H_1} \stackrel{C}{\Rightarrow} B_{H_2}$ with $|A_{H_1}|, |B_{H_2}| < k$ are generated and counted together. A frequent event set X of size k, at a transaction T in history H_1 is counted only if T supports the left component of a cross association $A_{H_1} \stackrel{C}{\Rightarrow} B_{H_2}$ such that $A_{H_1} \subset X$, and $|A_{H_1}| = k - 1$. Similarly, a frequent event set X of size k, at a transaction T in history H_2 is counted only if T supports the right component of a cross association $A_{H_1} \stackrel{C}{\Rightarrow} B_{H_2}$ such that $B_{H_1} \subset X$ and $|B_{H_1}| = k-1$. This prevents redundant counting of frequent event sets by cutting the counting process at an early stage when a transaction does not support the left or right component of any cross

Symbol	Meaning
$\|DB\|$	Database size in number of transactions
$\|T\|$	Average transaction length
$\|I\|$	Number of items
$skew$	skew in the size of the two histories
sup	minimum support threshold
$conf$	minimum confidence threshold (for confidence pruning)

Table 4.9: Main parameters used for performance experiments

association.

4.3 Experimental Work

We implemented the merge and concurrent algorithms using Perl programming language. We used both synthetic and real datasets. In order to assess the performance of the concurrent algorithm relative to the merge algorithm, we performed several experiments on a PC, running Linux operating system, with 512 MB of main memory, and a Pentium III processor that has a CPU clock rate of 500 MHz.

4.3.1 Performance Evaluation With Synthetic Data

In order to simulate correlated event histories we used the synthetic data that was produced by Han et al. [HPY00]. In the performance experiments, transaction identifiers are assumed to be timestamps and the data is analyzed in the order of transaction identifiers. Our main simulation parameters are listed in Table 4.9. The first data set contains 10K transactions and 1K items. Average transaction length is 20 items. The second data set contains 100K transactions and 10K items where the average transaction length is 25 items. To simulate the correlated histories, each transaction is divided into two transactions and two separate datasets are constructed. The skew parameter is used to determine how

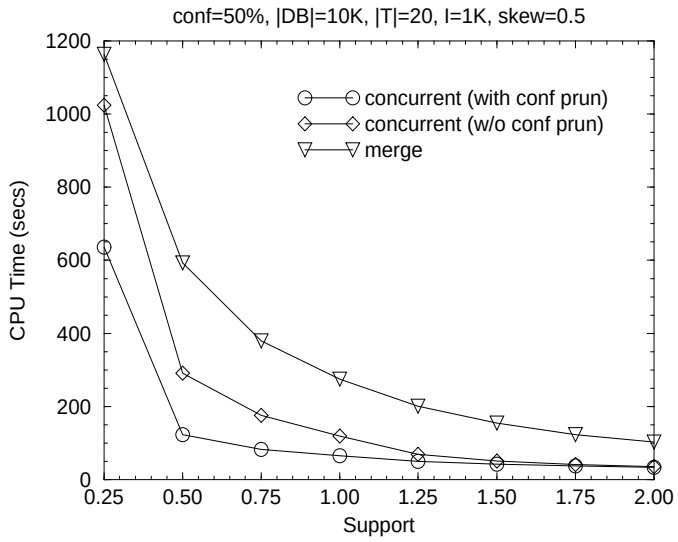

Figure 4.4: CPU time for different support values

the data will be partitioned into two histories. The skew is a real number in the range $[0, 1]$ and it is the ratio of the number of items in the first history to the number of items in the second history.

In the first experiment, we measured the CPU times produced by the merge and concurrent algorithms for varying support values. The results displayed in Figure 4.4 show that the concurrent algorithm with and without confidence pruning outperforms the merge algorithm for all support values. We also conducted an experiment to measure the effect of confidence pruning for various confidence values. The results of this experiment are shown in Figure 4.5. As expected, for increasing confidence values, more and more itemsets are pruned, leading to a sharp decrease in the CPU time.

The number of database scans is an important issue for the performance of the data mining algorithms. The number of scans needed for the merge algorithm is $k+1$, where k is the size of the largest frequent cross association spanning histories H_1 and H_2. The final scan is executed during the search for a cross association of size $k+1$ in the merged history. The number of scans needed for the concurrent algorithm is $Max(m, n) + 1$ where m is the maximal size of an event set A in H_1, where $A_{H_1} \stackrel{C}{\Rightarrow} B_{H_2}$ is a frequent cross association for some event set B in H_2.

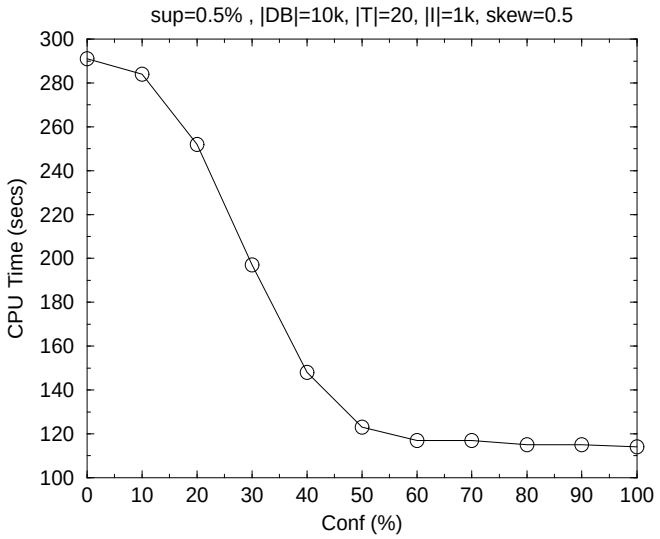

Figure 4.5: Effect of confidence pruning for different confidence values

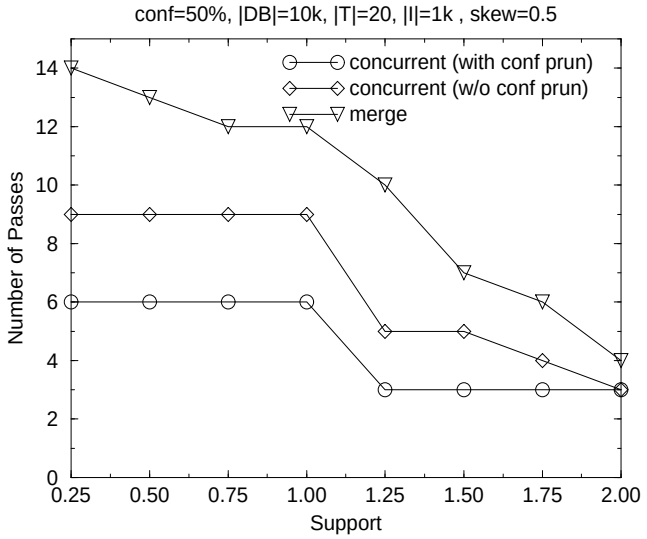

Figure 4.6: Number of database scans for different support values

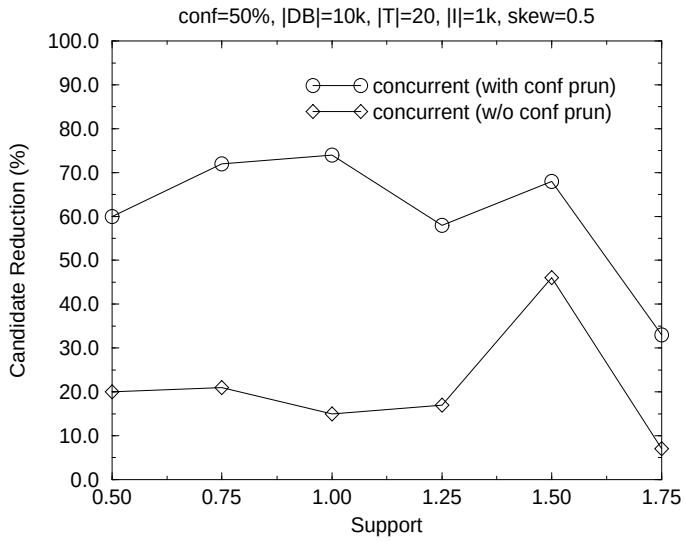

Figure 4.7: Reduction in the number of candidates for different support values

Similarly, n is the maximal size of an event set D in H_2 where $C_{H_1} \stackrel{C}{\Rightarrow} D_{H_2}$ is a frequent cross association for some event set C in H_1. As a result, the concurrent algorithm provides us the best performance in terms of the number of database scans. Figure 4.6 displays the number of scans performed by the algorithms as a function of the support value. As can be seen from the figure, the concurrent algorithm needs fewer database scans compared to the merge algorithm. When we also include confidence pruning strategies, the number of scans is further reduced as shown in the figure.

We measured the reduction in the number of candidates generated as a result of the pruning strategies for varying support values, and the results are shown in Figure 4.7. In this figure, the merge algorithm is considered as a base, and the percentage reduction in the number of candidates is shown for the candidates obtained after the second iteration of the algorithm. In the first iteration, the number of candidates are the same for both merge and concurrent algorithms.

The performance impact of skew in the two histories was also evaluated and the results are presented in Figure 4.8. The skew parameter describes the size of the first history H_1, relative to the size of the second history H_2, in terms of the average transaction lengths. For example, a skew of 0.1 indicates that the

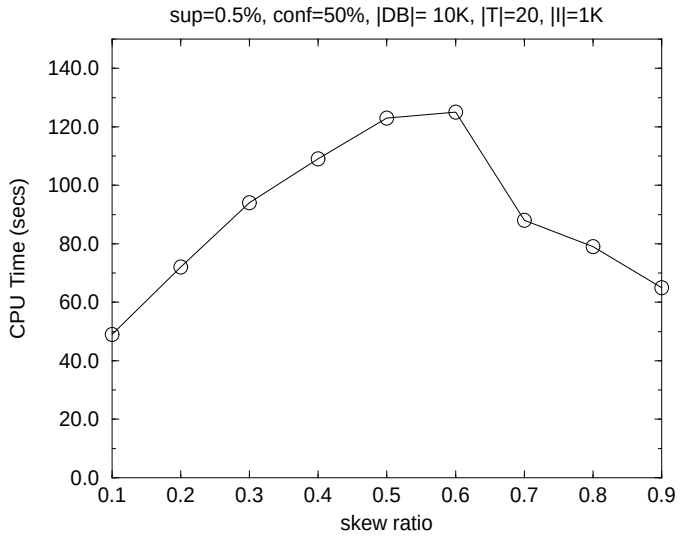

Figure 4.8: CPU time for different skew ratios

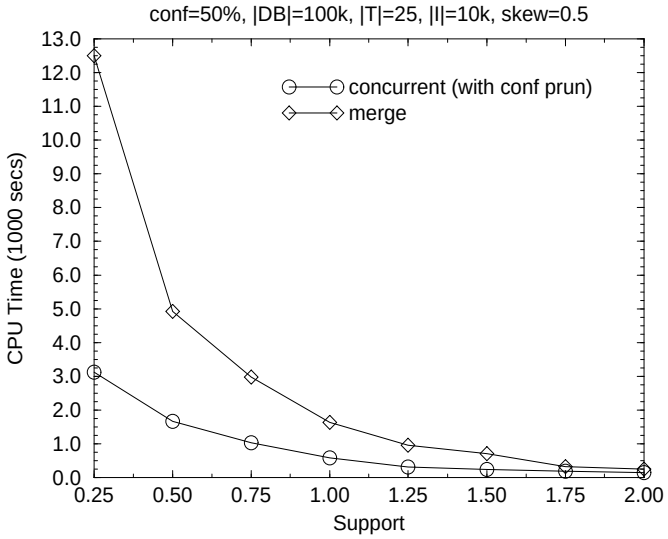

Figure 4.9: CPU time for different support values for larger data

ratio of the average transaction length in H_1 to the average transaction length in H_2 is 1 to 10. A cross association $A_{H_1} \stackrel{C}{\Rightarrow} B_{H_2}$ is skewed if the size of A_{H_1} is much smaller than the size of C_{H_2}, or vice versa. Cross associations obtained from two skewed event histories will also be skewed with a high probability. The concurrent algorithm is superior to the merge algorithm when there is no skew in the frequent cross associations. This is obvious since the merge algorithm scans the database $size(A_{H_1}) + size(B_{H_2})$ times, where on the average the concurrent algorithm scans the database only half of the number of database scans needed for the merge algorithm. If there is skew in the cross associations, then the merge and concurrent algorithms will perform comparably in terms of the number of database scans. However, the concurrent algorithm benefits from support and confidence pruning in case there is skew in the history sizes. Confidence pruning is more effective in case there is skew towards B_{H_2} which is also observed in the experiments. As can be observed in Figure 4.8, the CPU time increases as the skew is varied from 0.1 to 0.5. The CPU time decreases as the skew is varied from 0.5 to 0.9, however the CPU time at the skew level of 0.9 is slightly higher compared to the skew level of 0.1 since confidence pruning works better when the size of H_1 is smaller than the size of H_2. We did not depict the CPU time results of the merge algorithm with varying skew since the performance of the algorithm does not change much with skew. This is due to the fact that the merge algorithm does not exploit the pruning strategies used in the concurrent algorithm.

We tested the proposed algorithms on a larger dataset as well. The new data set consists of 100K transactions and 10K items. The average transactions length in this data set is 25 items. We measured the CPU times produced by the merge and concurrent algorithms for varying support values in this larger data set. The results displayed in Figure 4.9 show that the concurrent algorithm outperforms the merge algorithm for larger data sets as well.

4.3.2 Experiments on Real Data

We used the weather and power consumption data in the context of CIMEG [CIM], collected from Chicago area over one year, from 05/01/1997 to

Support	**Rule**
10%	$\{T5\} \Rightarrow \{k1, n4\}$
12 %	$\{T4\} \Rightarrow \{b4, o4\}$
11%	$\{T2\} \Rightarrow \{a5, i5, m3\}$
10%	$\{H4\} \Rightarrow \{i5, j5\}$
11%	$\{H2\} \Rightarrow \{j2, k1\}$

Table 4.10: Cross associations for 5 categories and 10% support

	category 1	**category 2**	**category 3**	**category 4**	**category 5**
Temp(F)	5 - 21	22 - 38	39 - 55	56 - 72	73 - 85
Hum(%)	39 - 51	52 - 64	65 - 77	78 - 90	91 - 99
comp. a	103 - 301	302 - 500	501 - 699	700 - 898	899 - 1094
comp. b	7 - 202	203 - 398	399 - 594	595 - 790	791 - 985
comp. i	198 - 318	319 - 439	440 - 560	561 - 681	682 - 801
comp. j	146 - 389	390 - 633	634 - 877	878 - 1121	1122 - 1363
comp. k	1 - 94	95 - 188	189 - 282	283 - 376	377 - 466
comp. m	4930 - 6850	6851 - 8771	8772 - 10692	10693 - 12613	12614 - 14530
comp. n	95 - 270	271 - 446	447 - 622	623 - 798	799 - 972
comp. o	374 - 646	647 - 919	920 - 1192	1193 - 1465	1466 - 1735

Table 4.11: Categorization of weather and power consumption measurements

04/30/98. The weather and power consumption data constitute two different sets of histories which are correlated. The weather data contain the temperature, wind speed, pressure and humidity. The power consumption data contain daily average power consumption of twenty anonymous companies, labeled "a" through "t". These companies have different scales of production. Residential power consumption was also contained in the data. We partitioned the data values into ranges, where each range corresponds to an event. This type of an approach was also used to create events out of stock price histories [BWJL98].

After running the cross mining algorithms, we obtained the cross associations among the weather events like temperature ranges and the power consumption ranges of different companies. Some of the interesting rules we obtained are shown in Table 4.10 for five ranges with 10% support, and 30% confidence thresholds. The capital letters T and H denote the temperature and humidity respectively, while the lower case letters denote the power consumption of different companies. The numbers attached to the letters denote the level (category) of power consumption, temperature, or humidity depending on the label they are attached to. We used a simple categorization technique utilizing the minimum and maximum values as described in Example 2.2.1. The categorization we used for the weather and power consumption measurements for the events that appear in Table 4.10 are provided in Table 4.11 to give a perspective. Company m is a very large scale company, and the others are large scale companies. Temperature is given in Fahrenheits, humidity is given by percentage, and the power consumption is given in kilowatt-hours, averaged over a day. As can be seen from the table, the power consumption of some companies are actually affected by temperature and humidity levels. The first rule in Table 4.10, i.e., $\{T5\} \Rightarrow \{k1, n4\}$ states that a temperature level of 5 is associated with a power consumption level of 4 in company n and a power consumption level of 1 in company k. Another rule, $\{T2\} \Rightarrow \{a5, i5, m3\}$ states that a temperature level of 2, which is a very low temperature, is associated with a very high power consumption of companies a and i, while being associated with a medium power consumption in company m which is a very large scale company.

4.4 Summary

In this chapter, we have introduced a new data mining problem for mining associations among events spanning different histories, called cross associations. We have presented two algorithms, namely the merge and concurrent algorithms, for extracting cross associations from two correlated event histories. We have evaluated the performance of the proposed algorithms on synthetic data sets, and conducted experiments on different support and confidence values. The performance results obtained show that a considerable performance improvement is provided by the concurrent algorithm with pruning strategies over the merge algorithm. The experiments on synthetic data have proven that processing two correlated histories separately is a better approach compared to merging and mining the histories. We have also conducted experiments on the real life weather and power consumption data.

Chapter 5

Fuzzy and Proactive Rules

In the preceding two chapters, we have explained how a single and correlated event histories can be analyzed to obtain:

- event similarities that describe binary relationships among events,
- sequential event patterns that represent the sequences of events appearing frequently in the event history, and
- cross associations that describe associations among events spanning multiple histories.

In this chapter, we show how all these different types of relationships among events can be used for event detection and rule execution in active database systems. In Section 5.1, we explain how the similarities among the events that are captured in a matrix can be used to divide the events in a system into sub-event-sets modeled as fuzzy sets. We further demonstrate in Section 5.2 how event similarities and fuzzy event sets can be used for similarity based event detection and fuzzy rule execution in active databases. In Section 5.3, we show how the event patterns like associations and cross associations can be used for predictive event detection and proactive rule execution in active database systems.

5.1 Construction of Fuzzy Event Sets

Fuzzy event sets are fuzzy sets where the elements are events, and each event has a degree of membership to the fuzzy event set. A fuzzy event set, E_F, over a universal event set ξ is a tuple of the form $< E, \mu >$ where $E \subseteq \xi$ and μ is the membership function that describes the degree of membership of the events in E to E_F. In Chapter 3, we have explained the construction of the sequential proximity matrix that describes the similarities of events in terms of the proximity of event signaling. In the following subsection, we discuss how the sequential proximity matrix is used for constructing fuzzy event sets and their membership function.

5.1.1 Partitioning The Event Space Using Sequential Proximity Matrix

The fuzzy event set construction is done by partitioning the event space using the sequential proximity matrix. The events in each partition have a degree of membership greater than 0, and the rest of the events have 0 as their degree of membership to the fuzzy event set. The exact membership degree functions are provided later on. The problem here is to partition the set of events into k subsets such that the resulting subsets are balanced, the total proximities of the events in all subsets are maximized, and the total similarities among the events belonging to different subsets are minimized. This problem is similar to the *graph partitioning problem*. Given a graph G with costs on its edges, partition the nodes of G into subsets no larger than a *given maximum size*, so as to minimize the total cost of the edges cut. Following is the formal definition of the graph partitioning problem [KL70].

Definition 5.1.1 *Let G be a graph of n nodes, of size (weights) $w_i > 0, i = 1, ..., n$. Let p be a positive number, such that $0 < w_i \leq p$ for all i. Let $C = c_{ij}, i, j = 1, ..., n$ be a weighted connectivity matrix describing the edges of G.*

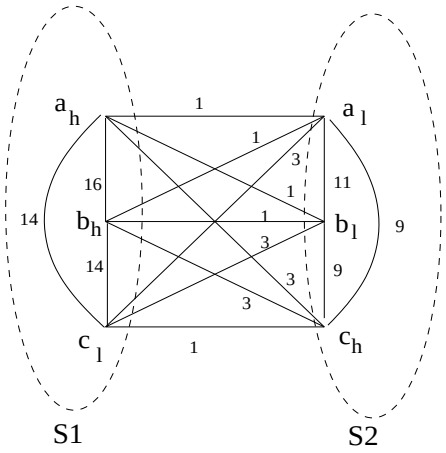

Figure 5.1: A sample event graph

Let k be a positive integer. A k-way partition of G is a set of nonempty, pairwise disjoint subsets $v_1, ..., v_k$ of G, such that $\cup_{i=1}^{k} v_i = G$. A partition is admissible if $|v_i| \leq p$ for all i, where $|v_i|$ stands for the size of the set v_i and equals to the sum of the sizes of all the elements of v_i.

An exact solution to this problem is shown to be NP-hard in [BJ92].

We simplify the graph partitioning problem defined above so that the weights of the nodes are all equal to 1. We also set p to the constant value $\frac{n}{k}$ for a balanced partition. The number of partitions, i.e., the number of fuzzy event sets is application dependent. The weighted connectivity matrix is the sequential proximity matrix described in Section 3.1.1.

The algorithms provided by Kernighan and Lin assume undirected graphs. In our case the proximity of event e_i to event e_j may be different from the proximity of e_j to e_i. We may consider our sequential proximity matrix as a directed complete graph where the events correspond to nodes and weights of directed edges of the form $e_i \rightarrow e_j$ correspond to the matrix entry $M[i, j]$. We convert our directed graph to a single undirected graph by replacing the directed edges $e_i \rightarrow e_j$ and $e_j \rightarrow e_i$ with an undirected edge $< e_i, e_j >$. Weight of the edge $< e_i, e_j >$ is the sum of the weights of $e_i \rightarrow e_j$ and $e_j \rightarrow e_i$. Collapsing the directed edges connecting the same nodes to an undirected edge does not affect

the outcome of the partition since either both of the edges or none of them are in the cutset.

METIS Software package can be used for graph partitioning [KK]. The event graph that results from the sequential proximity matrix given in Figure 3.5 is provided in Figure 5.1. The nodes of the graph represent the events and the edges between the nodes represent the similarities between the corresponding events. The undirected edges of the graph are obtained by summing the weights of the edges with the same end-points. Two event sets resulting from the partitioning step are shown in Figure 5.1 by ellipses with dashed lines. The first set, S_1 consists of the events a_h, b_h, and c_l while the second set, S_2 consists of a_l, b_l, c_h where a and b here correspond to the power consumption of industry and c corresponds to a residential customer. This grouping may show us that in the snapshot of the power consumption event history, high power consumption of a residential customer corresponds to low consumption of industry. This situation may occur in summer where lots of ACs are operated in the houses and the electricity price goes up forcing the large scale companies to lower their production resulting in lower power consumption in the industry.

5.1.2 Computation of Membership Functions

One method of obtaining the membership functions for fuzzy event sets is to rely on the experts of the particular application which is usually the case in fuzzy control. Since we have the sequential proximity matrix ready at hand, we can utilize it to construct fuzzy event sets. As we have explained in Section 5.1.1, fuzzy event sets are constructed using a graph partitioning algorithm. Graph partitioning is used to cluster strongly related events. We can deduce the *average coherence* among the events in a fuzzy event set by taking the average proximity of all event pairs. The proximity between the event pair (e_i, e_j) is denoted by $proximity(e_i, e_j)$ and it is the value stored in the i^th row and j^th column in the sequential proximity matrix. The average coherence is further used for determining the membership function of a fuzzy event set.

Definition 5.1.2 *Coherence of an event e_i in a fuzzy event set S is denoted by $C_S(e_i)$, and defined as the average proximity of event e_i to the rest of events in S. Coherence is calculated as follows:*

$$C_S(e_i) = \frac{\sum_{e_j \in S \wedge i \neq j} proximity(e_i, e_j)}{|S|-1}$$

where $|S|$ is the cardinality of the fuzzy event set S and $proximity(e_i, e_j)$ is the proximity of e_i to e_j. We exclude the coherence of an event to itself in coherence calculation.

Coherence of an event e, in a fuzzy event set S specifies how closely e is related to S.

Definition 5.1.3 *Average Coherence for a fuzzy event set S is denoted by AC_S and calculated as follows:*

$$AC_S = \frac{\sum_{ei \in S} C_S(e_i)}{|S|}$$

Definition 5.1.4 *Coherence Deviation of an event e_i, in a fuzzy event set S is denoted by $CD_S(e_i)$ and calculated as:*

$$CD_S(e_i) = C_S(e_i) - AC_S.$$

Maximum absolute coherence deviation for a fuzzy event set S, denoted by $MaxCD_S$, is the maximum of absolute coherence deviations in that fuzzy event set, i.e., $MaxCD_S = max_{e_i \in S}\{|CD_S(e_i)|\}$.

Coherence Deviation is a measure of how far is the coherence of an event from the average coherence.

Definition 5.1.5 *Mean Absolute Deviation of a fuzzy event set S is denoted by MAD_S and calculated as:*

$$MAD_S = \frac{\sum_{ei \in S} |CD_S(e_i)|}{|S|}$$

Mean Absolute Deviation is an approximate measure of how close are the different pairwise similarity values in a fuzzy event set.

Lemma 5.1.1 *The formula,* $\frac{CD_S(e_i)\times MAD_S}{2\times MaxCD_S^2} + 0.5$ *produces a number in the range* $[0,1]$ *for an event* e_i *when* $MaxCD_S \neq 0$.

Proof

For any event e_i, $CD_S(e_i)$ is in the range $(-\infty, \infty)$, while $MaxCD_S$ and MAD_S are both in the range $[0, \infty)$. By definition, $CD_S(e_i) \leq MaxCD_S$, and also $MAD_S \leq MaxCD_S$. Therefore, we have

$$-1 \leq \frac{CD_S(e_i)}{MaxCD_S} \leq 1 \text{ and } -1 \leq \frac{MAD_S}{MaxCD_S} \leq 1.$$

Since $MaxCD_S \neq 0$, both of the fractions are defined.

$-1 \leq \frac{CD_S(e_i)}{MaxCD_S} \leq 1$ and $-1 \leq \frac{MAD_S}{MaxCD_S} \leq 1$ together imply that

$$-1 \leq \frac{CD_S(e_i)\times MAD_S}{MaxCD_S \times MaxCD_S} \leq 1.$$

This leads to, $-0.5 \leq \frac{CD_S(e_i)\times MAD_S}{2\times (MaxCD_S)^2} \leq 0.5$.

When we add 0.5 to the inequality, we get

$$0 \leq \frac{CD_S(e_i)\times MAD_S}{2\times (MaxCD_S)^2} + 0.5 \leq 1.$$

Thus, $\frac{CD_S(e_i)\times MAD_S}{2\times (MaxCD_S)^2} + 0.5$ is in the range $[0,1]$ ■

The membership function of a fuzzy event set S is denoted by μ_S and calculated as follows:

$$\mu_S(e_i) = \left\{ \begin{array}{ll} 0 & if\ e_i \notin S \\ \frac{CD_S(e_i)\times MAD_S}{2\times MaxCD_S^2} + 0.5 & if\ e_i \in S \wedge \\ & MaxCD_S \neq 0 \\ 1 & if\ e_i \in S \wedge \\ & MaxCD_S = 0 \end{array} \right\} \quad (5.1)$$

An example for the membership calculation for the graph in Figure 5.1 is given below:

Example 5.1.1 *There are two fuzzy event sets,* S_1 *and* S_2*, in Figure 5.1. We calculate the membership function for* S_1 *as follows:*

1. *First we need to calculate the coherence for each event and find the average coherence.*

   - *The coherence of event* a_h *is* $C_{S_1}(a_h) = \frac{similarity(a_h,b_h)+similarity(a_h,c_l)}{|S_1|-1}$ *which equals to* $C_{S_1}(a_h) = \frac{8.0+6.5}{2} = 7.25$.
   - $C_{S_1}(b_h) = \frac{similarity(b_h,a_h)+similarity(b_h,c_l)}{|S_1|-1}$ *which is* $C_{S_1}(b_h) = \frac{8.0+6.5}{2} = 7.25$.
   - $C_{S_1}(c_l) = \frac{similarity(c_l,a_h)+similarity(c_l,b_h)}{|S_1|-1}$ *which is* $C_{S_1}(c_l) = \frac{7.5+7.5}{2} = 7.5$.

2. *Average coherence for* S_1 *is calculated as:*
   $AC_{S_1} = \frac{C_{S_1}(a_h)+C_{S_1}(b_h)+C_{S_1}(c_l)}{|S_1|} = \frac{7.25+7.25+7.5}{3} = 7.33$

3. *Coherence deviation for each event is calculated to find the maximum coherence deviation.*

   - $CD_{S_1}(a_h) = C_{S_1}(a_h) - AC_{S_1} = 7.25 - 7.33 = -0.08$
   - $CD_{S_1}(b_h) = C_{S_1}(b_h) - AC_{S_1} = 7.25 - 7.33 = -0.08$
   - $CD_{S_1}(c_l) = C_{S_1}(c_l) - AC_{S_1} = 7.5 - 7.33 = 0.17$
   - *Maximum absolute coherence deviation is,* $MaxCD_{S_1} = 0.17$

4. *Mean Absolute Deviation is calculated as:*
   $MAD_{S_1} = \frac{|-0.08|+|-0.08|+|0.17|}{3} = 0.11$

After obtaining all the intermediate values needed, we can calculate the membership values of the individual events as:

- $\mu_{S_1}(a_h) = \frac{CD_{S_1}(a_h)\times MAD_{S_1}}{2\times MaxCD^2_{S_1}} + 0.5$ *which is* $\frac{-0.08\times 0.11}{2\times(0.17)^2} + 0.5 = 0.35$

- $\mu_{S_1}(b_h) = \frac{-0.08\times 0.11}{2\times(0.17)^2} + 0.5$ *which is also* 0.35.

- *Finally the membership value of* c_l *is*
  $\mu_{S_1}(c_l) = \frac{0.17\times 0.11}{2\times(0.17)^2} + 0.5 = 0.82$

5.2 Application of Sequential Proximity Matrix and Fuzzy Event Sets to Active Database Systems

Fuzzy rules have been involved in many areas from control to expert systems. The application of fuzzy concepts into active database systems has attracted the attention of some researchers; however, we cannot say that a considerable amount of research has been conducted addressing this issue. This section is devoted to the description of fuzzy rule execution in active database systems. In the following subsections we describe how fuzzy event sets are used for fuzzy rule execution and how sequential proximity matrix is used for similarity based event detection in active database systems.

5.2.1 Fuzzy Rule Execution and Fuzzy Event Sets

In the previous sections, we have shown how we can partition the whole event space ξ into fuzzy event sets. Scenarios in the active database context are the aggregation of fuzzy event sets and rules. The idea of scenarios comes from the need to group rules into sets corresponding to different situations such as a "high power consumption scenario" in the summer. Dividing the rule space into scenarios has many advantages. First of all, dealing with a group of related rules is much easier than dealing with all the rules. Also, the number of rules triggered can be tuned by pruning the triggered rules using the membership function of the scenario as will be explained later on. This combined effect of decreased number of events and rules in consideration will improve the efficiency and effectiveness of the system.

Assume that we have a set of fuzzy event sets $\{E_{F_1}, E_{F_2}, ..., E_{F_k}\}$ where $E_{F_i} =< E_i, \mu_i >$. Assume also that the set of rules in the system is R where each rule is a triple of the form $< e, c, a >$, e being the event, c being the condition, and a being the action. Let $events(R_i) = \{e | r :< e, c, a > \in R_i\}$ be the events of

the rule set R_i. We partition the set of rules R, into subsets, $\{R_1, R_2, ..., R_k\}$ such that $events(R_j) \subseteq E_j$ for some fuzzy event set $E_{F_j} =< E_j, \mu_j >$ and $R_1 \cup R_2 \cup ... \cup R_k = R$ and for $1 \leq i < j \leq k$, $events(R_i) \cap events(R_j) = \emptyset$. The scenario corresponding to R_i, and E_{F_j}, $1 \leq i, j \leq k$, is a triple $< R_i, E_j, \mu_j >$.

Example 5.2.1 *Assume that we have the set of rules*
$R = \{r_1 :< a_h, c_1, a_1 >, r_2 :< b_h, c_2, a_2 >, r_3 :< a_l, c_3, a_3 >, r_4 :< b_l, c4, a_4 >\}$
and the two fuzzy event sets
$E_{F_1} =< \{a_h, b_h, c_l\}, \mu_1 >$ *and* $E_{F_2} =< \{a_l, b_l, c_h\}, \mu_2 >$.
We first partition R *into* $R_1 = \{r_1 :< a_h, c_1, a_1 >, r_2 :< b_h, c_2, a_2 >\}$ *and*
$R_2 = \{r_3 :< a_l, c_3, a_3 >, r_4 :< b_l, c4, a_4 >\}$.
The corresponding scenarios are $< R_1, E_1, \mu_1 >$ *and* $< R_2, E_2, \mu_2 >$.

Event signaling is done by considering the membership degree of the event parameter in the fuzzy event. We use the fuzzy event structure described in [BW97] where a primitive event is a tuple, $e :< e_c, e_f >$, consisting of a crisp part e_c which is the crisp parameter coming from the system and a fuzzy part e_f which denotes the fuzzy event.

Definition 5.2.1 *The strength of an event* $e :< e_c, e_f >$ *for the rule* $r :< e, c, a >$ *in scenario* $R_{S_i} :< R_i, E_i, \mu_i >$ *is defined as :*

$$strength(e,r) = \mu_i * \mu_{e_f}(value(e_c))$$

where $value(e_c)$ *is the value of the crisp event detected, and* μ_{e_f} *is the membership function of the fuzzy event* e_f. *The value of an event can be* e_c *itself in case* e_c *is coming from a sensor. Or the value can be a parameter to the crisp event such as a database update event and the updated value.*

Each rule has a firing threshold which is used to decide if a rule will be fired or not. In order to decide whether a rule r will be fired in response to the signaling of a fuzzy event e, the strength of event e for rule r is calculated and the result is compared to the threshold value for rule r. If the result is greater than or equal

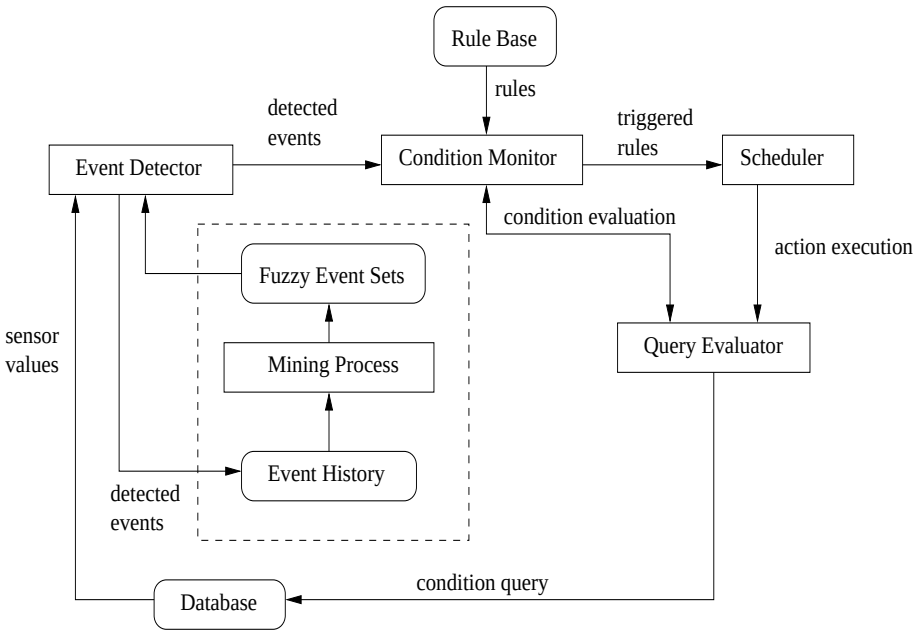

Figure 5.2: Event detection and rule execution model

to the threshold value, then the rule is fired. Threshold values of rules can be changed dynamically to tune to particular scenarios.

Prioritization of rules in an active database system is very important. Some rules may be more urgent than the others. The priority of rules can be set by the people who defined the rules. Another method is to use the events of the rules to set the priorities. Consider two rules whose events are "significant temperature change" and "significant pressure change". If the membership value of the significant temperature change is much higher than the membership value for the pressure change, the rule with event "significant temperature change" should be given higher priority of execution since late execution of that rule may be hazardous. Strength of the rules is a good indicator of rule priorities and they can be used to order the rules during rule execution. Prioritization based on the strength of the rules will give higher priority of execution to the rules with higher strength values.

The rule execution model provided in [PD99] is modified and depicted in Figure 5.2. Events are detected by the event detector using the sensor values

that are continuously stored in the database. Rules corresponding to the detected events are obtained from the rule base and their conditions are evaluated. Those rules whose conditions are satisfied are scheduled for execution by the scheduler. As an extension to this model, we add the component in the rectangle with dashed lines. In this modified model, detected events are stored in the event history and they are fed to the mining process periodically. The mining process together with the partitioning, produces the fuzzy event sets which are further used by the event detector. Utilization of the fuzzy event sets and sequential proximity matrix is explained in the following subsections.

5.2.2 Similarity Based Event Detection

Detection of similar events upon an event occurrence is something very useful when the cost of missing events is very high in supported applications, like a nuclear reactor control system. Assume that an event such as "update in temperature level" is detected. Events with a high similarity degree, like "update in pressure level" should also be signaled automatically. This way, the system can predict that an event which escaped the event detection process has occurred. In a power producing and selling company, occurrence of some events can imply the occurrence of some other events. Advantage of similarity based event detection in such a system is that, the dealers may assume that the events similar to the signaled events will occur and can take the necessary actions in advance.

In similarity based event detection, when an event is signaled, other events which are similar to it should also be fired. Similarity based event detection considers only primitive events. In order to facilitate similarity based event detection, the sequential proximity matrix explained in Section 3.1.1 is utilized. We also need similarity thresholds in order to avoid the system to continuously detect irrelevant events via similarity based event detection.

Definition 5.2.2 *Similarity threshold for an event e in a scenario S is the minimum similarity requirement for the events in S to be detected automatically by e via similarity based event detection.*

Similarity threshold for an event e in a scenario S can be set to be the coherence of e in S which is explained in Section 5.1.2. Upon the detection of an event, e, all the events in the same scenarios as e whose similarity to e is greater than the similarity threshold will be detected automatically. Similarities of the events in a scenario are the proximity values in the sequential proximity matrix. An example would be helpful in explaining similarity based event detection.

Example 5.2.2 *Consider the events and the sequential proximity matrix in Figure 3.5 and the fuzzy event sets in Figure 5.1. Assume that the current scenario is S_1 and a_h is detected. Coherence of a_h in S_1 was calculated in Example 5.1.1 as 7.25. Among the rest of the events in S_1 which are b_h and c_l, only b_h will be detected by similarity based event detection since $proximity(a_h, b_h) = 8.0 > 7.25$, but c_l will not be detected since $proximity(a_h, c_l) = 6.5 \leq 7.25$.*

Conventional event detection in active database systems is a special case of similarity based event detection where the similarity relation among the events is an identity relation and similarity thresholds are equal to infinity.

5.3 Predictive Event Detection and Proactive Rule Execution in Active Database Systems

In this section, we describe how the event patterns obtained by mining event histories can be used for predictive event detection and proactive execution of rules. The rule structure and event detection for predictive event handling are described in Section 5.3.1 and Section 5.3.2, respectively. Section 5.3.3 explains the condition evaluation and action execution in case of predictive event detection.

5.3.1 Rule Structure

In a proactive database system, there are three types of rules. The first type is the standard Event-Condition-Action (ECA) rule. Each ECA rule is associated with

two rules, namely, *proactive* and *compensating* rules which are the second and third types, respectively. Although the system views all the rules as independent from each other, we will consider a rule together with its corresponding proactive and compensating rule as a triple of the form (R, R_p, R_c) where R is the ECA rule, R_p is the proactive rule, and R_c is the compensating rule. R, R_p, and R_c are represented by (e, c, a), (e_{virt}, c, a_{pre}), and $(\neg e_{virt}, c_{comp}, a_{comp})$, respectively. A *proactive rule* consists of an event which is the virtual event of the corresponding ECA rule, a condition which is the same as the condition of the ECA rule, and a preaction. The compensating rule of an ECA rule has an event that is the negation of the virtual event of the corresponding proactive rule, a standard condition which checks whether the condition evaluation of the previous signaling of the same virtual event is true, and a compensating action. This way we can implement proactive capabilities seamlessly using the ECA rule paradigm, without changing the rule structure of the existing active database systems.

A sample rule is given in Figure 5.3. This rule describes the action that will be taken when the temperature level is high. To prevent a shortage in electricity supply, electricity producers sell cheap electricity to consumers with the condition that they may cut down the electricity for nonvital units in case of peak demand like ACs in universities that are not vital. The rule in Figure 5.3 describes a case where the power supply of ACs in a university will be cut down in case of high temperatures. Such rules are obtained by using the cross associations that show the correlation between the high temperature and power consumption of individual industries.

5.3.2 Event Detection

Events are detected in discrete time periods. The currently detected events are used together with the extracted patterns and sequences to predict the events in the future. We assume that proactive event handling is performed at the primitive event level.

The set of currently signaled events in the system consists of two groups, *real*

RULE
event: When the temperature level is *high*
EC Coupling: *immediate*
condition: Check whether it is weekend
CA Coupling: *immediate*
Action: Curtail the electricity for the nonvital ACs of the nearby university

PROACTIVE RULE
event: When the temperature level is *high* (virtual)
EC Coupling: *immediate*
condition: Check whether it is weekend
CA Coupling: *immediate*
PreAction: Send a warning message to the nearby university indicating that the electricity for the nonvital ACs will be curtailed

COMPENSATING RULE
event: *NOT* when the temperature level is *high* (virtual)
EC Coupling: *immediate*
condition: Check whether the condition of the proactive rule was *true*
CA Coupling: *immediate*
Compensation Action : Withdraw the previously sent warning message

Figure 5.3: Sample proactive rule for power management

events and *virtual events* which are defined as follows:

Definition 5.3.1 *Real events are the events that are actually signaled by the state changes or sensors of a system.*

Definition 5.3.2 *Virtual events are the events signaled via predictive or similarity based event detection. Primitive virtual events are the duals of the real events. Negation of a virtual event is also a virtual event which is signaled to indicate the missing occurrence of a real event.*

Virtual events are associated with a time frame called *event detection frame.*

Definition 5.3.3 *Event detection frame of a virtual event is the time interval during which the actual occurrence of the corresponding real event is monitored.*

Size of an event detection frame could be fixed for all virtual events or could be different for different event classes. Transaction boundaries is a good determiner of event detection frames. Event detection frame for a virtual event could be set from the virtual event signaling time to the transaction commit or abort point. At the end of the transaction, compensating actions could be performed prior to the commit or abort operation. If the virtual event is not actually signaled in the event detection frame, a negation event of this virtual event is signaled, so that the effects of the false signaling could be undone or compensated.

5.3.3 Condition Evaluation and Action Execution

We can immediately evaluate the conditions of the rules fired by predictive or similarity-based event detection, and evaluate their actions; however, in some cases it may be more meaningful to evaluate the condition and defer the action until the occurrence of the actual event. This is especially very meaningful when the action has observable effects and cannot be rolled back.

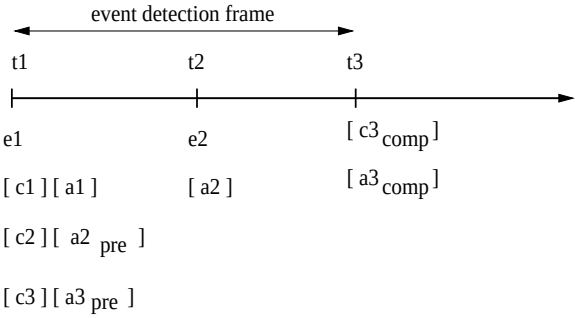

Figure 5.4: Sample proactive rule execution

One problem with deferred action execution is that conditions may no longer be true when the actual event is signaled. To alleviate this problem we may define deadlines for condition validity and execute the actions with valid conditions only, otherwise check the condition again. However, if the deadline of the action is very close, then we may rely on the current condition evaluation. For the sample rule given in Figure 5.3, it is obvious that the condition will not be false since the days of the week are fixed. Therefore, we can use the result of the condition for the actual run of the action.

A more reliable way for rule execution with predictive event detection is to use different actions in case of real event and virtual event detection. When the virtual event is detected, the corresponding proactive rule is fired instead of the actual rule. Action (which is called preaction) of the proactive rule is executed if the condition evaluates to true. Upon the detection of the real event, the action is executed. When the event never occurs inside a time frame, the event detection module signals a negation of the corresponding virtual event which fires the compensating rule. Compensating action is executed if the action of the proactive rule for that event is executed. Referring to Figure 5.4 for the rules:

$$R_1 = ((e_1, c_1, a_1), (e_{1_{virt}}, c_1, a_{1_{pre}}), (\neg e_{1_{virt}}, c_{1_{comp}}, a_{1_{comp}}))$$
$$R_2 = ((e_2, c_2, a_2), (e_{2_{virt}}, c_2, a_{2_{pre}}), (\neg e_{2_{virt}}, c_{1_{comp}}, a_{2_{comp}}))$$
$$R_3 = ((e_3, c_3, a_3), (e_{3_{virt}}, c_3, a_{3_{pre}}), (\neg e_{3_{virt}}, c_{1_{comp}}, a_{3_{comp}}))$$

assume that event e_1 occurs at time t_1 and we have a rule saying that occurrence

Regular Rule	Proactive or Compensating rule
immediate	immediate, detached, or deferred
detached	detached or deferred
deferred	deferred

Table 5.1: Coupling mode options

of event e_1 is followed by the occurrence of events e_2 and e_3. Predictive event detection mechanism tells us that events e_2 and e_3 will occur with a high probability and therefore signals virtual events $e_{2_{virt}}$ and $e_{3_{virt}}$. Conditions of the rules whose events are $e_{2_{virt}}$ and $e_{3_{virt}}$ are checked and their preactions are executed if the corresponding conditions are true. Assuming that the event detection frame spans from t_1 to t_3, and event e_2 actually occurs at time t_2 which is still in the event detection frame, we first check for the validity time of condition c_2. If c_2 is still valid, then we execute the action. If the condition check is no longer valid, we need to check it again. Action a_2 is executed if the condition evaluates to true. If event e_3 does not occur during the event detection frame of $e_{3_{virt}}$, then condition $c_{3_{comp}}$ is evaluated which checks whether the preaction for the corresponding proactive rule, i.e., $a_{3_{pre}}$, is executed. Compensating action $a_{3_{comp}}$ is executed in our example since $a_{3_{pre}}$ has already been executed.

5.3.4 Coupling Modes and Priority Assignment in Proactive Rule Execution

Coupling modes are an important functionality provided by active database systems and should be considered for proactive rule execution. The coupling modes[1] of the proactive and compensating rules should be less restrictive compared to the corresponding rule. Coupling modes in the order of increasing strictness are: $immediate > detached > deferred$. According to this order, semantically a regular rule should not have a less restrictive coupling mode than that of its corresponding proactive and compensating rule. Also, a proactive rule should not

[1]By coupling mode we mean both EC and CA coupling modes.

have a less restrictive coupling mode than that of its corresponding compensating rule. The coupling modes that could be assumed by the proactive and compensating rules for the given coupling mode of a regular rule are shown in Table 5.1. Let's first analyze the case of immediate and deferred coupling modes. If the coupling mode of a rule is deferred, then coupling mode of the corresponding proactive or compensating rule should also be deferred. This makes sense since immediate and deferred coupling modes prioritize the rule execution in a sense and it is not meaningful to give a higher priority to the proactive or compensating rule than the priority of the actual rule. Also, it does not make sense to assign deferred coupling mode to the proactive rule and immediate coupling mode to the compensating rule, since logically compensation will follow preaction.

In the case of detached coupling mode, we should make sure that the commit order of the rules should be consistent. This could be achieved by a proper priority assignment mechanism.

The priorities of the proactive and compensating rules should also be lower than the priority of the corresponding rule since they refer to the future states of the system. Priority assignment can be done according to the strength of the estimate.

5.4 Summary

In this chapter, we have proposed some methods for automated construction of fuzzy event sets which are sets of events where each event has a degree of membership to a set. Fuzzy event sets are constructed by using the sequential proximity matrix. Construction of fuzzy event sets is approximated to the well known graph partitioning problem, and fuzzy event sets are constructed using graph partitioning algorithms. The sequential proximity matrix is utilized for similarity based event detection in active database systems. We have also introduced in this chapter, the notion of proactive event handling which is based on

predictive event detection. Introduction of predictive event detection and proactive rule execution has necessitated the revision of standard event detection and rule execution processes in active database systems. We have provided a framework for predictive event detection and introduced a rule structure to support proactive rule execution in a seamless manner.

Chapter 6

Broadcast Data Management

In a data broadcast environment, client request events to data items are stored in a broadcast history which is a single history consisting of the events from the same domain. In Chapter 3, we have discussed how a broadcast history can be mined to obtain client request event patterns. The client request event patterns mimic the data items requested in sequence by clients. In Chapter 5, we have described how the similarities among the events extracted from a single event history can be used to group events by using graph partitioning techniques. In this chapter, we use the same idea of grouping events. However, in case of data broadcast, we obtain sequential patterns of client request events which can be of length larger than 2. This model of representing the relationships among events differs from representing the similarities which is a binary relationship. For that reason, we use hypergraph partitioning techniques, rather than graph partitioning techniques, to divide the set of broadcast data items into subsets of related data items. Predictive event detection and proactive rule execution techniques described in Chapter 5 are applied to the broadcast environment with the aim of improving the system performance in terms of data access delay. Predictive event detection is achieved via the event patterns obtained from an event history using data mining techniques. Predictive event detection in the context of broadcast environments is used to predict future client data request events. Provided that the future data request event can be predicted, the data items that are predicted to be requested

session no	requests
1	$e\ b$
2	$d\ a$
3	$e\ b\ c$
4	$d\ a\ f$

Table 6.1: Sample database of user requests

rule	confidence	support
$e \rightarrow b$	100%	50%
$d \rightarrow a$	100%	50%

Table 6.2: Sample rules

in the near future are loaded to the client cache proactively using proactive rule execution methods. Proactive loading of the data items into client cache is called prefetching. The motivation for broadcast organization, and proactive loading of data items to client cache is provided in Section 6.1. We discuss the conversion of sequential patterns to proactive rules in Section 6.2. Broadcast data organization using sequential patterns of events is described in Section 6.3. We explain the details of future data request prediction techniques, and prefetching of data items into client cache in Section 6.4. In Section 6.5, we show through performance experiments that the prefetching techniques actually provide better performance than the state of the art prefetching techniques .

6.1 Motivation

Suppose that we have a set of data items $\{a, b, c, d, e, f\}$. A sample broadcast history over these items consisting of four sessions is shown in Table 6.1. The sequences extracted from this history with minimum support 50% are (e, b) and (d, a). The rules obtained out of these sequences with 100% minimum confidence are $e \rightarrow b$ and $d \rightarrow a$, as shown in Table 6.2. Two broadcast data organizations are depicted in Figure 6.1. Figure 6.1(a) shows a broadcast schedule without

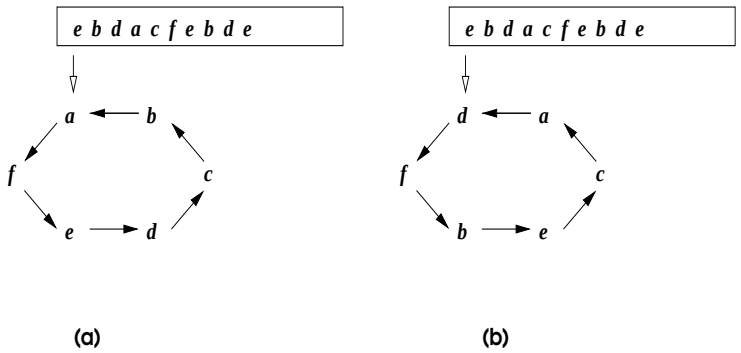

Figure 6.1: Effect of broadcast data organization

any intelligent preprocessing, and Figure 6.1(b) shows a schedule where related items are grouped together and sorted with respect to the order of reference. Assume that the disk is spinning counter-clockwise and consider the following client request pattern, $e, b, d, a, c, f, e, b, d, e$, also shown in Figure 6.1. For this pattern, if we have the broadcast schedule (a, b, c, d, e, f) which does not take into account the rules, the total waiting time for the client will be 4+3+2+3+2+3+5+ $3+2+3 = 30$, and the average latency will be $30/10 = 3.0$ broadcast units, which is an expected value, i.e., half of the disk size. However if we partition the items to be broadcast into two groups with respect to the sequential patterns obtained after mining, then we will have $\{a, c, d\}$ and $\{b, e, f\}$. Note that, data items that appear in the same sequential pattern are placed in the same group. When we sort the data items in the same group with respect to the rules $d \rightarrow a$ and $e \rightarrow b$, we will have the sequences (d, a, c) and (e, b, f). If we organize the data items to be broadcast with respect to these sorted groups of items, we will have the broadcast schedule presented in Figure 6.1(b). In this case, the total waiting time for the client for the same request pattern will be $3+1+2+1+1+3+4+1+2+1 = 19$, and the average latency will be $19/10 = 1.9$, which is much lower than 3.0.

Another example that demonstrates the benefits of rule based prefetching is shown in Figure 6.2. The first two requests of the previous client request pattern are chosen as a snapshot. The first request is for data item e. While the client scans the broadcast disk, it checks whether the currently broadcast item is going to be requested in the future. This prediction is done by using the rules obtained

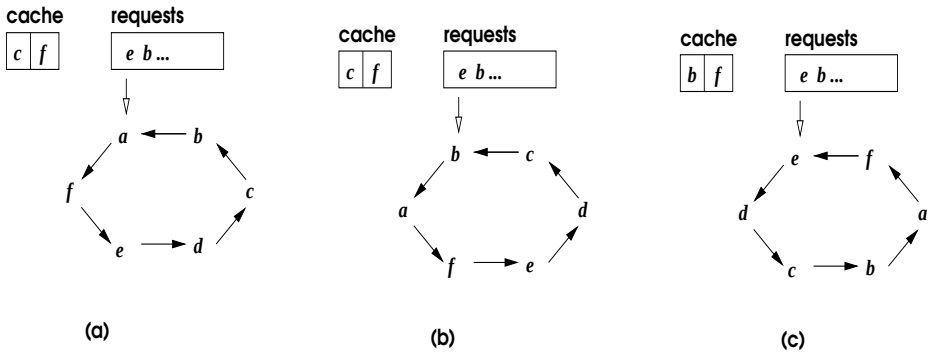

Figure 6.2: Effect of prefetching

from the history of previous requests. The current request is e, and there is a rule stating that if data item e is requested, then data item b will also be requested in the near future. Therefore, the prefetching decides to replace an item from the cache with the currently broadcast data item b. This way the client sweeps the broadcast disk when searching for an item, prefetching the items on the way, that may be accessed in the future.

These simple examples show that with some intelligent grouping and reorganization of data items, and with predictive prefetching, average latency for clients can be considerably improved. In the following sections, we describe how we can extract the sequential patterns out of the client requests. We also explain how we group data items with respect to the sequential patterns and how we can sort the data items in the same group taking into account the ordering imposed by sequential patterns. We also discuss rule based prefetching and cache replacement strategies.

6.2 Implementation of Sequential Rules as Active Rules

The sequential rules obtained as a result of mining the broadcast history can be implemented as active rules. Active rules are used to declaratively express system

responses and they are usually called Event-Condition-Action (ECA) rules as explained in Section 2.4.1. We can implement sequential rules as event-action (EA) rules by dropping the condition (C) part. EA rules have an event and an action. The conditions of EA rules are assumed to be true by default. A sequential rule of the form $S_1 \Rightarrow S_2$ can be defined as an EA rule with event S_1 and action S_2. This way we can exploit features of active systems in broadcast environments. Assuming that we have a set of n data items $D = \{d_1, d_2, ..., d_n\}$, our set of primitive events E has n elements as well where each event corresponds to the arrival of a client request. The primitive event set is defined as $E = \{e_1, e_2, ..., e_n\}$ where e_i denotes the *arrival of a request for* d_i. The composite events which are constructed by using primitive events can be described by using the sequential rules. For a sequential rule $S_1 \Rightarrow S_2$, the head of the rule (i.e., S_1) can be specified as a composite event to be detected. Actions of the rules just affect the new broadcast schedule by inserting the items that are specified at the tail of the rule (i.e., S_2) to the current schedule. For example, for a rule $S_1 \Rightarrow S_2$ where $S_1 =< d_3, d_2, d_5 >$ and $S_2 =< d_6, d_9, d_7 >$, if the primitive events for the arrival of requests for data items d_3, d_2, and d_5 are e_3, e_2, and e_5, respectively, then the composite event for the rule is $e_c = e_3 \rightarrow e_2 \rightarrow e_5$ specifying the order of event occurrence. The action of the rule can be specified as "append data items, d_6, d_9, d_7 into the *new_broadcast_schedule*" where *new_broadcast_schedule* is the schedule constructed with the help of the sequential rules.

When we implement the sequential rules as EA rules, the system automatically detects the rules whose events are signaled by the arrival of requests, and then updates the broadcast schedule accordingly. However, the issue of cascaded rule triggering should be carefully examined since cascaded rule firing may cause cycles of triggered rules. Termination of a set of ECA rules was studied in [AWH92]. The cascaded rule firing and the termination problem are beyond the scope of our work.

6.3 Broadcast Organization Using Sequential Patterns

Our main model for organizing the data items in a broadcast disk is sequential patterns. Sequential patterns are used to cluster the items that are accessed together frequently, and also to impose an ordering on the data items inside a cluster.

6.3.1 Broadcast Organization By Clustering Data Items

Clustering in the context of data mining deals with grouping similar data items together such that the similarity among different clusters is minimized [HKKM97]. An evaluation of different clustering methods for grouping objects was studied by Bouguettaya in [Bou96]. In the context of broadcast environments, clustering can be used to group the data items that are similar to each other. Similarity of data items is defined by the client access patterns. Items that are frequently accessed together are considered to be similar. We cluster data items based on the sequential patterns obtained after mining. In other words, we infer items that are going to be accessed in the near future based on the rules obtained from sequential patterns. Clustering based on sequential patterns is therefore a natural method to use. Han et al. described a clustering method in the context of transactional data based on association rule hypergraph in [HKKM97]. A hypergraph is an extension of the standard graph model where a hyperedge connects two or more vertices, whereas in standard graphs, an edge connects exactly two vertices. The hypergraph model perfectly fits to model sequential patterns in that the items in sequential patterns correspond to the vertices and the sequential patterns themselves correspond to the hyperedges connecting those vertices. The notion of similarity in [HKKM97] is defined by the associations among data items. They use a hypergraph model to cluster both the transactions and the data items. However, their methods are generic and not intended for a specific application like ours. Similar to the clustering method of Han et al., our method for clustering employs a hypergraph as the data model. However, due to the

Sequence	Support(%)	Rules	Confidence(%)	Weight
$< a, b, c >$	2.0	$< a > \Rightarrow < b, c >$	60	220
		$< a, b > \Rightarrow < c >$	80	
$< b, c, a >$	1.0	$< b > \Rightarrow < c, a >$	50	160
		$< b, a > \Rightarrow < a >$	60	
$< b, c, d >$	2.5	$< b > \Rightarrow < c, d >$	70	260
		$< b, c > \Rightarrow < d >$	90	
$< c, e, f >$	2.0	$< c > \Rightarrow < e, f >$	60	210
		$< c, e > \Rightarrow < f >$	70	
$< e, f, h >$	1.5	$< e > \Rightarrow < f, h >$	80	230
		$< e, f > \Rightarrow < h >$	90	
$< f, g, h >$	1.0	$< f > \Rightarrow < g, h >$	60	170
		$< f, g > \Rightarrow < h >$	70	

Table 6.3: Sample sequential patterns

nature of broadcast disk environment where the sequence of data items is important, we use sequential patterns to describe similarities among data items rather than associations. We believe that, in the case of broadcast, sequential patterns give more information than pure associations. The strength of similarity is determined by the support of the sequential pattern and confidence of the rules obtained by the sequential pattern.

The problem we deal with is to partition the set of data items at hand into k subsets such that the resulting subsets are balanced, the total similarities of data items in each subset are maximized, and the total similarities among the data items belonging to different subsets are minimized. This problem is similar to the *graph partitioning problem* which was explained in Section 5.1.1.

A hypergraph $H = (V, E)$ is defined as a set of vertices V and a set of hyperedges E (also called nets). Every hyperedge h_i is a subset of vertices. $P = \{V_1, V_2, \ldots, V_K\}$ is a K-way partition of $H = (V, E)$ if and only if the following three conditions are satisfied:

- $V_k \subset V$ and $V_k \neq \emptyset$ for $1 \leq k \leq K$

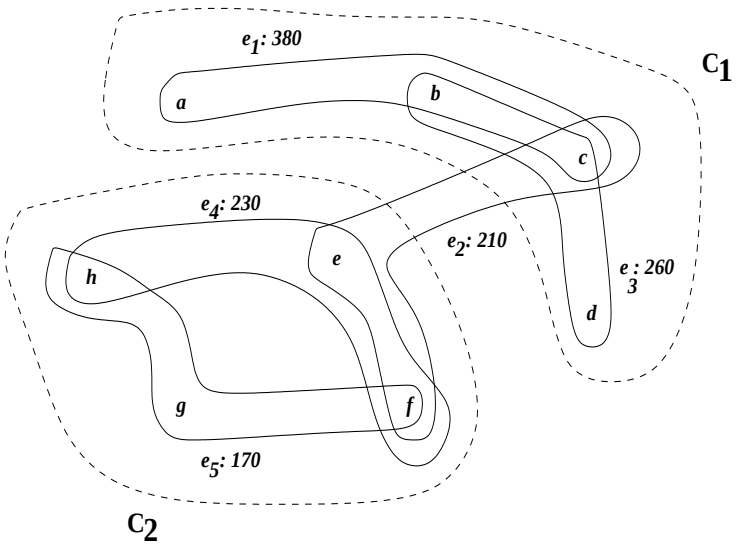

Figure 6.3: The Hypergraph structure for sequential rules

- $\bigcup_{k=1}^{K} V_k = V$
- $V_k \cap V_l = \emptyset$ for $1 \leq k < l \leq K$

The partitioning is sometimes referred to as bisection in the case of two-way partitioning. For $K > 2$, the partitioning is called multi-way or multiple-way partitioning by some researchers.

In our model, vertex set of the hypergraph consists of the data items to be broadcast and hyperedges correspond to sequential patterns. Determination of edge weights is a crucial point in clustering. We define edge weights based on both the support of the corresponding sequential pattern and the confidence of the rules obtained from that sequential pattern. The confidence of the rules is comparable to numeric value 100, however the supports are usually close to 1. To ensure a balance between the confidence and support in weight calculation, we scale the support values to the range 0..100 by dividing the support to the maximum support value of the sequences and then multiplying the result by 100. Sample sequences and the rules obtained out of those sequences are provided in Table 6.3 together with the weights. For the sequential pattern $< b, c, d >$ in the table, the possible sequential rules are $< b > \Rightarrow < c, d >$ and $< b, c > \Rightarrow < d >$

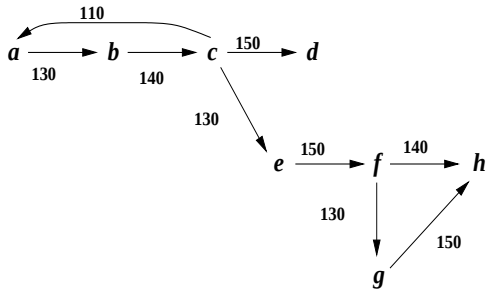

Figure 6.4: Weighted graph of binary sequential rules

with confidence 70, and 90, respectively. Support of $< b, c, d >$ is 2.5%. Since the maximum support among the sequences is also 2.5%, contribution of the support of $< b, c, d >$ in the weight calculation of the hyperedge is $\frac{2.5}{2.5} \times 100 = 100$. The weight is calculated as, $100+70+90 = 260$. Each hyperedge corresponds to one or more sequential patterns. The weight of a hyperedge is the sum of the support of the corresponding sequential patterns and the confidence of the rules obtained by that sequential patterns. The weight for this hyperedge is calculated by summing the weights of the corresponding sequential patterns, and it is $220 + 160 = 380$. The hypergraph corresponding to the sequential patterns given in Table 6.3 is provided in Figure 6.3. Hyperedge e_1 which connects three items, a, b, and c corresponds to two sequential patterns, i.e., $< a, b, c >$, and $< b, c, a >$.

6.3.2 Weighted Topological Sorting of Items Inside a Cluster

As we have discussed above, clustering of data items based on sequential patterns produces sets of data items that are accessed together frequently. Besides the clustering, the ordering of the data items inside a cluster is also important. Therefore, we would like to order the data items inside a cluster such that they conform to the order imposed by sequential patterns.

In order to perform the ordering inside a cluster we consider the set of sequential patterns of size two which form a directed graph structure. This graph may

contain *cycles*, and needs not to be *connected*. For a given set S of sequential patterns over a set I of items, I constitutes the vertices of the graph. There is an edge $i \rightarrow j$ if and only if there is a sequence $(i, j) \in S$.

The vertices of an acyclic directed graph G can be sorted by labeling the vertices with the integers from the set $\{1, 2, 3, ..., n\}$, where n is the number of vertices in G. If there is an edge from vertex i to vertex j in G, then the label assigned to i is smaller than that of j. Ordering the vertices in this manner is called the topological sorting of G [TS92].

The graph of sequential patterns may contain cycles, and in order to topologically sort the graph, we need to eliminate these cycles. Cycle elimination should be done in such a way that the edges removed have minimum impact on the overall ordering of the items. Therefore, we break a cycle by removing the edge with the minimum weight in the cycle. As described before, weights of the edges are determined by the support of the sequential pattern, and the confidence of the rule obtained from that pattern.

The graph structure for the sequential patterns given in Table 6.3 is provided in Figure 6.4. The cycle $a \rightarrow b \rightarrow c \rightarrow a$ is broken by removing the minimum weight edge which is (c, a). The algorithm used for weighted topological sort is presented in Figure 6.5. Input of the algorithm is a weighted directed graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. Output of the algorithm is the topologically sorted list S of vertices in V. After the removal of the cycle, vertex a which has zero indegree, is appended to the list of topologically sorted vertices. Vertex a is removed from the graph together with the edge (a, b). Similarly, vertex b is chosen next for removal and then vertex c. After the removal of vertex c, we have two vertices which are candidates for removal: d and e. We choose vertex d since the weight of edge (c, d) is higher than the weight of edge (c, e). The result of the topological sort of the graph is $< a, b, c, d, e, f, g, h >$.

Begin
Break the cycles in G using minimum weight heuristic
$S \leftarrow \emptyset$ // S keeps the sorted list of vertices
$\forall v_i \in V$, $weight(v_i) = \sum_{v_j \in V, (v_j, v_i) \in E \wedge i \neq j} weight(v_j, v_i)$
$V' \leftarrow V$ // V' keeps track of the set of vertices to be removed.
$E' \leftarrow E$ // E' keeps track of the set of edges to be removed.
while $V' \neq \emptyset$ **do**
begin
$Z \leftarrow Z \cup \{v_i | v_i \in V' \wedge indegree(v_i) = 0\}$
select $v_i \in Z$ such that $weight(v_i)$ is maximum
append v_i to S
$\forall (v_i, v_j) \in E$ *remove* (v_i, v_j) from E'
remove (v_i) from V'
remove (v_i) from Z
end
Append rest of the vertices in Z to S in decreasing order of weight
End

Figure 6.5: Weighted Topological Sorting Algorithm

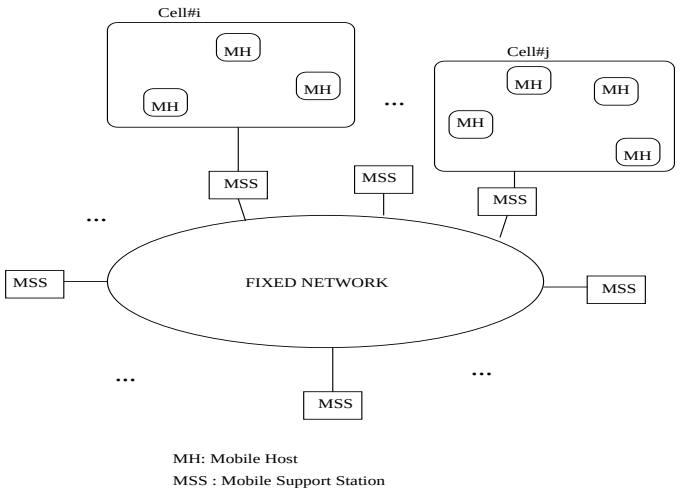

Figure 6.6: A typical architecture for a mobile computing system

6.4 Utilization of Sequential Rules in Prefetching and Cache Replacement

A typical architecture for mobile computing systems inspired from [IB94] is depicted in Figure 6.6. In this architecture, there is a fixed network of *Mobile Support Stations* (MSSs). *Mobile Hosts* (MHs) are the computers which are portable and capable of wireless communication. Each MH is associated with an MSS, and MHs are connected to MSSs via wireless links. An MSS is generally fixed and it provides MHs with a wireless interface inside a prespecified area called a *cell*.

We assume that there is a fixed set of data items that is periodically broadcast to mobile clients. Mobile clients issue requests for data items when the required items are not in the current broadcast set. The data server responds to client requests by placing the requested items on the broadcast disk. The server should carefully decide what portion of the broadcast disk will be dedicated to requests and what portion will be used for the fixed set of broadcast items. The portion dedicated to client requests is called pull bandwidth, and the portion dedicated to the frequently requested items is called the push bandwidth [AFZ97]. We are particularly interested in the organization of the push bandwidth and prefetching of the data items from the broadcast disk.

```
Begin
    inferred_items = ∅
    for every sequential rule S1 ⇒ S2 do
        if there is a match for S1 in the current n requests then
            inferred_items = inferred_items ∪ S2
        endif
    endfor
End
```

Figure 6.7: Construction of the set $inferred\_items$

The following subsections discuss the automated use of sequential rules for prefetching and cache replacement in broadcast environments.

6.4.1 Rule-based Prefetching and Cache Replacement Strategies

Both prefetching and cache replacement strategies exploit the sequential rules obtained by mining the broadcast history. The set of data items inferred by the sequential rules (that we denote by $inferred\_items$) is used as a base for rule based prefetching and caching strategies. The set $inferred\_items$ is constructed by using the algorithm given in Figure 6.7. The prefetching algorithm looks at the current n requests to predict what will be requested next. The predicted items are stored in the set of $inferred\_items$. In case the number of rules is large, we limit the number of inferred items by sorting them in decreasing order of priority and selecting the items with relatively high priority. The priority of each inferred item is determined by the confidence of the rule that has inferred it.

In order to perform rule-based prefetching and cache replacement, the mobile clients should be aware of the sequential rules in the current cell. This can be provided by ensuring that the rules also appear in the broadcast set; i.e., they are

broadcast periodically. Broadcasting of rules together with the data items induces an overhead on the broadcast disk since the size of broadcast disk increases with the addition of sequential rules. However, the number of the rules in the system is usually much smaller than the number of broadcast items, therefore the induced overhead turns out to be negligible. When a mobile client enters a cell, it fetches the current rule set in its new cell at the time sequential rules are broadcast. This also induces an overhead on mobile clients which may not be significant if the client does not move from one cell to another frequently. However, in case the cell sizes are small and the mobile client changes its cell frequently, loading and setting up the current rule set may take a considerable amount of time. This problem can be overcome by examining the profiles and mobility patterns of the users. The client on a mobile computer decides to load the current rule set if the user of the computer who enters a new cell will stay in this cell for a sufficiently long period of time. This information can be explicitly obtained from the user. A better approach could be to use user profiles if they are available. User profiles can contain information like [IB92]:

- Probabilities of relocation between any two locations within a location server base (MSS).
- Average number of calls per unit time.
- Average number of moves per unit time.

The problem of prefetching broadcast items is discussed in [AFZ96]. Prefetching is useful when the data items to be used subsequently are already on the air, ready to be retrieved. We suggest that sequential rules generated from broadcast histories can also be used in prefetching broadcast items.

The rule set loaded by the clients is utilized for prefetching data items in the broadcast set. Clients and the server symmetrically and synchronously utilize the sequential rules. The server uses the rules to organize the broadcast requests and clients use the rules to do prefetching. Clients consider current n requests they have issued and do prefetching using the rules broadcast by the server. Prefetching, used with a rule-based cache replacement strategy can decrease the

```
Begin
    if bcast_item is not in cache then
        if bcast_item is in inferred_items − cached_items then
            pvalue = rule_conf(bcast_item)
        if minimum pvalue of cached_items > pvalue then
            perform prefetching
    endif
End
```

Figure 6.8: Prefetching algorithm

waiting times of the mobile clients for the arrival of required broadcast items when those items are available on the broadcast channel. This is very intuitive since fetching a data item beforehand when it is available means that the client does not have to wait for the data to appear on the broadcast when it is actually required.

We have compared our approach with the prefetching method called PT that takes the broadcast environment into account. This method was proposed by Acharya et al. in [AFZ96]. PT considers both the access probabilities of a data page and the time that the page will be broadcast again for prefetching. Each page is associated with a value, called the *pt* value, which is used for prefetching and cache replacement decisions. PT computes the *pt* value of a page by multiplying the probability of access (P) of the page with the time (T) that will elapse for the page to reappear on the broadcast. Broadcast page is replaced with the page in the cache which has the minimum *pt* value if the minimum *pt* value is less than the *pt* value of the broadcast page.

The algorithm we use for prefetching is presented in Figure 6.8. Similar to the PT method, a value is associated with each data item to be used for prefetching, which is called the *pvalue* of the item. Prefetching of a data item is performed for data items that are not cache resident. Prefetching is performed if the *pvalue* of the item on broadcast is greater than the minimum *pvalue* of the cache resident

Begin
  if $cached\_items - inferred\_items = \emptyset$ **then**
    for each data item in $inferred\_items \cap cached\_item$ **do**
      replace the data item supported by a rule with the least *confidence*
  else
    replace a data item in $cached\_items - inferred\_items$ using their *pvalues*
  endif
End

Figure 6.9: Cache Replacement algorithm

data items. The *pvalue* of a data item corresponds to the confidence of the rule that inferred that item (denoted by $rule\_conf(bcast\_item)$) if the item is in the list of $inferred\_items$.

Cache replacement is also an important issue for mobile clients considering the limitations on wireless bandwidth and cache capacity of mobile computers. Clients can use the sequential rules broadcast by the servers (or MSSs) in determining which items should be replaced in their cache. Clients consider a window of current n requests in order to determine the next items that might be needed by the user and do the cache replacement according to their rule-based predictions. The rule-based cache replacement algorithm we propose is provided in Figure 6.9. The algorithm first determines the data items that will probably be requested in the near future by using both the sequential rules and the last n requests issued by the client, as shown in Figure 6.7. These data items are accumulated in the set $inferred\_items$. Any cache replacement strategy can be used on the set $cached\_items - inferred\_items$, i.e., the difference between the set of data items currently residing in the cache and the data items inferred by the sequential rules. Another possible approach is to use the set of inferred items for determining the replacement probabilities of cached items, instead of completely isolating them from the set of items to be considered for cache replacement. Both of the approaches can be classified as hybrid since they are taking the advantages of both conventional and rule-based cache replacement strategies.

There might be a special case where the size of the set of items inferred by the rules as candidates to be requested in the near future may exceed the cache size. There are two possible approaches that might be adopted for cache replacement to handle such a case:

- Consider the inferred items for replacement in the order of timestamps of requests.
- Consider the inferred items for replacement in the order of confidences of the rules that have inferred them.

With the first approach, the temporal order of the requests is considered, and the items inferred as a result of requests which have been issued earlier are preferred to be kept in the cache. The second approach prioritizes the rules according to their confidences; i.e., among any two data items, the one inferred by the rule with a higher confidence has higher priority than the other item. The item with the lowest priority is replaced first.

6.5 Simulation and Experimental Results

We implemented the data mining algorithms, and the rule based scheduling and prefetching mechanisms to show the effectiveness of the proposed methods. A Web log was used for simulating the broadcast history. We think that requests for Web pages are actually a good approximation to the list of past requests by mobile clients. The simulation model we used and the experimental results are provided in Section 6.5.2 and Section 6.5.3, respectively.

6.5.1 The Training and Test Data

We used the anonymous Web data from www.microsoft.com created by J. S. Breese, D. Heckerman, and C. M. Kadie from Microsoft. Data was created by

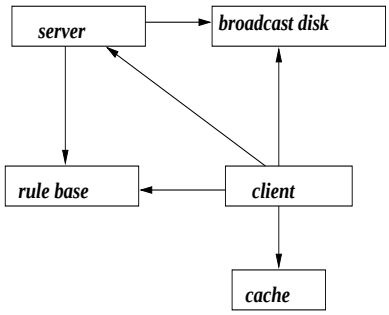

Figure 6.10: Object Relationship Diagram of the Simulation Program

sampling and processing the www.microsoft.com logs and donated to the Machine Learning Data Repository stored at University of California at Irvine Web site [mic]. The Web log data keeps track of the use of Microsoft Web site by 38000 anonymous, randomly-selected users. For each user, the data records list all the areas of the Web sites that the user visited in a one week timeframe. The number of instances is 32711 for training and 5000 for testing. Each instance represents an anonymous, randomly selected user of the Web site. The number of attributes is 294 where each attribute is an area of the www.microsoft.com Web site.

We selected 50 most frequent data items for broadcasting. This is a reasonable method since in broadcast environments only the frequently accessed items are broadcast while the others are requested through the back channel. However, since the set of 50 data items is too small for a broadcast disk, we replicated these 50 data items together with the requests and the rules corresponding to these items, and constructed a disk of 200 data items.

6.5.2 Simulation Model

The whole system consists of three independent parts:

- Rule extraction module,
- Broadcast organization module, and
- Broadcast simulation module.

The rule extraction module performs the task of extracting sequential rules from the Web log. Rule sets with different minimum confidence and support requirements can be constructed by the rule extraction module. The resulting sequential rules are written to a file in a specific format to be read later by the broadcast organization and broadcast simulation modules. The broadcast organization module performs clustering of data items using hypergraph partitioning based on sequential patterns. We used the PATOH hypergraph partitioning tool for our experiments [CA99]. The broadcast organization module also performs weighted topological sort of data items inside clusters. Details of the clustering and topological sorting of data items are provided in Section 6.3. Main objects of the broadcast simulation module and their relationship are depicted in Figure 6.10. The objects are *server, client, cache, broadcast disk*, and *rule base*. Both the *server* and *client* objects have a common *rule base* and a *broadcast disk*. The *client* object interacts with the *server* object by sending broadcast requests to it. The *server* and *client* objects interact with the *broadcast disk* by placing new items to the *broadcast disk* and fetching data items from the *broadcast disk*, respectively. The *rule base* object is constructed by reading the rule file generated by the data mining module and is shared by the *client* and *server* objects. The *rule base* object is contacted by the *client* and *server* objects for prefetching and broadcast scheduling, respectively. The *client* has a *cache* object to store a limited amount of requested items.

The architecture of our system is depicted in Figure 6.11. In this architecture, we use the the microsoft Web logs to simulate the broadcast history. The sequence of Web logs that are organized into sessions is fed into the data mining program to be used for extracting sequential rules. The resulting rule set is fed into the rule base which is then used for broadcast organization, cache replacement, and prefetching. The Web log provided for testing is used for simulating the requests of a client.

As we present in Figure 6.11, our broadcast environment has a server that sends data items in the broadcast disk, and a client that reads data items on the broadcast disk and sends data item requests whenever necessary. Broadcast disk is organized by the server using the sequential patterns. Clients use the sequential

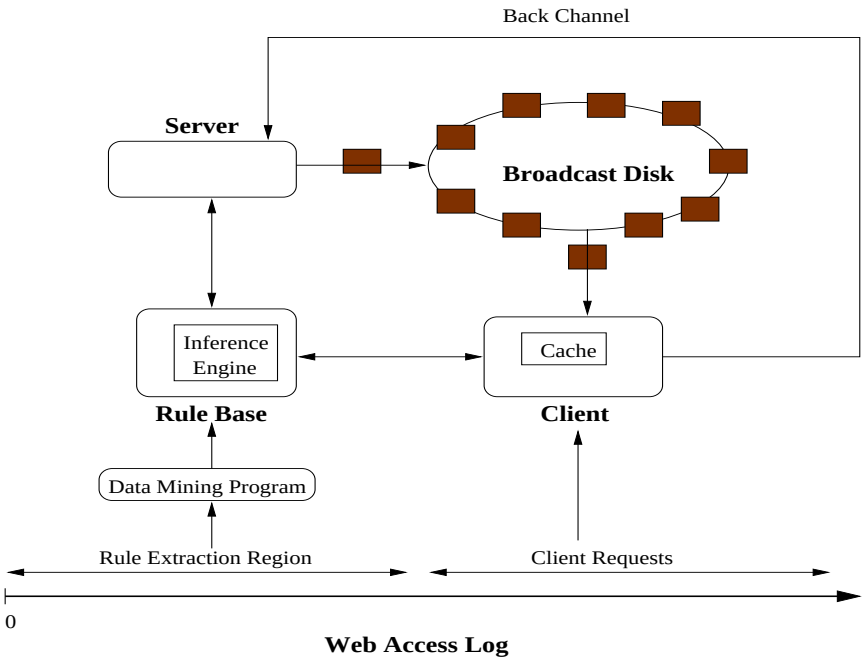

Figure 6.11: System Architecture

rules for prefetching from the broadcast disk.

We assume a simple broadcast structure, as this would be sufficient for constructing an execution environment that would enable us to measure the effectiveness of extracted rules over client requests.

6.5.3 Experimental Results

We implemented the rule mining algorithms on our Web log. We extracted rules with different support values and evaluated their effect on the performance. Our main performance metric is the *average latency*. We also measured the *client cache hit ratio*. A decrease in the average latency is an indication of how useful the proposed methods are. The average latency can decrease as a result of both increased cache hit ratio via prefetching methods, and better data organization in the broadcast disk. An increase in the cache hit ratio will also decrease the number of requests sent to the server on the backlink channel, and thus, lead to both saving of the scarce energy sources of the mobile computers and reduction

Broadcast Size	Maximum size of the broadcast disk
Cache Size	Maximum size of the client cache
Minimum Support	Minimum support value of the extracted rules
Minimum Confidence	Minimum confidence value of the extracted rules
Max Inferred	Maximum number of items that can be inferred by rules
Queue Size	Size of the queue that stores the inferred items

Table 6.4: Main parameters of our system

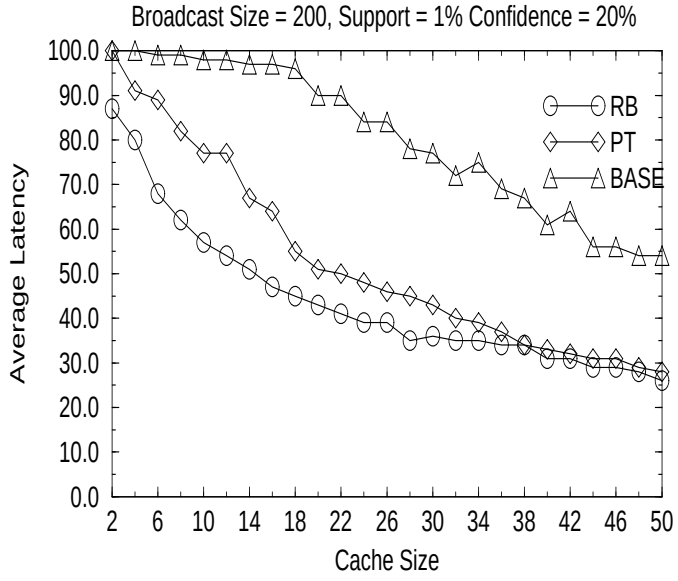

Figure 6.12: Average latency as a function of the cache size

in the server load.

Data mining is performed in main memory. The running time of the data mining algorithm does not exceed a few minutes. Considering that the mining process is not done very frequently, the running time is not significant. The running time of the rule checking algorithm is not significant either since the number of rules is not so large. We observed in our experiments that the optimum number of rules for the best cache hit ratio does not exceed a few hundreds.

The basic parameters of our system are presented in Table 6.4. As the broadcast size does not have a serious impact on the cache hit ratio, we assumed a fixed broadcast size of 200 data items in all our experiments.

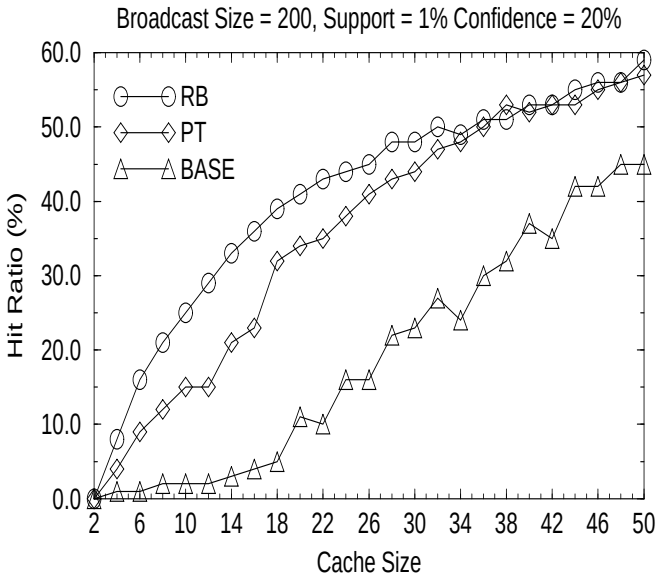

Figure 6.13: Cache hit ratio as a function of the cache size

We performed experiments under varying values of the cluster size, and the best results for the test data were obtained with a cluster size of four. We set the minimum support threshold to 1% and minimum confidence threshold to 20% for the first part of the experiments which aims to evaluate the impact of cache size. We varied the cache size to observe how the average latency is affected. The cache size is measured in terms of the number of items, assuming that the items retrieved are html documents, possibly with images. The average latency is measured in terms of broadcast ticks.

We compared our rule based prefetching method (RB) with the state of the art prefetching algorithm[1] (PT) for broadcast disks. We also evaluated the performance of a base algorithm (BASE) which does not do prefetching, and only performs LRU based cache replacement. PT is a very good heuristic when the locations of the data items in the disk and their relative access frequencies are known. However with an extra knowledge which describes the sequence of data item accesses by clients (i.e., by involving RB) further improvement in the performance is possible, as can be seen in Figure 6.12. The cache hit ratios obtained with the methods RB, PT, and BASE are provided in Figure 6.13. As can be

[1]See Section 6.4.1.

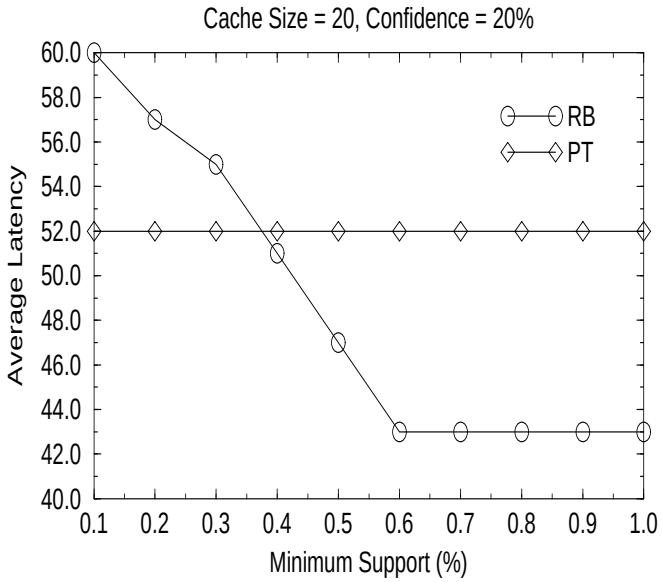

Figure 6.14: Average latency as a function of the minimum support threshold (for small support values)

observed from both figures, all three methods react similarly to the changes in the cache size. As expected, increasing the cache size leads to an increase in cache hit ratio and a decrease in average latency. For any cache size value tested, the cache hit ratio values obtained with RB and PT are much higher compared to that with BASE, which leads to much better performance for RB and PT in terms of the average latency. Comparing RB and PT, it can be observed that except for large cache sizes tested, the cache hit ratio and the corresponding average latency values obtained with RB are consistently better than those with PT. Large cache size values (i.e., cache size > 30) lead to comparable performance results for these two algorithms. RB does not provide an improvement in the performance for large cache size values, because most of the data items inferred by the rules are already stored in the cache. For small cache sizes, the content of the cache is more dynamic and RB is more effective in this case. Performance improvement by RB is achieved via rule based prefetching and cache replacement. Prefetching and cache replacement works hand in hand with RB as well as PT, i.e., these two algorithms prioritize the cache items using the rule weights and pt values, respectively.

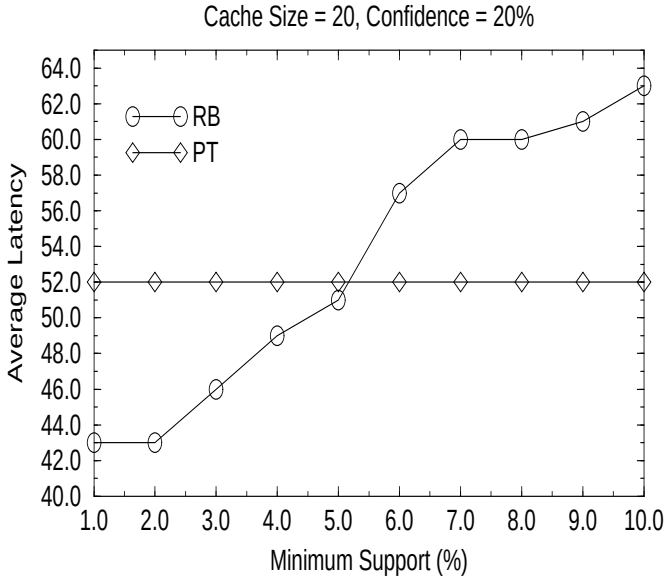

Figure 6.15: Average latency as a function of the minimum support threshold (for large support values)

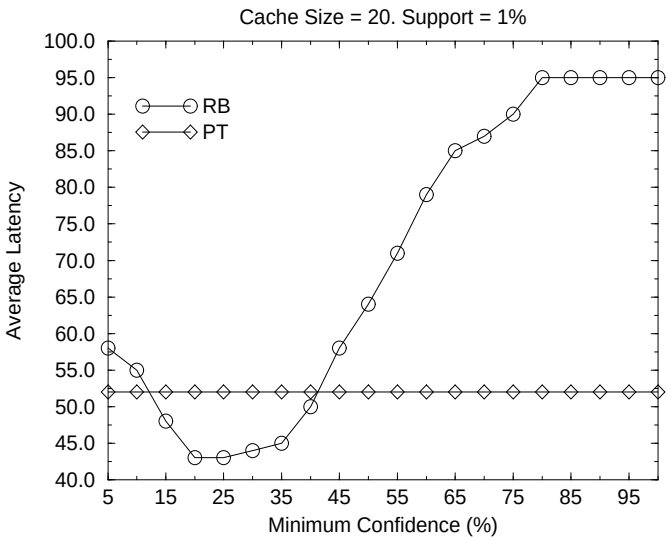

Figure 6.16: Average latency as a function of the minimum confidence threshold

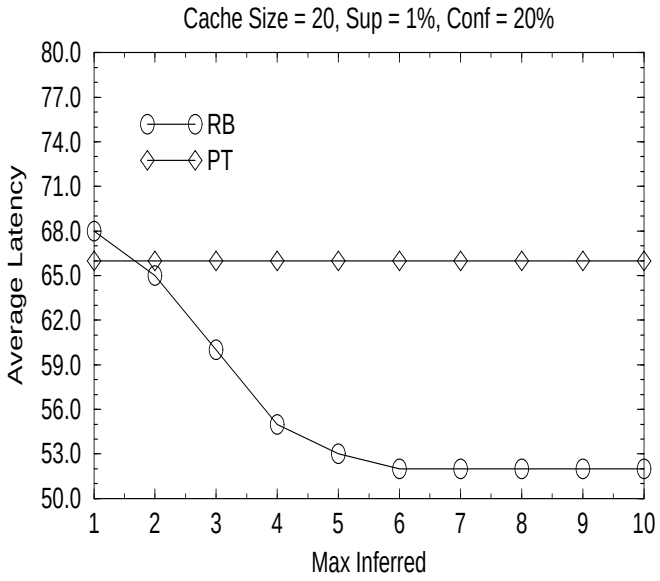

Figure 6.17: Average latency as a function of the maximum number of items that can be inferred by rules

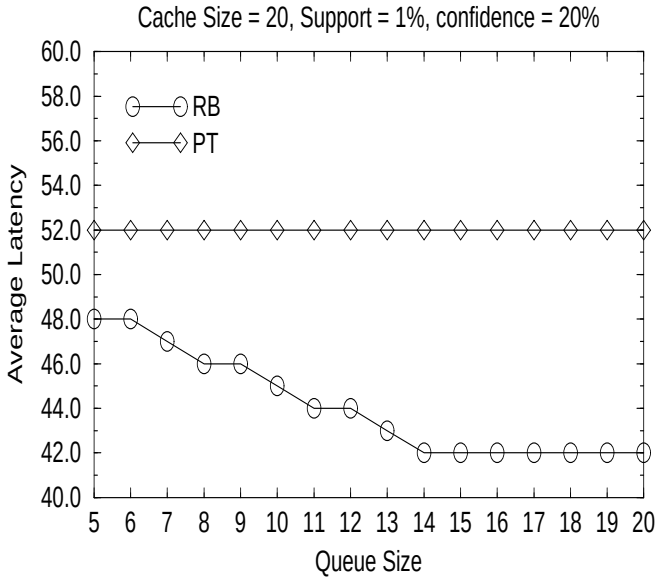

Figure 6.18: Average latency as a function of the size of the queue that stores the inferred items

In order to evaluate the impact of the minimum support and confidence threshold on the performance of the rule based prefetching, we also conducted experiments by varying these two parameters. The results obtained in these experiments are provided in Figures 6.14, 6.15, and 6.16. We conducted two separate experiments in investigating the performance impact of different support values; one for small support values (in the range 0.1-1.0 in steps of 0.1), and one for large support values (in the range 1.0-10.0, in steps of 1.0). The performance results obtained for these two sets of support values are provided in Figures 6.14 and 6.15, respectively. As the support is increased for small support values, the average latency decreases since the rules chosen become more effective. The best results are obtained for the minimum support values of 0.6 through 2.0. When the support is increased further, the average latency starts to increase since the number of rules decreases. The performance impact of the confidence was also examined by varying its value from 5 to 100 in steps of 5. The results are similar to those obtained with the minimum support experiments, as shown in Figure 6.16. For small confidence values it is possible to improve the performance by increasing the confidence threshold. However, after a certain confidence threshold (i.e., 25% in this experiment), increasing the confidence leads to an increase in the average latency since less number of rules, and thus less number of inferred items, are involved for large confidence values.

We also conducted experiments to investigate the performance impact of the maximum number of items inferred at each step (i.e., *max inferred*) and size of the queue that stores the inferred items (i.e., *queue size*). The results of these two experiments are displayed in Figures 6.17 and 6.18, respectively. As the maximum number of inferred items is increased, the average latency decreases up to a certain point (i.e., *max inferred* = 5 in this experiment). Further increase in the maximum number of inferred items does not improve the performance, because the number of items inferred can not be increased any more. A similar performance pattern is also observed by varying the size of the queue that stores the inferred items (Figure 6.18). Some of the items stored in the queue may become obsolete after a while, and they are pushed out of the queue as new items are inferred. For large queue sizes, increasing the queue size does not lead to

better performance, because this would just increase the fraction of obsolete items stored in the queue, and such items do not have any effect on the performance.

6.6 Summary

In this chapter, we have discussed some data mining techniques to be used for broadcasting data in mobile computing environments. We have proposed a new method of organizing the data broadcast in mobile environments using the sequential rules obtained by mining the broadcast history. The sequential rules are used as a base for clustering data items and thus for data organization in the broadcast disk. We have also shown that sequential rules are beneficial for prefetching of data items by mobile clients. A cache replacement mechanism for mobile computers has been proposed which utilizes the sequential rules to determine the items to be replaced in the cache. The proposed methods have been evaluated through performance experiments on a Web log. The rules resulted from mining a Web log have been used to test the effectiveness of the proposed methods.

Chapter 7

Conclusions and Future Work

In this thesis, we have explored data mining techniques for mining both single and multiple correlated event histories to obtain event patterns. We have shown how a single event history can be mined via a sliding window algorithm to extract similarities among events in terms of the proximity of event occurrence. We have also proposed methods for mining broadcast histories that store client data request events to obtain sequential patterns of data requests.

We have introduced a new data mining problem for mining associations among events spanning different correlated event histories. These types of event patterns are called *cross associations*. We have presented two algorithms (concurrent and merge) for extracting cross associations from two correlated event histories. The performance of the proposed algorithms has been evaluated on synthetic data sets. We have conducted experiments on different support and confidence values. It has been proven through the experiments that processing two correlated histories separately in a concurrent manner is a better approach compared to merging and mining the histories. This result is due to the fact that considering the two histories separately allows the use of confidence and support pruning strategies on both cross associations and frequent event sets. We have also conducted experiments on the real life weather and power consumption data in the context of the CIMEG project [CIM].

We have demonstrated that the event patterns obtained by data mining techniques from both single and correlated event histories have many implications in active and mobile data management:

- From the active data management perspective, the event patterns are used to divide the event set into smaller related groups of events which we call fuzzy event sets. We have proposed methods for automated construction of fuzzy event sets for unsupervised event organization and rule modularization. Fuzzy event sets are constructed via standard graph partitioning methods by modeling the similarities among the events as a graph structure. Event similarities are further used for similarity based event detection in active databases. With fuzzy events and fuzzy event sets, a priority assignment mechanism for sequential rule execution can be utilized based on membership functions of fuzzy events to fuzzy event sets. Organization of rules into groups of related rules, which are further mapped to fuzzy event sets, increases the efficiency and modularity of rule execution. The event patterns and the groups of events are used for similarity based event detection and fuzzy rule execution in active database systems. As another extension to active database systems, we have shown that the event patterns can also be used for predictive event detection and proactive rule execution. Predictive event detection and proactive rule execution in active database systems is a novel idea that aims to improve the system performance especially for real-time applications where detecting critical events before they actually happen, and executing actions proactively are crucial. We have provided a framework for predictive event detection and proactive rule execution where the event detection mechanism of existing active database systems can be seamlessly used to support these new features.

- From the mobile data management perspective, we have shown that in data broadcast environments, the broadcast data can be organized into groups of related data items as in the case of the organization of events into fuzzy event sets. We have shown that client request event patterns can be extracted from the broadcast history via data mining techniques.

The event patterns which are expressed as sequential patterns of client request events can be used to group the data items requested close to each other in time. Our novel method of organizing the data broadcast in mobile environments uses the sequential client access event patterns obtained by mining the broadcast history. With this approach, we have achieved a considerable decrease in the delay experienced by the clients while waiting for the required data items. This result is due to the fact that the data items that are requested together in a time frame are placed close together in the broadcast. We have also shown that sequential rules obtained from the sequential patterns of client request events are beneficial for proactive loading (i.e., prefetching) of the data items to the cache of mobile clients. A cache replacement mechanism for mobile computers has been proposed which utilizes the event patterns to determine the items to be replaced in the cache. The proposed methods for prediction and prefetching have been evaluated through performance experiments on a Web log. The rules resulting from mining a Web log have been used to test the effectiveness of the proposed methods. The performance of our methods has been compared with a state of the art prefetching technique, as well as a base algorithm. It has been observed through performance experiments that a considerable increase in the cache hit ratio can be obtained when the rule based broadcast organization, cache replacement and prefetching techniques are used together. The increase in the cache hit ratio leads to lower average latency especially for small cache sizes which is typical for mobile devices.

In our work on broadcast data management, we have not dealt with the temporality issues. However, temporal information can also be exploited for scheduling broadcast requests. Temporal sequential rules can be obtained by mining the broadcast history considering the relative times of the broadcast requests as well. The issue of mining temporal patterns is discussed in [BWJL98]. Temporal sequential rules can improve the effectiveness of rule-based scheduling by considering the time of the requests as well as the sequence of requests. The issue of utilization of temporal sequential rules in broadcasting is left as a future work.

Our work on active data management can also be extended in several ways. In

some active database system applications there might be more than two correlated event histories, and pairwise correlations among these event histories might be more complex compared to the case of two correlated event histories. Consider an application that deals with the stock exchange information in different stock exchange markets. An abrupt change in one of those markets, say NASDAQ can have a ripple effect on the other stock exchange markets. In this case the relationship among multiple correlated event histories is a star schema. This can be a complex relationship and it may not be obvious which events trigger which other events. As a future work, our algorithms will be extended to a collection of histories of size larger than two. Sometimes the events in two histories may be related to each other with a phase difference. For example, the temperature change may affect the power consumption not in the same hour it happens but in the next hour. Mining cross associations with a phase difference will also be considered as a future work.

Bibliography

[AAFZ95] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: Data management for asymmetric communication environments. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, San Jose, California, June 1995.

[AFZ96] S. Acharya, M. Franklin, and S. Zdonik. Prefetching from a broadcast disk. In *Proceedings of the 12th International Conference on Data Engineering(ICDE'96)*, New Orleans, LA, February 1996.

[AFZ97] S. Acharya, M. Franklin, and S. Zdonik. Balancing push and pull fro data broadcast. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, Tucson, Arizona, May 1997.

[AGI+92] R. Agrawal, S. Ghosh, T. Imielinski, B. Iyer, and A. Swami. An interval classifier for database mining applications. In *Proceedings of the 18th Int'l Conference on Very Large Databases*, pages 560–573, Vancouver, August 1992.

[AS94] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th Int'l Conference on Very Large Databases*, Santiago, Chile, September 1994.

[AS95] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. of the Int'l Conference on Data Engineering (ICDE)*, Taipei, Taiwan, March 1995.

[AWH92] A. Aiken, J. Widom, and J. M. Hellerstein. Behavior of database production rules: Termination, confluence, and observable determinism. In *Proceedings of ACM-SIGMOD Conference on Management of Data*, pages 59–68, 1992.

[BCP96] E. Baralis, S. Ceri, and S. Paraboschi. Modularization techniques for active rules design. *ACM Transactions on Database Systems*, 21(1), 1996.

[BJ92] T.N. Bui and C. Jones. Finding good approximate vertex and edge partitions is np-hard. *Information Processing Letters*, 42(3), 1992.

[BKPW97] T. Bouaziz, J. Karvonen, A. Pesonen, and A. Wolski. Design and implementation of tempo fuzzy triggers. In *Lecture Notes in Computer Science*, volume 1308, pages 91–100, Toulouse, Fran, September 1997. Springer.

[BMUT97] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1997.

[Bou96] A. Bouguettaya. On-line clustering. *IEEE Transactions on Knowledge and Data Engineering*, 8(2), 1996.

[BP82] B. Buckles and F. Petry. A fuzzy model for relational databases. *International Journal of Fuzzy Sets and Systems*, 7:213–226, 1982.

[BP83] B. P. Buckles and F. E. Petry. Information-theoretic characterization of fuzzy relational databases. *IEEE Trans. on Systems, Man and Cybernetics*, 13(1):74–77, February 1983.

[BW96] T. Bouaziz and A. Wolski. *Incorporating Fuzzy Inference into Database Triggers*. Technical report, VTT Information Technology, November 1996.

[BW97] T. Bouaziz and A. Wolski. Applying fuzzy events to approximate reasoning in active databases. In *Proceedings of the 6th IEEE International Conference on Fuzzy Systems(FUZZ-IEEE'97)*, Barcelona, Catalonia, Spain, July 1997.

[BWJL98] C. Bettini, X. S. Wang, S. Jajodia, and J-L. Lin. Discovering frequent event patterns with multiple granularities in time sequences. *IEEE Transactions on Knowledge and Data Engineering*, 10(2), 1998.

[CA99] U.V. Catalyurek and C. Aykanat. Hypergraph-partitioning based decompostion for parallel sparse-matrix vector multiplication. *IEEE Transactions on Parallel and Distributed Systems*, 10:673–693, 1999.

[CHNW96] D. W-L. Cheung, J. Han, V. Ng, and C. Y. Wong. Maintenance of discovered association rules in large databases: An incremental updating technique. In *Proceedings of the 12th International Conference on Data Engineering (ICDE)*, pages 106–114, 1996.

[CIM] Consortium for the Intelligent Management of the Electric Power Grid (CIMEG). http://helios.ecn.purdue.edu/~cimeg.

[CNFF96] D. W-L. Cheung, V. Ng, A. W-C. Fu, and Y. Fu. Efficient mining of association rules in distributed database. *IEEE Transactions on Knowledge and Data Engineering*, 8(6), 1996.

[Day88] U. Dayal. Active database management systems. In *Proceedings of the Third International Conference on Data and Knowledge Bases*, pages 150–169, Jerusalem, June 1988.

[GSPB96] R. George, R. Srikanth, F. E. Petry, and B. P. Buckles. Uncertainty management issues in object-oriented database systems. *IEEE Trans. on Fuzzy Systems*, 4(2):179–192, May 1996.

[HF99] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. *IEEE Transactions on Knowledge and Data Engineering*, 11(5), 1999.

[Hid99] C. Hidber. Online association rule mining. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1999.

[HKKM97] E-H. Han, G. Karypis, V. Kumar, and B. Mobasher. Clustering based on association rule hypergraphs. In *Proccedings of SIGMOD'97 Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD97)*, 1997.

[HPY00] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proc. of ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD'2000)*, Dallas, TX, May 2000.

[HW92] E. N. Hanson and J. Widom. *An Overview of Production Rules in Database Systems.* Technical report, University of Florida, Department of Computer and Information Sciences, October 1992.

[IB92] T. Imielski and B. R. Badrinath. Querying in highly distributed environments. In *Proceedings of the 18th VLDB Conference*, Columbia, Canada, 1992.

[IB94] T. Imielinski and B. R. Badrinath. Mobile wireless computing: Challenges in data management. *Communications of the ACM*, pages 19–27, October 1994.

[IVB97] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Data on air: Organization and access. *IEEE Transactions on Knowledge and Data Engineering*, 9(3), 1997.

[KCY97] G. J. Klir, U. H. St. Clair, and B. Yuan. *Fuzzy Set Theory Foundations and Applications.* Prentice Hall PTR, 1997.

[KF88] G. J. Klir and T. A. Folger. *Fuzzy Sets, Uncertainty and Information.* Prentice Hall, 1988.

[KK] G. Karypis and V. Kumar. *METIS Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparce Matrices, Version 4.0.* University of

Minnesota, Department of Computer Science/ Army HPC Research Center, Mineappolis, MN 55455.

[KL70] B. W. Kerninghan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2), 1970.

[LPT99] L. Liu, C. Pu, and W. Tang. Continual queries for internet scale event-driven information delivery. *IEEE Transactions on Knowledge and Data Engineering*, 11(4), 1999.

[mic] University of california at irvine machine learning repository, `http://www.ics.uci.edu/~mlearn/mlsummary.html`.

[MTV95] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. In *Proceedings of the 1st International Conference on Knowledge Discovery and Datamining*, Montreal, Canada, August 1995.

[PD99] N. W. Paton and O. Diaz. Active database systems. *ACM Computing Surveys*, 31(1):1–29, 1999.

[Pet96] F. E. Petry. *Fuzzy Databases: Principles and Applications.* Kluwer Press, 1996.

[SA95] R. Srikant and R. Agrawal. Mining generalized association rules. In *Proc. of the 21st Int'l Conference on Very Large Databases*, Zurich, Switzerland, September 1995.

[SA96] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1996.

[SRB97] K. Stathatos, N. Roussopoulos, and J. S. Baras. Adaptive data broadcast in hybrid networks. In *Proceedings of the 23rd VLDB Conference*, Athens, Greece, 1997.

[SU01] Y. Saygın and O. Ulusoy. Automated construction of fuzzy event sets and its application to active databases. *IEEE Transactions on Fuzzy Systems*, 9(3), 2001.

[SUC98] Y. Saygın, O. Ulusoy, and Sharma Chakravarthy. Concurrent rule execution in active databases. *Information Systems*, 23(1), 1998.

[SUY99] Y. Saygın, O. Ulusoy, and A. Yazıcı. Dealing with fuzziness in active mobile databases. *Information Sciences*, 120(1-4), 1999.

[SW98] A. P. Sistla and O. Wolfson. Minimization of communication cost through caching in mobile environments. *IEEE Transactions on Parallel and Distributed Systems*, 9(4), April 1998.

[TS92] K. Thulasiraman and M. N. S. Swamy. *Graphs: Theory and Algorithms*. A Wiley and Sons, Inc., 1992.

[WB98] A. Wolski and T. Bouaziz. Fuzzy triggers: Incorporating imprecise reasoning into active databases. In *Proceedings of the 14th Int'l Conference on Data Engineering(ICDE'98)*, Orlando, Florida, February 1998.

[WHH00] K. Wang, Y. He, and J. Han. Mining frequent itemsets using support constraints. In *Proc. 2000 Int. Conf. on on Very Large Data Bases (VLDB'00),*, Cairo, Egypt, 2000.

[YBP99] A. Yazıcı, B. P. Buckles, and F. E. Petry. Handling Complex and Uncertain Information in the ExIFO and NF^2 Data Models. *IEEE Trans. on Fuzzy Systems*, 7(6):659–676, 1999.

[YG99] A. Yazıcı and R. George. *Fuzzy Database Modeling*. Springer-Verlag, Heidelberg, 1999.

[YSBP99] A. Yazıcı, A. Soysal, B. P. Buckles, and F. E. Petry. Uncertainty in a nested relational database model. *Data and Knowledge Engineering*, 30(3):275–301, 1999.

[YSJ+00] B-K. Yi, N.D. Sidiropoulos, T. Johnson, H.V. Jagadish, C. Faloutsos, and A. Biliris. Online data mining for co-evolving time sequences. In *Proc. of the Sixteenth International Conference on Data Engineering (ICDE'00)*, San Diego, 2000.

[Zad65] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[ZFAA94] S. Zdonik, M. Franklin, R. Alonso, and S. Acharya. Are "Disks in the Air" just pie in the sky. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, December 1994.