SIGNIFICANT INTERVAL AND EPISODE DISCOVERY

IN TIME-SERIES DATA

The members of the Committee approve the master's
thesis of Ambika Srinivasan

Sharma Chakravarthy
Supervising Professor _____

Leonidas Fegaras _____

Manfred Huber _____

SIGNIFICANT INTERVAL AND EPISODE DISCOVERY IN TIME-SERIES DATA

by

AMBIKA SRINIVASAN

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

DECEMBER 2003

TO MY HUSBAND, PARENTS, FAMILY AND FRIENDS

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Dr. Sharma Chakravarthy, for giving me an opportunity to work on this challenging topic and providing me ample guidance and support through the course of this research.

I would like to thank Dr. Manfred Huber and Dr. Leonidas Fegaras for serving on my committee.

I am grateful to Raman Adaikkalavan, Stephen Lobo, HimaValli Kona, Anoop Sanka for their invaluable help and advice during the implementation of this work. I would like to thank all my friends in the ITLAB for their support and encouragement.

I would like to acknowledge the support by the Office of Naval Research, the SPAWAR System Center-San Diego & by the Rome Laboratory (grant F30602-01-0543), and the NSF (grants IIS-012370 and IIS-0097517) for this research work.

I would also like to thank my husband, parents and family members for their endless love and constant support throughout my academic career.

October 21, 2003

ABSTRACT

SIGNIFICANT INTERVAL AND EPISODE DISCOVERY IN TIME-SERIES DATA

Publication No. _____

Ambika Srinivasan, M.S.

The University of Texas at Arlington, 2003

Supervising Professor: Sharma Chakravarthy

There is ongoing research on sequence mining of transactional data. However, there are many applications where it is important to find significant intervals in which some events occur with specified strength. We study approaches to convert point-based data into intervals, thereby predicting the next occurrence of the event. We formulate four approaches for significant interval discovery and enumerate their advantages and disadvantages. We compare the performances of various approaches in terms of computation time, number of passes, coverage and interval statistics like density, interval-length and interval-confidence. We propose an approach to clustering using the significant intervals produced. Furthermore, we use these intervals, which serve as representative areas of the dataset as input to a Hybrid Apriori algorithm to mine for

sequential patterns. We present the two types of interval semantics that can be used with sequential mining. We formulate an SQL-based Hybrid Apriori sequential algorithm that accepts intervals as input. Finally, we summarize the results and indicate the applications and conditions for which the various approaches can be used.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

The rapid improvement in the size of storage technology with the associated drop in the storage cost, and the increase in the computing power has made it feasible for organizations to store unprecedented amounts of organizational data and process it. The information and knowledge derived from it can be used for applications ranging from business management to market analysis to engineering design. However, discovering the hidden knowledge is not a straightforward task. To compete effectively in today's market, decision makers need to identify and utilize this information buried in the collected data and take advantage of the high return opportunities in a timely fashion.

The key here is the generation of previously unknown knowledge from huge datasets. The process of mining is driven by the outcome requirements. Based on what we want, a specific data mining technique is employed. The different data mining techniques and their outcomes are briefly discussed below [1]:

Classification: This is a process of grouping items based on a classifying attribute. A model is then built based on the values of other attributes to classify each item to a particular class. A training dataset is typically used for validating and tuning the model. The classification technique may be used, for example, to identify the most probable consumers for a product, based on their spending patterns.

Clustering: The process of clustering tries to group the data set in such a way that the data points in one cluster are more similar to one another while the data points in different clusters are more dissimilar. A similarity measure needs to be defined and the quality of the outcome, to a large extent, depends on the appropriateness of the similarity measure for the data set. The technique of clustering, for example, can be used to divide the market into distinct groups, so that each group can be targeted with a different strategy.

The basic difference between classification and clustering is that in classification, the classifying class is known previously (also known as supervised), whereas clustering does not assume any knowledge of clusters (unsupervised).

Prediction: The technique of prediction is based on continuous or discrete valued attributes. Previous history of the attributes is used to build the model. This technique is very commonly used for the prediction of sales of a product.

Deviation analysis: This technique compares current data with previously defined normal values to detect anomalies. Deviation analysis tools may be useful for security applications, where it may warn the authorities of any sharp deviation in the usage of resources by a particular user.

Association Rules: It is the process of identifying the dependency of one item(s) with respect to the occurrence of other item(s) in a data set. These models are often referred to as Market/Basket Analysis when they are applied to retail industries to study the buying patterns of customers. Here an attempt is made to identify a product "A"

with another product “B” to an extent that it can be said that whenever “A” is bought, “B” is also bought with high confidence (the number of times B occurs when A occurs).

Sequential Mining: Sequential pattern mining is mining of frequently occurring patterns related to time or other sequences. An example of sequential pattern is ‘ A customer who bought Fellowship of the Rings DVD six months ago, is likely to buy the Two Towers DVD within a month. Since many business transactions, telecommunication records, weather data and production processes fall into the category of time sequence data, sequential pattern mining is useful in the analysis of such data for targeted marketing, customer retention and so on.

The work in the field of association and sequence rule mining has resulted in a wide range of architectural alternatives for integrating mining process with the DBMS. These alternatives are depicted in Figure 1.1 [2] and are described below.

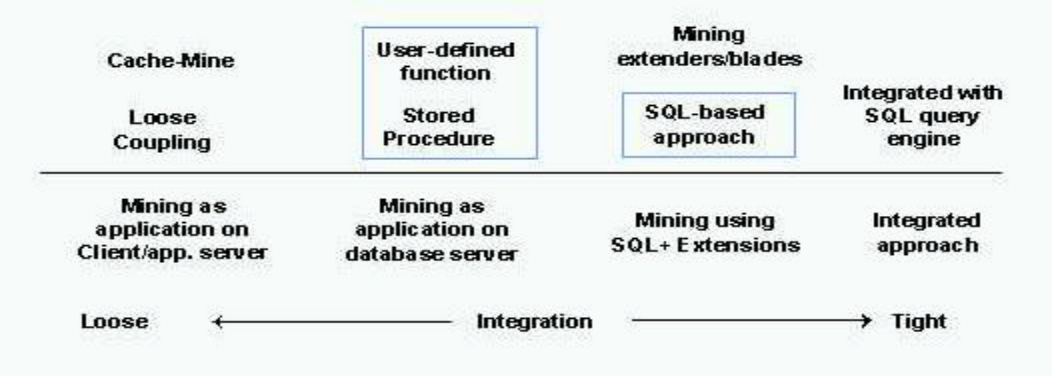

Figure 1.1: Architectural Alternatives

Loose Coupling or Cache based Mining: It is an example of the client/server architecture. The mining kernel can be considered as the application server. Here the

data is first fetched from the database and fed to the mining-kernel, which mines and pushes the results back to the database.

Stored procedures and user-defined functions: Here, mining logic is embedded as an application on the database server. Applications can be executed either in the same address space as the DBMS or a different one (fenced option in DB2). The flexibility in programming the stored procedure out-weighs their development cost.

SQL based approach: Here, for mining, queries are written in SQL. A mining-aware optimizer may be used to optimize these complex, long running queries based on the mining semantics. In this thesis, we use an SQL-base approach for sequence mining based on intervals.

Integrated Approach: This is the tightest form of integration that has no boundary between querying, OLAP, or mining. Mining operators or SQL (extended for mining) is optimized by the underlying system without any hints from the user. The long-term goal is to extend the current query optimizers to cover OLAP and mining along with SQL queries.

The focus of this thesis is on the combination of prediction of significant intervals in time-series data and use of the results further for sequential mining to determine interesting patterns. Exploration of the large data set in various ways using a number of parameters is taken into consideration in this work. . The applicability of the approaches proposed for both time-series data and transactional data makes the proposed approaches versatile.

1.1 Prediction

Data mining, typically makes use of statistical analysis when it comes to predicting the next value of a continuous variable rather than a categorical label. Prediction of continuous values can be modeled by statistical techniques of regression [6]. Many problems can be solved by linear regression and even more by applying transformations to the variables so that a non-linear problem can be converted into a linear one. Linear regression is the simplest form of regression where the data is modeled as a straight line. Bivariate linear regression models a random value Y, as a linear function of another random value X, that is,

$$Y = a + bX$$

Where a and b are regression coefficients specifying the Y intercept and slope of the line respectively. Many times, although the application cannot be modeled as a straight line to predict the value of Y given X, it can be viewed as Y=F(X). We focus on prediction of Y given X, which is a series of events changing with time. This can be described as a time-series database. Sometimes, there are several other variables that affect Y as it can have multiple values over a range of X. Multiple variables and the classification of the variables given below can make regression overtly complicated. There are four major characteristics that are used to categorize time-series data [6].

Long-term or trend movement: This indicates the general direction over which the time-series graph is moving over a long interval of time.

Cyclic movements or cyclic variations: These refer to the cycles, or long term oscillations about a trend line or curve, which may or may not be periodic.

Seasonal movements or seasonal variations: These movements are due to the events that recur annually. In other words, seasonal movements are the nearly identical patterns that a time-series appears to follow during corresponding months of each year.

Irregular or Random movements: These characterize the sporadic motion of the time-series due to random or chance events.

The common method of determining trend is to calculate the moving average, also referred to as smoothing of time-series. The concept of seasonal index is introduced to show the relative values of the variables in each group. To form the index, the data is divided into a set of partitions such as groups of months or groups of hours and the variation of the variable is monitored over each group to identify recurring patterns. However without any predefined knowledge, the above grouping is very arbitrary. An interesting solution would be to identify the groups from the data and look for patterns.

1.2 Mining Sequential Patterns

The sequential associations or sequential patterns can be represented as follows: when A occurs, B also occurs within a certain time. The difference between traditional association rules mining and sequence mining is that the time information is included both in the rule and also in the mining process in the form of constraints. In general three attributes characterize the sequence data: object, timestamp, and event. Hence, the corresponding input records consist of occurrences of events on an object at a particular time. Depending on the data and the problem in hand, various definitions of the objects and events can be used. As an example, an object can be a customer in a

book store and events are the books bought by the customer. As another example, an object can be a day and the events a switch-alarm pair of telecommunications network.

The major task associated with this kind of data is to discover sequential relationships or patterns present in these data. This can be very useful for prediction of future events. Several approaches have been proposed to tackle the problem. However the problems they assess, and the resulting solutions are very much problem dependent and often not suitable for other types of sequential data. Chapter 2 discusses the major approaches proposed for sequential mining in the literature along with the problems associated with them when used for our problem domain.

1.3 Focus of this Thesis

The predominant problem domain for this thesis is a smart home environment where discovery of patterns and their automation is of prime interest. However, the solution proposed by this thesis is not restricted to smart home environments alone but can be used for other domains where the need is to extract useful segments from the data based on user specified parameters. Some of the characteristics of a smart home are that it is formed of several devices, which interact with each other to give an inhabitant friendly, and reduced interaction environment. To accomplish this the smart home reacts to the changes in inhabitant's behavior by automating the operations of various devices instead of waiting for the inhabitants to manually interact with them. Following are some of the questions for which the answer is needed to accomplish the goals of a smart home

- When does device A turn on?

- When does device A turn on Monday or between 9 and 10am on Monday?
- What are the properties associated with device A when it is turned on at a specific time?
- Given the time, can the system tell with a degree of certainty that the device will be turned on at that time.
- Is there any other device that triggers the operation of device A (pattern)?
- What are the different patterns in which multiple devices interact with each other?
- Of these, which are the most frequently occurring patterns?
- What are the times during which the patterns occur?
- How many times patterns occur during a given time interval and their count?

The current work proposes to answer most if not all of the above questions raised by smart home applications. In addition to intelligent environments, many applications such as telephone logs, security logs and other time or numerical applications want to know, ‘Illustrate intervals in terms of groups of time or activity which best represents the data’ or ‘Illustrate intervals in terms of groups of time or activity, which have the following characteristics’. With telephone logs, periods of high activity are useful information for making informed business decisions. Magazine subscription logs can also be mined to determine the age groups that subscribe the most

to the magazine. Security logs can also be mined to extract intervals with certain characteristics. These intervals can be compared to values associated with normal conditions, to raise alerts when abnormal conditions are discovered. The characteristics of an interval can be its density, length or the strength. Given the above general problem domain, we divide our task into two phases: Identify the intervals which best represent based on interval characteristics provided by the user and use the intervals to identify frequently occurring patterns of different sizes and strengths. Four significant interval discovery approaches (termed SID[1], SID[n-1], SID[1,n-1] and SID[n-2, n-1]) have been formulated to discover intervals from the data. The user is provided with an option of choosing the approach, which best satisfies his/her requirements. Following the interval discovery, an algorithm for detecting patterns (termed Hybrid-Apriori, an SQL-based algorithm) has been formulated to accept interval-based input from various events to discover interactions between them.

The remainder of the thesis is organized as follows: Chapter 2 discusses related work by providing a brief introduction to the various algorithms proposed for sequential mining. Chapter 3 outlines the architecture of the MavHome, a smart home being researched at University of Texas at Arlington. The architecture, presented in this thesis, primarily concentrates on the database issues. Chapter 4 provides an in-depth discussion on the various interval discovery algorithms, their applications, their formulation, advantages of each approach and finally their performance evaluation based on several experiments. Chapter 5 discusses the interval-based sequential mining algorithm, its features and performance evaluation. We conclude by enumerating the

salient points of the thesis and discussing additional work that can be performed to improve its utility, efficiency and scalability in Chapter 6.

CHAPTER 2

RELATED WORK

The following sections provide a survey of the existing algorithms. WINEPI [3], MINEPI [3], GSP [4] are described in detail in sections 2.2, 2.3 and Section 2.4 provides a brief overview of several other algorithms, which are related to the current domain. Finally, we provide a brief introduction to our proposed solution in section 2.5.

2.1 Introduction

We discuss several approaches proposed to predict the occurrences of each event as well as algorithms, which discover interactions between multiple events. We provide a brief introduction to the prediction of single events since they primarily use statistical techniques and more information on sequential mining, which uses data mining techniques. A lot of work has also been done on prediction, from Markov's m^{th} order model to using statistical techniques in time series analysis. Markov's model [5] predicts which event will occur next, or when an event occurs using probabilities. This model is primarily used for pre-fetching of pages in computer architecture and other applications (e.g., speech recognition) from an input sequence, the next event is predicted using probability distribution functions. Time series analysis [6, 7] describes various techniques such as exponential smoothing, regression analysis, Box-Jenkins methodology to predict the value of Y (response variable) given X (predictor variable). Exponential smoothing is used to detect existing trends in the data such as an upward

trend observed in stock market prices over a period of time. Factors or components are added to the equation to take care of seasonal variations. However our approach of finding tight intervals does not involve any upward or downward trends, only patterns. Moreover the analysis can be used to predict intervals within the time hierarchy specified by the user, and does not determine the best hierarchy that fits the data. Regression analysis is the most commonly used approach to identify trends. The factors on which the predicted Y(t) depends are used to design a regression model (Linear or quadratic or complex). Many a times the response and the predictor variable have a relationship that can be modeled by adding polynomial terms to the basic linear models. Sometimes by applying transformations to the variables, we can convert the non-linear model to a linear one and other times models are intractably non-linear and cannot be converted to a linear model. The present thesis concentrates on solving the prediction question using data mining techniques like sequential mining as compared to statistical techniques. Universal formulation of sequential patterns is discussed in detail in section 5.1. A quick introduction to the constraints associated with patterns is described below. They give a better insight on the techniques used by the various algorithms for support counting. Support for a pattern refers to its number of occurrences within the dataset. The input parameter minimum-support ensures that all the discovered patterns have their support greater than the minimum-support. All algorithms use different techniques for support counting so as to make it as efficient and scalable as possible. Sequential patterns are associated with a set of timing constraints, which can be translated into:

Maximum Span (ms): The maximum allowed time difference between latest and earliest occurrences of events in the entire sequence,

Event-set Window size (ws): The maximum allowed time difference between latest and earliest occurrences of events in an event-set,

Maximum Gap (xg): The maximum allowed time difference between the latest occurrence of an event in an event-set and the earliest occurrence of an event in its immediately preceding event-set, and

Minimum Gap (ng): The minimum required time difference between the earliest occurrence of an event in an event-set and the latest occurrence of an event in its immediately preceding event-set.

The major differences between the traditional algorithms are found in the approach taken in the candidates generation and counting phase. The standard modes of sequence counting are,

COBJ: One occurrence per object

CWIN: One occurrence per span window

CWINMIN: Number of minimal windows of occurrence

CDIST_O: Distinct occurrences with possibility of event timestamp overlap

CDIST: Distinct occurrences with no event-timestamp overlap allowed.

More information on the universal formulation of sequences can be found in **[8]**. The remainder of this chapter discusses the significant sequential mining techniques starting with the algorithms proposed by [3].

2.2 WinEpi and MinEpi Approaches

WinEpi [3] is an algorithm, designed for discovering serial, parallel or composite sequences. Serial sequences require a temporal order of events whereas parallel sequences do not. Composite sequences are generated from the combination of parallel and serial sequences. In addition to the above, events of the sequences must be close to each other, which is determined by the window parameter. A time window is slid over the input data and only the sequences within the window are considered. The support for the sequence is determined by counting the number of windows in which it occurred. Referring to the timing constraints described above, the algorithm finds all sequences that satisfy the time constraints *ms* and whose support exceeds a user-defined minimum *min_sup,* counted with the CWIN method. The algorithm makes multiple passes over the data. The first pass determines the support for all individual events. In other words, for each event the number of windows containing the event is counted. Each subsequent pass *k* starts with generating the k-event long candidate sequences C_k from the set of frequent sequences of length k-1 found in the previous pass. This approach is based on the subset property of apriori principle that states that a sequence cannot be frequent unless its subsequences are also frequent. The algorithm terminates when no frequent sequences are generated at the end of the pass. WinEpi uses set of counters and sequence length for support counting of parallel sequences and finite state automata for serial.

An alternate way of discovering the frequent sequences is a method based on their minimal occurrences. In this approach the exact occurrences of the sequences are

considered. A minimal occurrence of a sequence is determined as having an occurrence in a window w=[ts,te], but not in any of its sub-windows. For each frequent sequence s, the locations of their minimal occurrences are stored, resulting in a set of minimal occurrences denoted by mo(s)={[ts, te] | [ts, te] is a minimal occurrence of s}. The support for a sequence is determined by the number of its minimal occurrences |mo(s)|. The approach defines rules of the form:

s'[w_1]-> s[w_2], where s' is a subsequence of s and w_1 and w_2 are windows. The interpretation of the rule is that if s' has a minimal occurrence at interval [ts, te] which is shorter than w_1, then s occurs within interval [ts, te'] shorter than w_2. The approach is similar to the universal formulation with w_2 corresponding to *ms* and an additional constraint w_1 for subsequence length, with CWINMIN as the support counting technique. The confidence and frequency of the discovered rules with a large number of window widths are obtained in a single run. MinEpi uses the same algorithm for candidate generation as WinEpi with a different support counting technique. In the first round of the main algorithm mo(s) is computed for all sequences of length one. In the subsequent rounds the minimal occurrences of s are located by first selecting its two suitable subsequences s_1 and s_2 and then performing a temporal join on their minimal occurrences. Frequent rules and patterns can be enumerated by looking at all the frequent sequences and then its subsequences. For the above algorithm, window is an extremely essential parameter since only a window's worth of sequences are discovered. Moreover, the data structures used for this algorithm can exceed the size of

the database in the initial passes. Interval based mining, on the other hand, allows the use of windows in two ways:

It allows events, which *start* less than window units away from the start time of the sequence to join an existing sequence irrespective of the event's finish time.

It also allows events with *start* and *end* within window units from the *start time* of the sequence join an existing sequence, which is more in tune with the traditional approach

In the first case, the user sets the window parameter based on the domain requirements. In order to consider an event as a candidate to join the sequence, the user needs to consider only the maximum span (ms) or the maximum time difference within which an event has to start. For example, if an event occurs within 60 minutes of the sequences, it can be considered to merge with it. This results in the generation of sequences with intervals lengths greater than the window parameter. The second alternative considers only those events, which complete within window units of the start of the sequence as candidates for the sequence. This approach is more in line with the traditional approach that uses points instead of intervals.

2.3 GSP

The GSP (Generalized Sequential Patterns) by [4] is designed for transactional data where each sequence is a list of transactions ordered by transaction time and each transaction is a set of items. It extends their previous work [9] by enabling specification of the maximum time difference between the earliest and latest event in an element as well as the minimum and maximum gaps between adjacent elements of the sequential

patterns. Thus the timing constraints included are *ws, xg* and *ng.* Support is counted using COBJ method. The algorithm works the same way as WinEpi described in the previous section. The difference is in the way the candidates are generated and their support counted. GSP introduces the notion of contiguous subsequences. The sequence c is a subsequence of s if any of the following holds:

- c is derived from s by dropping an event from its first or last event-set.
- c is derived from s by dropping an event from any of its event-sets that have at least 2 elements.
- c is a contiguous subsequence of c', which is a contiguous subsequence of s.

The determination of the support of the candidates is done by reading one data sequence at a time and incrementing the support count of the candidates contained in the data sequence. Given a set of candidate sequences C and a data sequence d, all sequences in C that are subsequences of d are found. Our domain considers data to be a series of events with timestamps with frequent patterns discovered between various events, in contrast to GSP, which discovers sequential relationships between items within a set of transactions.

2.4 Other Algorithms

CSpade [10] has the same application domain as GSP but involves more constraints that are versatile. CSpade is an extension of the earlier Spade [11] algorithm, which efficiently integrates constraint into the algorithm. The key features of Spade are the use of vertical layout and idlists, which include the object timestamp tuples of the

events. Equivalence classes partition the data set into several classes, which are processed independently. Problem decomposition using equivalence classes is decoupled from pattern search. Depth-first search is used for enumerating the frequent subsequences within each equivalence class. Our approach also considers a vertical database layout similar to that of Spade, partitions the database on the number of events and identifies intervals of occurrences based on user specified 'measure' independently.

Cyclic association rules [12] attempt to find rules, which are very prominent in a segment of data but are lost when the entire dataset is considered for mining. Partitioning the data correctly plays a crucial role in the discovery of these hidden rules. In addition to the mining techniques, many mathematical and statistical models [6, 7] also attempt to predict or discover the intervals by formulating an equation, which best describes the data. However these models have the drawback that they predict one answer based on historical data. One answer may not be adequate in several situations. In order to get multiple answers the data needs to be partitioned thereby predicting the best answer for each partition. This however would introduce some arbitrariness in the choice of best partition in the absence of appropriate guidelines.

[13] uses data cubes and Apriori mining techniques for mining segment-wise periodicity with respect to a fixed length period. In [14] MDL (minimum description length) principle, instead of support, is used to find candidate item-sets. The merit of this approach lies in the application of the periodicity of the event to prune unwanted sequences. This approach has some similarity to the first approach of [3] in the use of a sliding window defined by the user to find frequent episodes. Defining the periodicity,

however, can be an error prone task. As for [13], the algorithm discovers rules based on different measures for each time partition. Our primary interest is to find partitions which best describe the nature of the data. One of the distinct disadvantages of using traditional k-means [15, 16] or density based clustering algorithms [17, 18] is the determination of input parameters such as k or threshold density. Determination of the values of these input parameters either requires proficient domain knowledge or sufficient time for re-running algorithms with different inputs. Even though the primary aim of the present study is not cluster identification, to decide a better value for k and the threshold density, the number of clusters identified at the end of interval discovery algorithm along with their density and length can be used as input to the traditional clustering algorithms.

2.5 Conclusion

It is evident that number of algorithms have been proposed for solving the problem of frequent pattern discovery. Approaches that work for one domain do not necessarily form the best solution for another. The focus of our approach lies between clustering and sequential mining since both kinds of information are required to discover frequent patterns and answer queries related to intelligent environments. As mentioned above, our approach has enormous potential in intelligent environments where the key is to continuously learn from the surroundings and automate the inhabitant's activities. The MavHome (**M**anaging **A**n Intelligent and **V**ersatile **Home**) project is a multi-disciplinary research project at the University of Texas at Arlington (UTA) focused on the creation of an intelligent and versatile home environment [19]. Finding frequent

patterns enables us to automate device usage and reduce human interaction. The MavHome project focuses on the creation of a home, which acts as a rational agent. For finding patterns, the algorithm uses the intervals derived from various devices based on a user-defined confidence, density or interval length to predict the time of operation of each device. This information is used to answer user queries as well as to find sequential patterns. Representative intervals can be classified as the smallest intervals with highest density satisfying the desired interval-confidence. The following chapter discusses the MavHome architecture with a detailed description on its data collection process, data stream management system and prediction algorithms which form the heart of its database architecture.

CHAPTER 3

MAVHOME DATABASE ARCHITECTURE

This chapter gives a brief introduction on the requirements of smart homes followed by a quick overall view of MavHome architecture. The discussion continues with more emphasis on the database aspects of MavHome architecture. Section 3.1 gives an overall design of MavHome and its data collection process. Section 3.2 provides an in depth discussion on the implementation of the data collection process. Data Stream architecture and prediction driven sequential mining form the heart of MavHome on the database front. Section 3.3 highlights the needs and features provided by data streaming and section 3.4 gives a quick overview of prediction and sequential mining. The current chapter acts as a preview of prediction and sequential mining, and an in-depth discussion is carried out in chapters 4 and 5.

3.1 MavHome Architecture

Smart Homes link computers to everyday tasks and environments that have been traditionally considered as outside the purview of automation. Important features of such environments are that they possess a degree of autonomy and adapt themselves to changing conditions. The smart home assumes the control of devices and relieves the inhabitant of interacting with it. A smart home naturally requires location/context aware computing, allowing the environment to process information as if computational devices are everywhere. The MavHome Smart Home project is a multi-disciplinary

research project at the University of Texas at Arlington (UTA) focused on the creation of an intelligent and versatile home environment. The goal is to create a home that acts as a rational agent, perceiving the state of the home through sensors and acting upon the environment through effectors [19]. The agent acts in a way to maximize its goal, i.e., maximizes comfort and productivity of its inhabitants, minimizes cost, and ensures security. Some of the concepts have been taken from (http://mavhome.uta.edu/files/description/). More information on the overall architecture can be found in [19].

MavHome operations can be characterized by the following scenario. At 6:45am, MavHome turns up the heat because it has learned that the home needs 15 minutes to warm to optimal temperature for waking. The alarm goes off at 7:00, which signals the bedroom light to go on as well as the coffee maker in the kitchen. Bob steps into the bathroom and turns on the light. MavHome records this interaction, displays the morning news on the bathroom video screen, and turns on the shower. While Bob is shaving MavHome senses that Bob is two pounds over his ideal weight and adjusts Bob's suggested menu. When Bob finishes grooming, the bathroom light turns off while the kitchen light and menu/schedule display turns on, and the news program moves to the kitchen screen. During breakfast, Bob notices that the floor is dirty and requests the janitor robot to clean the house. When Bob leaves for work, MavHome secures the home, and starts the lawn sprinklers despite knowing the 70% predicted chance of rain.

Later that morning, a rainstorm hits the area that further waters the lawn. Due to a nearby lightning strike, the VCR experiences a power surge and breaks down while

taping Bob's favorite show. MavHome places a repair request and informs Bob at work of the event. Because the refrigerator is low on milk and cheese, MavHome places a grocery order to arrive just before Bob comes home. When Bob arrives home, his grocery order has arrived and the hot tub is waiting for him.

A number of capabilities are required for this scenario to occur. Some of these capabilities include active databases, prediction algorithms, mobility predictions, multimedia capabilities and many more. Active databases allow a house to be able to record inhabitant interaction and trigger sequences of events such as the bedroom light / coffee maker sequence. Machine learning allows for efficiently processing the data generated by the various sensors located around the house and taking immediate actions. Suite of prediction algorithms succeed in predicting inhabitant movement patterns and typical activities, and use that information in automating house decisions and optimizing inhabitant comfort, security, and productivity. For MavHome to track Bob's movements in between rooms and transmit his news program to him as well as to find him away from the home, multimedia and mobile computing capabilities need to be present. As can be observed from the scenario, MavHome automates the control of numerous devices within the home. To scale to this size problem, the MavHome agent needs to be decomposed into lower-level agents responsible for subtasks within the home, including robot and sensor agents, and this organization should be dynamically composable. Finally, these capabilities must be organized into an architecture that seamlessly connects these components while allowing improvement in any of the underlying technologies.

3.1.1 Data Collection Architecture for MavHome

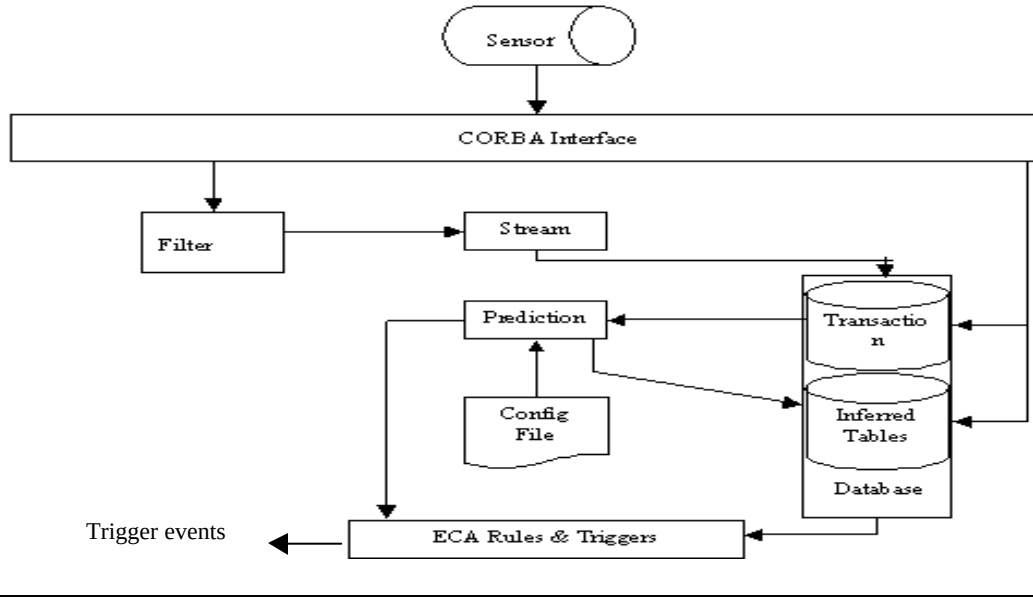

Figure 3.1: Database Architecture for MavHome

The Figure 3.1 illustrates the different components of the database architecture. MavHome project entails processing stream data in real-time from a large number of sensors (thermostat, lights, motion detectors, video etc.) and predicting and actuating agents in addition to storing the data in a conventional DBMS for querying and analysis. The various components in the above architecture are discussed below in detail.

Sensors monitor the environment (e.g., lawn moisture level) and, if necessary, transmit the information through the CORBA interface to the database. CORBA forms the intermediate layer through which communication is performed between the database and the devices. The database agent in MavHome registers itself with all the devices through a callback interface wherein an instance of the database object is given to each

device. Whenever a state change in a device occurs, CORBA invokes the database object with the information on the state change, which automatically inserts the values into the database. If all of the data from the sensors is not required to be stored in the database, a filtering operation can be performed at this level. This is very useful for devices such as temperature sensors, where temperature changes only above a threshold need to be recorded. However in the current implementation, filtering is not performed. Stream refers to the Data Stream Management System, which provides for the processing of queries on the stream instead of on the database. Detailed discussion on the DSMS architecture will be provided in the later section.

The database consists of two sets of tables one for transactional data and one for inferred data. The transactional tables record the day-to-day information on the change of state in each device and a set of inferred tables stores the results of the prediction algorithm. Prediction results refer to the set of algorithms that run on the stored data and discover the tightest intervals of state change associated with each device. In addition to the interval discovery, the algorithm extracts any frequently occurring patterns with different periodicities. A configuration file is used to input parameters to the prediction algorithms. More details on the design and implementation of these algorithms will be given in Chapters 4 and 5. A traditional database, as a passive repository to collect information, can store information but cannot react or provide assistance to various agents. The repository not only needs to provide the functionality of a traditional DBMS, but also should have capabilities to react to and in some situations proactively predict the trends for the immediate future and take appropriate actions. A combination

of collected information and current events need to be used for continuous monitoring of the house and its environment to initiate appropriate actions (triggering agents, initiating diagnostics and inferences, providing appropriate data for mining, periodic checking, etc.) in a timely manner. We use the concept of an active database, Sentinel in this case along with the event specification language Snoop and the ECA (event-condition-action) paradigm [20-23]. Currently it is possible to specify fairly complex ECA rules to invoke various computations (agents) based on how the information repository is updated. In summary the proposed architecture provides 3 important functionalities.

Data Collection and Storage

Data Stream Management System

Prediction algorithms

Design and implementation of data collection followed by a brief overview of the current DSMS architecture and Prediction algorithms is discussed below.

3.2 Design and Implementation of Data Collection and Storage

Data collection in MavHome consists of 2 steps. The first step includes communication with the CORBA interface and registration with all the devices. The second step comprises of the invocation of the callback object, which enters the device information into the database. The detailed description of the two steps is given below.

3.2.1 CORBA Interface

The MavHome communication layer consists of a nameservice to which all the room servers subscribe. Roomservers are the point of contact for all devices present in a

room. To register with a device, the communication layer provides a set of methods which allow an outside source to query the roomserver on the number of devices, device characteristics and their registration. The database being a repository of knowledge, registers itself with all the devices. Currently since all the roomservers are static, the collection program, invokes each roomserver in sequence. The X10LocationClientObject class connects to a roomserver given its name. Initially it gets the number of devices associated with the room by invoking the getNumberOfDevices() method. For each device in the room, the process then invokes the getDeviceInfo() method, to get the device characteristics such as device id, name, type, location and position. It then calls the subscribe() method and passes it a reference of X10Location_cb_impl. X10Location_cb_impl is the class associated with the callback object, which contains methods to update the database. The successful completion of registration ensures communication of any change in state of a device to the database. The main classes used for the above are X10Location_cb_impl, X10LocationClientObject and MavHomeDAL. MavHomeDAL class consists of all methods for inserting/updating the tables in the database.

3.2.2 Callback and Database Design Features

X10Location_cb_impl is responsible for updating the database with the current change in the device's status. Currently the information collected on all the devices is stored in Oracle RDBMS. The database has been divided into 2 sets of tables.

Static Tables

Dynamic Tables

Static tables contain information on the devices along with their respective characteristics, which does not change frequently. The tables are updated only when a new device is added to MavHome, or the location of a previous device is changed. Dynamic tables are the set of transactional tables, which record the periodic changes occurring to the devices. These also include the tables required to store multimedia information. Currently the set of static tables include *tbdevcice*, *tbdevicedescription* and *tbroom.* Tbdevice enumerates the different device types available with a brief description on the device type. Currently the set of device types available are illustrated in Table 3.1

Table 3.1: Different Device Types in MavHome

TXTDEVICETYPE	TXTDEVICEDESCRIPTION
X10_LAMP	Lamp
X10_MOTION	Motion detector
X10_SWITCH	Light switch
X10_FOB	Light switch
X10_GENERIC	Appliance
X10_CAMERA	Camera
X10_ APPLIANCE	Any appliance

Tbroom consists of information on the various rooms and their position and location as room_id and description. Tbdevicedescription contains the information on all devices including their id, name, type, room id, position and location. Dynamic tables include *tbtransload*, which contains deviceid, status, property value, timestamp, command source. Property value refers to any value associated with the device's state change. As an example, if the light is turned on and its intensity is set at 50, the property value of the lamp=50. CommandSource contains information on event that

triggered the state change. This event can be classified as manual, Heyu (through the web interface) or through RF_Remote(centralized switch). Manual refers to the situation when the user physically operatives the device to change its state. Heyu refers to the situation wherein the user controls the device using the web interface and RF_Remote refers to the operation of the device through a central switch (one for each room). All communication with database is conducted through the Mavhomedal object. Insertintotbdevicedescription() inserts the characteristics of the devices and insertintotbtransload() inserts the device's state change.

3.3 Data Stream Management System

Research on traditional database management systems (DBMSs) has been concentrated on the data that has been collected and stored. A wide variety of applications – network management, finance system, and sensor-based system (smart homes) – generate real time data streams and their processing requirements are very different from the traditional applications. Some of the common characteristics of these applications are summarized below:

- The data processed by these applications arrive in the form of a continuous stream, generated by sensors or embedded agents. Also the size of the data in such a continuous stream is unbounded.
- These applications mainly focus on the most recent data, which requires storing a predefined window-size of information for processing the data.
- These applications raise important events (e.g., congestion, alarm conditions) that are detected by various continuous queries. This entails

that the data processing system should fire pre-defined actions immediately once an event of interest is detected.

Some applications also have quality of service (QoS) requirements in order to respond to the events in a timely manner. As MavHome can be characterized by the above features, DSMS is a better alternative to a traditional RDBMS to respond quickly to events and answer queries continuously. It is clear that a traditional database management system is designed to process stored data efficiently. Furthermore, it assumes that the data is stored on a storage device that can be accessed as many times as needed, and QoS requirements are not considered at all. A number of architectures have been proposed to support the new requirements: the DSMS system by the database group at Stanford [24], the Fjord system by Berkeley [25], the Aurora system [26], and the NIAGARA system at university Wisconsin [27], to name a few here. A common characteristic of these architectures is that they associate a queue with each operator to support continuous queries over data streams. Hence a Data Stream Management System needs to be designed and developed for the above characteristics. Some of the important characteristics of DSMS are:

1. DSMS handles continuous streams of data wherein data is processed on the fly and results are generated. It does not store raw data on the disk but discards or archives the data elements once processed. Thus the resource limitation problem of storing each and every piece of information with large continuous streams is avoided.

2. DSMS provides a new set of operators, which can operate on continuous streams without blocking. Traditional Join and other Aggregate operators, which are difficult to use with streams, are modified to efficiently handle streaming data. They operate on windows, which define the boundaries for the continuous streams. Continuous operations are supported by moving the windows and changing their size. The results are evaluated on a windows worth of data and then the next window is considered for further evaluation.
3. Whenever the arrival rate of data in the stream exceeds the data processing rate, techniques such as sampling and histograms may have to be used in order to produce approximate results instead of losing data due to resource limitations (e.g., memory buffers)
4. DSMS succeeds in providing real time response to queries. Queries submitted to the system are run continuously against streaming data. Thus output is produced continuously and incrementally at the end of every window. Updates of routing table, network security, monitoring traffic are some of the other applications that require real time response of DSMS.

DSMS also supports different types of continuous/streaming queries. Classification of streaming queries along with the different types of queries supported by the current DSMS architecture is provided below. Streaming queries are broadly classified into:

- Predefined Queries
- Ad hoc Queries

Predefined queries: These are the queries that are made available to the system before the arrival of data relevant to it.

Ad hoc Queries: These queries are submitted to the system when the data stream has already begun. Hence for evaluating the queries requiring past information, the system needs to support some amount of storage. Since Ad hoc queries are not known beforehand, query optimization as well as finding common sub-expressions adds complexity to the system.

Predefined and Ad hoc Queries are further classified into:

- One-Time queries
- Continuous queries

One-Time Queries: These queries are evaluated only once over a given window. Once the query is evaluated it is removed from the system. It generates output once at the end of the window.

Continuous queries: These queries are evaluated continuously as the data streams arrive. Results are produced incrementally and continuously at the end of every new window. Most queries in streaming applications are continuous. Results can be stored and updated as streaming data arrives or can be streamed.

Figure 3.2 illustrates the various components of the current DSMS architecture. A brief overview of query processing is explained below.

Query Input: The user submits a query either as text (SQL query) or as GUI (could be box and arrow). This module enters the query created by the user into the system and registers the query with the module responsible for active queries.

Static Optimizer/ Alternate Plan Generator: Once the user submits a query, a static plan is generated. A plan is nothing but a partially ordered tree that decides the order in which operators are instantiated and executed. Once a plan is generated, the alternate plan generator generates all possible alternate equivalent plans that ensures the same output.

Processing Graph instantiates operators and their associated queues for evaluating queries. This is a forest of operators that constitute a global plan. A list of root operators and a list of leaf operators are maintained that help in graph traversal so as to instantiate operators in the correct order.

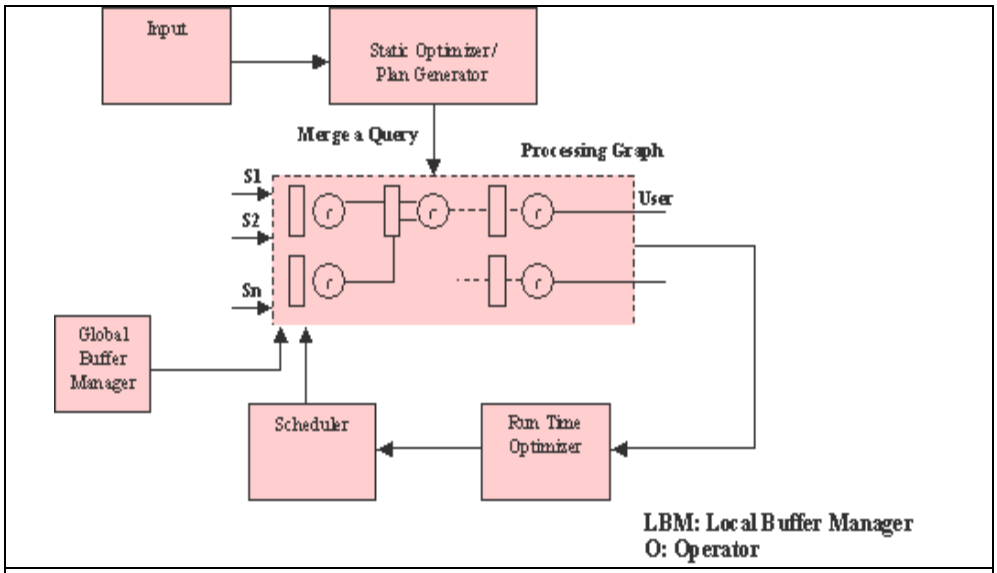

Figure 3.2: MavHome DSMS Architecture

Global Data Structure: This data structure keeps run time information such as priorities, state, number of output queues associated, parentlist, childlist and windowing information about active operators.

Global Buffer Manager is used for dynamic allocation and de-allocation of buffers. It works on various buffering policies that are implemented for the efficient management of memory. This module decides when to swap pages from disk or which operators to be given more memory over others based on priority.

Run-Time Optimizer performs run time optimization based on the quality of service observed. To ensure QoS, optimizer may ask scheduler to increase the priority of query, to allocate more buffers to respective operators or to select a better alternate plan from the sequence of plans available. Run time optimizer uses all these parameters intelligently to improve QoS.

Scheduler is used to schedule operators depending on the order defined by the global graph and the priorities assigned to the operators.

The above section provided a brief overview of the need for DSMS as well as its current architecture in MavHome. Further details on the design and implementation of DSMS can be obtained from [29] thesis.

3.4 Interval Discovery and Sequential Mining for MavHome

The importances of an interval-based representation for certain domains as well as the ways in which the interval information can be used for knowledge discovery are discussed herein. With time-series or numerical data, wherein the data is a series of events occurring at a point, intervals are a better alternative to timestamps. An interval

represents an area of activity and provides a more accurate and meaningful picture of the data set. Converting time stamped data into intervals results in coalescing the points and creating boundaries between them. The boundaries signify the start and end points within which the events occur. From a time-series data, interesting intervals can be extracted that satisfy certain properties, or clusters can be formed around these intervals to provide an overall view of the dataset with its distinctive areas. Again, the formulation of clusters can be based on overlapping or disjoint intervals. One of the distinct disadvantages of using traditional k-means [16,15] or density based clustering algorithms [17, 18] is the determination of input parameters such as k or threshold density. Determination of the values of these input parameters either requires proficient domain knowledge or sufficient time for re-running algorithms with different inputs. The primary aim of the interval discovery algorithm is not cluster identification. However, in order to decide on a better value for k and the threshold density, the number of clusters identified at the end of interval discovery along with their density and length can be used as input to the traditional clustering algorithms. The intervals can also be used as input to a sequential mining algorithm equipped to deal with interval based input. This results in a smaller dataset as input to sequential mining, as the interval discovery eliminates noise and low support points. Instead of generating candidate item sets and pruning them based on their support, eliminating them from participating in the pattern discovery saves time, computation and achieves scalability. It can be argued that the effort saved during support counting and pruning is compensated by the extra computation performed to discover intervals. However

interval discovery, in its own right, extracts some valuable characteristics (outlined above) of the data set as compared to mere support counting and pruning in the initial passes of a traditional sequential mining algorithm.

As mentioned before, for applications where intervals provide better semantics, the algorithms described in this thesis can be used very effectively. As an example, in a smart home that has X number of devices, best intervals for each device can be discovered first. They can then be combined using an interval based sequential mining algorithm to discover interactions between devices. The interval discovery approach also provides a natural way to partition the problem and is amenable to parallel processing. We strongly believe that representing events with intervals has several advantages: Firstly, it provides an opportunity to explore and identify significant intervals, and in the process provides a better understanding of the underlying data. Secondly, it provides nuggets of information about the data that can be used for clustering the data by traditional algorithms (that require input parameters such as the number of clusters, density, etc.). Finally, it reduces the size of the data used for discovering sequential patterns. In summary, there are several advantages to extending the traditional mining algorithms to work with intervals instead of points. However, converting the existing algorithms in a naive manner to achieve the above may not make use of interesting interval-based characteristics to improve the output quality and efficiency. The problem of discovering tight intervals and frequent patterns (derived from MavHome) can be described as follows. Given a history of event occurrences (single or multiple) over a period of time, how can one predict the best (shortest or a

bounded interval with some notion of confidence) interval in which the event occurred? Initially, we look at the problem with a single event and then extend them to multiple events and predict the occurrences of combination of event patterns (or episodes) on sequence data.

To illustrate the problem consider Figure 3.3**,** which shows events of a lamp being turned on (each day) over a 30-day period (time on x-axis and number of occurrences on y-axis). It is of interest to predict when MavHome should turn on the lamp each day to capture the inhabitant's behavior. This necessitates finding intervals with "good" confidence and providing mechanisms to explore and analyze the intervals using different values of support, confidence etc. Figure 3.4 shows our initial approach, which identifies four intervals as "interesting" that satisfy a user-defined confidence of 80% (the device was on 24 days out of 30). The x –axis represents the time of occurrence of the event and the y-axis represents the number of times the event occurred at a particular time referred to as strength. Cluster B has more than one significant interval in which the device was turned on more than once with 80% confidence from [1:07-1:47]. A suite of interval discovery algorithms have been formulated to identify intervals for all devices in MavHome given the domain requirements and input parameters.

More information on these algorithms will be provided in Chapter 4. After extracting such intervals from all the devices, they are used as input to a sequential mining algorithm to discover frequent patterns and sequences along with their time of occurrence.

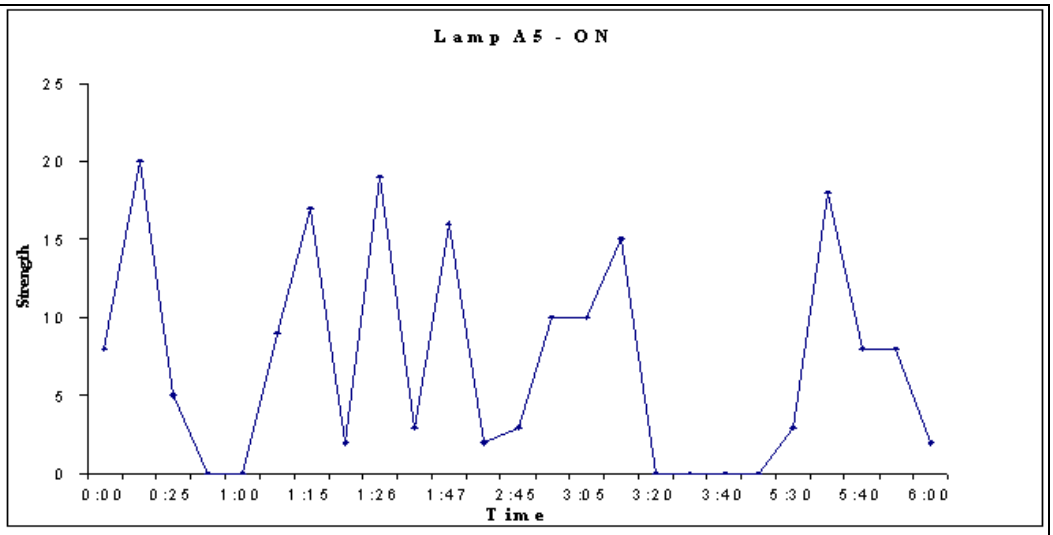

Figure 3.3: Sample of MavHome Data

This is accomplished by the interval-based sequential mining algorithm discussed in Chapter 5. The above algorithms give us the information needed to automate device operations thereby reducing inhabitant interactions.

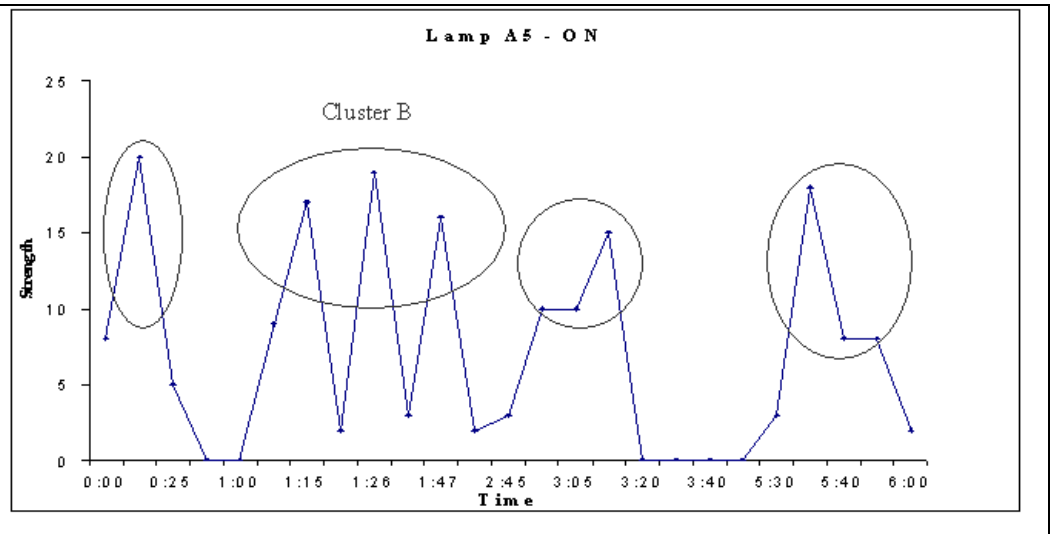

Figure 3.4: Identification of Intervals and Clusters from Figure 3.3

CHAPTER 4

SIGNIFICANT INTERVAL DISCOVERY

In chapter 2 we discussed the various approaches proposed in the literature and their drawbacks with respect to MavHome requirements and other related applications. In this chapter we revisit our goal and discuss our proposed solution with its performance results. Domain related modifications to the approach could be easily integrated with the main algorithm to get improved performance. Several approaches to the problem are described in this section along with their significant differences. Section 4.1 discusses the conversion of point-based data into intervals and the various properties associated with an interval. Section 4.2, 4.3, 4.4 respectively deal with different approaches such as Naïve, significant interval discovery and modified significant interval discovery along with their advantages and disadvantages. Section 4.5 describes the implementation of the approaches and their performance results and comparison with each other.

4.1 Interval Discovery

Not many data mining algorithms discuss the formation of intervals on time series data based on the interaction of events. The data collected from MavHome exhibits the interactions between the inhabitant and the devices. This results in large amount of information stored over a period of time for each device, with data value at every point in the time scale. The primary aim is to coalesce the points and convert

them to intervals. Start and end times associated with an event signifies the occurrence of the event within it with certain characteristics of the interval such as its strength, length and density. With large numerical and time series data, events occur with a high degree of certainty not at specific points but within tight intervals (sets of points). Therefore intervals give us more information on the total strength of the device activity during a period as compared to points. Based on this observation, the data related to each device is mined separately to identify the intervals with maximum strength.

MavHome can greatly benefit from an algorithm that can infer the usage patterns of each device as well as interactions between different devices. MavHome consists of numerous sensors deployed around the house that monitor different activities. The devices include lamps, thermostats, computers, coffee machine, TV, blinds, lawn mower, garage door opener and refrigerator to name a few. Every change in the state of a device is recorded in the database. Abstractly such data can be viewed as a collection of events, where each event has an associated time of occurrence. Multiple events can occur at the same time, which means different events can have the same timestamp. Discovery of the frequent sequences and automation of the home using discovered sequences could reduce the interaction between the inhabitant and the home. The crux of the study is to find, when each device is turned on and off and to determine the interaction between the devices (such as the causality of their usage), using the intervals. This gives the answer to the exact time of occurrences of each device/event as well as of the frequent patterns (sets of devices).

Even though interval discovery can be used with various applications as described in Chapter 1, the MavHome scenario is used for further discussion with typical examples.

Table 4.1: Sample of MavHome Input Data

Device Id	Status	Time Stamp	Support
A5	ON	08:30	10
A5	ON	08:31	14
A5	ON	08:32	6
A5	ON	09:40	2
A5	ON	09:41	10
A5	ON	09:50	15
A5	ON	10:00	20
A5	OFF	10:01	10
A5	OFF	10:02	15
A5	OFF	10:04	15
A5	OFF	10:40	2
A5	OFF	10:50	3

The sample data in Table 4.1 gives us the information about the device, its status whether it was ON or OFF, the times of occurrences and the strength at those occurrences. In simpler terms, it can be seen from Table 4.1, device A5 was turned ON 10 times at 08:30 during the period of X days for which the data is being mined. A cursory glance at the sample data will suggest that instead of working with all the points, for sequential mining, or prediction, working with the high-density areas alone, will result in a faster and efficient application. For example, intervals 08:30 to 08:32 and 09:41 to 10:04 indicate the tight periods of high activity. On the other hand points such as 10:50 or 10:40 not having much support, do not form meaningful intervals with their neighbors. This implies that they need not participate in the discovery of sequential

patterns, as they would be pruned during the initial passes. In addition to sequential mining, the tight intervals can be used for prediction of the next occurrence of the event. In the above example only the times of their occurrence are considered, implicitly including only the daily events in the database. However that is not a restriction and week type or week number can be included to indicate the week of occurrence, thereby forming the input for discovering weekly events. Week type is whether the day was a weekday or a weekend and week number refers to the day of the week. Various levels of periodicity can be used to discover the highly active intervals within that periodicity. As we zoom out, that is move from daily to weekly to monthly, additional information is discovered, which could not be inferred at the lower level.

For example, with daily periodicity, the days or week information is ignored and the number of times the device was operated at a particular time is considered. At the weekly periodicity, an additional attribute could be used to indicate whether the day is a weekday or weekend, the time when it was operated and the number of times it was operated at that time on the weekday. As an output the time intervals on a weekday during which the device was turned on can be provided. The same can be found on a week level by dividing the week into specific weekdays (Monday, Tuesday, Wednesday and so on). The output reveals the best intervals on Monday the device was turned on. This feature plays an important role in discovering events which are weekly in nature. If the lawn mower is operated every Sunday at 11:00 am, this information could be captured only if the algorithm was run with a periodicity of week type or weekday.

The intervals obtained at one periodicity for different devices can be used to form sequential patterns. A daily event will have an interval in its weekday/weekend and weekdays counterpart. However a weekly event, which occurs only on Monday, will be discovered only when the algorithm is run on a weekday basis. In the following sections, the design and implementation of the algorithm will be described in detail. In addition to using the algorithms with different periodicities, there are two options as one moves to higher levels of periodicity: to deal with the data at the respective periodicity or discover the time intervals in terms of hours and minutes at that periodicity. In the former, intervals of weeks with most activity are discovered. In the latter, for events, which occur only once or twice a week, the algorithm outputs the time intervals in which they occurred in each day of the week. The former can be used by applications, which do not need data in the granularity of time but of weeks or any other periodicity of interest. By running the algorithm with different periodicities, intervals of occurrence for each device with that periodicity is obtained, which in turn is used to predict the next occurrence of the event as well as to discover sequential patterns between devices with the given periodicity.

Table 4.2 describes the MavHome data when mined for weekly periodicity. The same algorithm is run on different partitions to obtain the intervals particular to that partition. Intelligence can also be built into the application so that, the algorithm can run on various levels of periodicity automatically to discover various patterns depending on the domain.

Table 4.3 gives the input format used with numerical domains to mine for the set of tightest intervals satisfying user constraints. There are several approaches that can be used to obtain intervals from a data set.

Even though clustering is an option that comes to mind, it is difficult for a user to ascertain before hand the input parameters usually needed for a clustering algorithm. Several parameters such as number of clusters, density etc. are required as inputs to various clustering algorithms. The present approach aims to find intervals with more meaningful parameters given as input from the user such as interval-confidence, length, and density of the interval. In addition to the discovery of intervals, characteristics of these intervals can be used to make a better decision on the input parameters needed for some of the traditional clustering algorithms.

Table 4.2: Input Format to discover weekly events with output at time granularity

Device Id	Status	Weekday	TimeStamp	Support
A5	ON	Monday	08:30	10
A5	ON	Monday	08:31	14
A5	ON	Monday	08:32	6
A5	ON	Monday	09:40	2
A5	ON	Monday	09:41	10
A5	ON	Monday	09:50	15

Table 4.3: Input Format to discover weekly events with output at granularity of week

Device Id	Status	Week	Support
A5	ON	1	10
A5	ON	4	14
A5	ON	5	6
A5	ON	10	2
A5	ON	12	10
A5	ON	14	15

4.1.1 Interval Definitions

Traditional time-series or numerical data sequences can be represented with an event timestamp model. Event *e* is associated with a set of timestamps $\{T_1, T_2, \ldots T_n\}$ which describe its occurrence over a period of time. The notion of periodicity (such as daily, weekly, monthly, etc.) is used to group the event occurrences. For each event, the number of occurrence at each point can be obtained by grouping on the timestamp (or periodicity attribute). Timestamp considered can be as specific as using date-hours-minutes-seconds or as general as a weekday (e.g., Monday). In either case grouping on the timestamp provides the number of times a particular event occurred at that time. Thus the data can be represented as $< e \{T_1, O_1\}, \{T_2, O_2\}, \ldots \{T_n, O_n\}>$ where Ti represents the timestamp associated with the event e and Oi represents the number of its occurrences for the event e. Oi is referred to as the *strength* of the event *e* at Ti.

A point can be represented as an interval with the same start and end point. Strength at the point thus reflects the strength of the interval. When the interval consists of several points, strength of the interval is the sum of the strength of the points that form the interval. In addition to strength, the notion of *Interval-Confidence* is introduced, which gives the average number of occurrences within an interval over a period of time. Therefore interval-confidence can be represented as a ratio of the strength and the maximum number of occurrences needed for the event to become a certainty. If interval-confidence >= 1, the event is considered certain to occur within the interval. Interval-confidence is always > 0, since an interval cannot exist without at least one occurrence in the history. Any interval-confidence between 0 and 1 provides a

degree of certainty with which the event occurs within the interval. For time-series data, the maximum number of occurrences needed for the Interval-Confidence to be 1 can be defined as the number of units (e.g., days, weeks, etc.) worth of data being mined. For a numerical domain, it can be defined as the total number of occurrences observed over the entire dataset. This results in the interval-confidence being the ratio of its strength to the strength of the entire dataset.

Intervals are represented as *[T1, T2, s, l, d, c]* where *T1* and *T2* represent the start and end of an interval, s represents the strength of the interval, *l* denotes the length of the interval *(T2-T1), d* indicates the density and *ic* represents the interval-confidence. Let *N* be the number of units (days, weeks, months, etc.) of the time-series data. For numerical domain, let *S* be the sum of the strengths at all the points in the dataset.

Length *(l): T2-T1*

Density *(d): s/l*

Interval-Confidence *(ic): s/N* (for time-series data)

: *s/S* (for numerical data)

Density is a characteristic associated with intervals with lengths >=1.Strength of an interval, denoted by s, is defined as the number of times an event occurred within the interval. For example, if the device was turned on 20 times within

[10:00-10:15] then *s* has the value 20, or

if age group of 20 bought 100 items, *s* is 100 for the interval [20-20]. If in 30 days of data, the device was turned on 20 times within

[10:00-10:15] then *ic*=.66 [20/30] to identify a daily event and when identifying weekly events, *ic*=s/number of weeks.

As an example of a numerical domain, a magazine subscription company interested in finding age groups with interval-confidence >=40% will discover either the best age group with the most orders or the top 2 age groups with confidence > 0.4 and <= 0.5 (maximum interval-confidence of 2 intervals which together span the entire dataset is 0.5)

4.2 Significant Interval Discovery Algorithm (SID)

The significant interval discovery algorithm proposed in this study can be partitioned into 3 phases:

- Preprocessing (one time processing)
- Interval Formation (Iterative process)
- Cluster Formation (one time processing)

Table 4.4: Input Data set in a Vertical format

Device	Status	Time of occurrence	Strength
LivRmLamp1	On	10:00	4
LivRmLamp1	On	10:01	7
LivRmLamp1	On	10:03	10
LivRmLamp1	On	11:04	8

Preprocessing converts the input dataset to a format suitable for the interval formation algorithm. It uses any domain specific parameters, if provided to form the

first level intervals from the point based data. For MavHome, every state change of a device is recorded in the database. The table therefore contains the device name, its status (On/Off), and timestamp of the state change. The preprocessing step converts this stream of data, into a vertical layout format as shown in Table 4.4. Strength at a point refers to number of times the device was changed to the mentioned status at that point. Table 4.4 gives the information about the Living Room Lamp1, the points at which it was turned 'On' and the number of times (over the entire dataset) it was turned 'On' at that point.

These set of points are converted into the first level intervals as shown in Table 4.5. Assuming the above data was collected over 30 days, interval-confidence = Strength/30 and density= Strength /Length.

Table 4.5: First Level intervals from Table 4.4

Device	Status	Start Time	End Time	Strength	Density	Length	Interval-Confidence
LivRmLamp1	On	10:00	10:01	11	11	1	0.3667
LivRmLamp1	On	10:01	10:03	17	8.5	2	0.5667
LivRmLamp1	On	10:03	11:04	18	0.3	61	0.6

The interval formation phase uses the first level intervals as input and follows an iterative process to generate the tightest intervals satisfying the user input, referred to as significant intervals. Cluster Formation uses the significant intervals to identify any clusters of high activity in the dataset. If the above algorithm is unable to discover any

clusters, the output of this phase is the significant intervals discovered in the second step. Before discussing the details of the three phases, the parameters used by the algorithm are described.

4.2.1 Configuration File

The interval discovery algorithm accepts a number of parameters from the user (from a configuration file) to compute the set of significant intervals and clusters. The input parameters accepted by the algorithm are:

- Minimum Strength
- Window
- Measure
- Measure Value
- Period
- Interval Semantics
- Sequential Window
- Number of Threads
- Approach

Minimum Strength and Window are parameters used in the preprocessing phase to prevent the formation of certain first level intervals from the point-based data. They are domain specific and optional; they make the process more efficient and accurate when provided. *Minimum Strength* ensures that only intervals with strength greater than specified value (threshold) form an interval. When dealing with data over a long period of time, the user can specify Minimum Strength=4, which prevents any first level

intervals with Strength < 4 to be formed. In the above example, minimum Strength of 11 eliminates [10:00-10:01] interval from being created and thereby any of its supersets. This parameter helps to identify noise and prevents certain points from participating in further passes. *Window* specifies the maximum length of a first level interval. This is another optional parameter, which ensures that points in different periods of interest do not combine to form an interval. For the above example, Window=30 prevents [10:03-11:04] from being created. All the intervals created in the preprocessing step are candidates for expansion in the following steps until a threshold condition is met. Window parameter prevents the formation of meaningless intervals.

Measure is an option by which the users can specify the field on the basis of which the interval selection is performed. The choices are interval-confidence, interval length, density and a combination of interval-confidence and interval length. Depending on the *Measure* chosen, its value is provided in *Measure Value*. If the *Measure* is specified as interval-confidence and Measure Value is 0.8, the algorithm discovers the smallest intervals in which LivRmLamp1 was turned on at least 23 out of 30 days or 3 out of 4 weeks. However, when the user is interested in discovering intervals of a specified length, the *Measure* can be set to the interval length with *Measure Value* giving the maximum length of an interval.

The above two *Measures* can be combined to discover intervals above the threshold interval-confidence and within the interval length specified. To improve the efficiency of the algorithm, the user can set the *Thread* parameter, which indicates an upper bound on the number of threads spawned by the algorithm. *Period* and *interval*

semantics are parameters used by the sequential mining algorithm. *Period* signifies discovering patterns within the specified periodicity (Daily, Weekly, etc). When mining for weekly events, the maximum number of occurrences needed for interval-confidence are the number of weeks worth of data. Based on the nature of the dataset (i.e. time-series or numerical), weekly periodicity results in a different output. With time-series, the algorithm discovers, the time-intervals for each event per weekday (Monday, Tuesday, etc). For Numerical data on the other hand, group of weeks are clustered to extract tightest intervals with required characteristics. The above is the same for any other periodicity such as monthly, yearly etc. More information on *interval semantics* is provided in the sequential mining section, which in a nutshell describes the joining criteria for events. *Sequential Window* refers to the window to be used for sequential mining. Depending on the interval Semantics chosen, the frequent sequences output may or may not be equal to the window size, as will be discussed later. None of the above parameters are necessary for the termination of the algorithm. However their specification helps in the generation of meaningful intervals. For example, if no Measure is specified, the algorithm will discover larger intervals, which in some cases can span the entire dataset. *Approach* refers to the interval discovery algorithm selected by the user. The user has four choices Naïve, SID[n-1], SID[1] and SID[n-2], which are discussed in detail in the remaining sections of this chapter. Based on the nature of the dataset, a suitable approach can be chosen to derive the output in terms of the number of intervals generated or length of the intervals generated.

4.2.2 Naïve Approach

The Naïve approach of identifying intervals based on a user-specified measure is discussed below. It can be viewed as an exhaustive approach in which every point can potentially combine with every other point in an iterative fashion until the threshold value is reached. All possible candidate intervals are generated before selecting the smallest interval satisfying the user input. The algorithm is given below.

```
//The first level intervals generated from the
preprocessing step
Store the first level intervals in intervalInput
//Base case for intervalOutput
Store the first level intervals in intervalOutput
//increasedIntervalOutput identifies the intervals
generated in the last pass
For each element in intervalOutput
     If ( Interval.measure_value <
     ConfigurationFile.measure_value)
           Join with adjacent interval from intervalInput
           to form a longer interval
     Store the new interval in increasedIntervalOutput
           If no new intervals are formed in
           increasedIntervalOutput
           //interval discovery terminates
```

```
            Break
        Else
            Replace intervalOutput with
            increasedIntervalOutput
```

In the first pass of the above algorithm, every 3 points combine to form an interval. That is, the first level interval (made of 2 points) is lengthened by the addition of its closest point. Along similar lines, the third pass results in the combination of every 4 points to form an interval. From Table 4.5 the intervals formed using Naïve approach in the first pass are shown in Table 4.6

Table 4.6: Intervals generated in the first pass by Naïve Approach

Device	Status	Start Time	End Time	Strength	Density	Length	Interval-Confidence
LivRmLamp1	On	10:00	10:03	21	7	3	0.7
LivRmLamp1	On	10:01	11:04	25	0.4	63	0.8

The Naïve approach performs a total of $O(n^2)$ combinations by merging each interval with all possible intervals until the user-defined measure is reached. It grows the intervals generated by the last pass with the first level interval set to form new intervals. From Table 4.6 the interval created in the next pass would be [10:00-11:04].

This approach is computationally expensive for large datasets. Furthermore, it does not help in reducing the number of intervals, especially when the number of intervals generated is large (n-1 intervals from n data points). However for smaller

datasets, with distinct intervals and well-defined input parameters, the Naïve approach can be used to generate all possible intervals satisfying the threshold measure (defined in the configuration file). This forms the complete input set for sequential mining, which discovers all possible interactions between events. The naïve approach always combines intervals of the nth pass with the intervals from the first level.

4.2.3 SID[n-1] Approach

The next approach – SID[n-1] (**S**ignificant **I**nterval **D**iscovery) approach – reduces the number of passes and the number of intermediate intervals generated by utilizing interval characteristics. SID uses a divide and conquer method for generating new intervals in the nth pass by using only the n-1th (or current) pass. Furthermore, instead of blindly growing the interval generated in the previous pass (as in the naïve approach), it makes use of density and interval-confidence of the current and adjoining intervals to ascertain whether a merge should be made. Since interval-confidence = s/N or s/S and its denominator is a constant, an increase in the numerator will always increase the interval-confidence value. Therefore it can be stated that interval-confidence is a monotonically increasing function.

Lemma1: If an interval *s* has interval-confidence *ic* then all of its supersets i.e., {∀a | a ∈ S and a.starttime <= s.starttime and a.endtime >= s.endtime } will have confidence >= *ic*.

The above lemma follows directly from the definition of interval-confidence.

In addition to interval-confidence, density of an interval relates its total strength with its length. High-density points indicate sudden burst in activity and merging them

with the lower density counterparts has a smoothing effect over the new interval. Merging intervals using density helps to eliminate the local maxima in favor of the global maxima around high-density points. When an interval reaches the threshold *measure* level, it is added to the output set. This interval does not participate in further merges because any further merging will only result in greater interval-confidence or length than required thereby forming an extra-fit, as indicated in Lemma 1. If no new intervals are generated in the subsequent pass, the interval-merging phase terminates. SID uses the following condition in the algorithm to prevent all possible candidate generation.

```
For each element in intervalOutput
          If ( Interval.measure <
          ConfigurationFile.measure)
               If interval.density <=
               AdjacentInterval.density or
                 interval.confidence <=
               AdjacentInterval.confidence
                         Join with adjacent interval from
                         intervalInput to form a longer
                         interval
               Store the new interval in
               increasedIntervalOutput
```

Following are the four cases, which occur when two intervals try to merge with each other:

Case 1:[T1, T2, c1] – [T2, T3, c2]: d2 >= d1 & c2 >= c1

Case 2:[T1, T2, c1] – [T2, T3, c2]: d2 >= d1 & c2 <= c1

Case 3:[T1, T2, c1] – [T2, T3, c2]: d2 <= d1 & c2 >= c1

Case 4:[T1, T2, c1] – [T2, T3, c2]: d2 <= d1 & c2 <= c1

Figure 4.1 represents the four cases graphically. The dashed lines are the adjacent intervals, which merge to form the interval represented by the solid line. The x-axis represents the time and the y-axis represents the density. The nature of interval-confidence is same as strength, which is indicated by the area within the interval (density * interval-length). A uniform distribution is used to illustrate an interval because the SID algorithm uses density, interval-confidence and length alone to decide whether an interval merges further. The density of the interval is assumed to be the mean of that interval's uniform distribution. After every merge the above-mentioned characteristics are re-calculated so as to allow for future merges.

The first and third cases have the adjoining intervals with greater mass as compared to the first interval, and merging the 2nd interval with the 1st results in [T1, T3, c3] with increased interval-confidence. Case 2 indicates a particular scenario in which the adjoining interval is of smaller mass but exhibits a sudden burst in activity, causing an increase in density.

The second interval forms the local maxima around the region, which is smoothed by its merge with the first. Case 4 indicates a natural end of an interval, since

compared to it; the adjoining interval acts as noise and can be ignored. As the number of passes increase, the number of merges that form new intervals start decreasing. It can also be seen that the total number of merges for the algorithm is proportional to n log n.

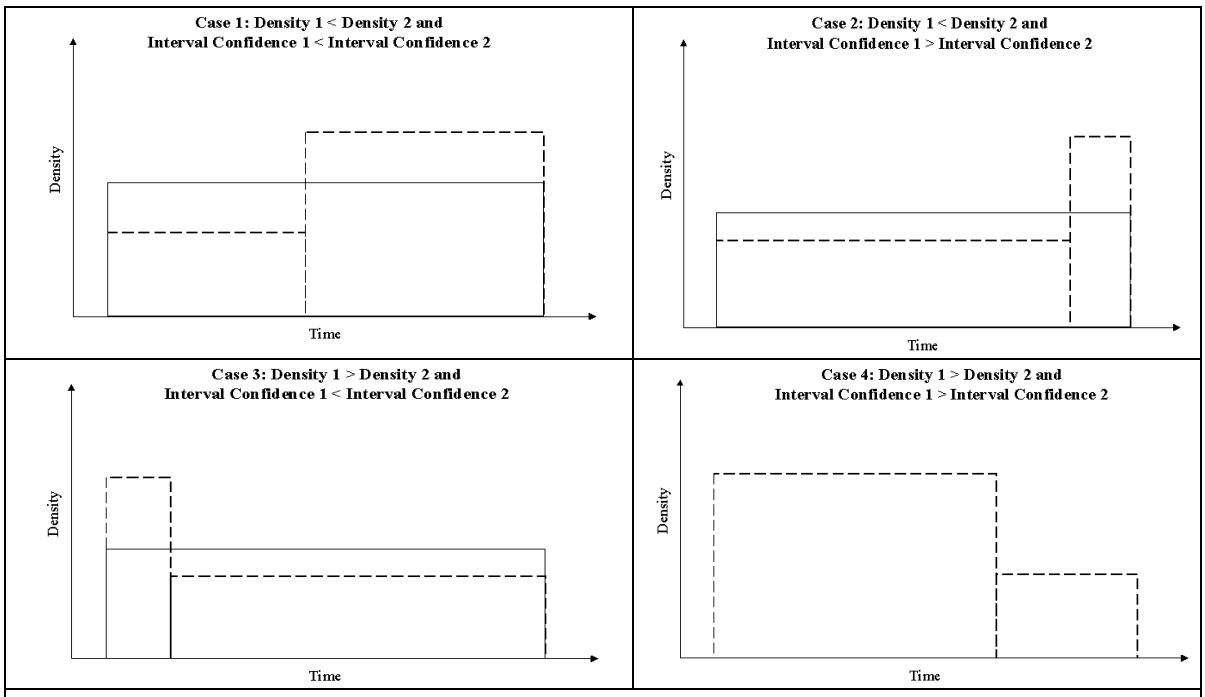

Figure 4.1: Different Cases for Interval Merges

In every pass, two intervals merge to form a larger interval. In the next pass, it is the larger interval, which is a candidate for further increase. This causes the size of the interval to grow at an exponential rate with the number of prospective merges reducing in an exponential rate. Due to this property, SID[n-1] produces longer intervals as compared to Naive. The amount by which SID[n-1] extends the output interval for the same measure level as Naïve, is referred as extra-fit. Since intervals are checked before they continue to merge, the extra-fit produced by SID[n-1] can never exceed twice the optimal length (as produced by Naive) of the interval. This is because interval

characteristic needs to be below threshold for a merge to happen. In the worst-case, an interval just before reaching its optimal length can merges with its adjacent interval that has already reached the threshold value, thereby resulting in an extra-fit. Even then the extra-fit is not more than twice its optimal length because the interval merged before its threshold was reached. The aim of the algorithm is to form the best interval from each point without enumerating all possible intervals from it.

Moreover the algorithm is flexible enough to stop the interval growth at a point especially when the point looks uninteresting and signifies noise. When experimenting with large data sets, SID[n-1] is faster, accurate and produces fewer intervals as compared to Naïve. Despite the smaller amount of intervals, SID[n-1]'s results are comparable to Naïve with large datasets as there always exists more than one significant interval in a high-density area. For example, if 10:00 to 11:00 represents a general area of high activity with 10 significant intervals with interval-confidence=0.8, Naïve enumerates the 10 tightest intervals whereas SID[n-1] enumerates 3 with interval-confidence=0.9. Therefore SID[n-1] maintains the same coverage provided by Naïve in addition to reducing the number of intervals generated. Coverage can be defined as the set of all distinct points included within the significant intervals produced by Naïve. This is very beneficial when the data set has to be mined for sequential patterns. The disadvantage could be as follows: if there exists only one interval around a range of points and SID's four cases (Figure 4.1) are unable to form an interval around them, the range might be missed. If SID[n-1] is used with smaller datasets, the conditions mentioned above can be relaxed to correct the above error. Instead of the

If Interval.density <= AdjacentInterval.density

or Interval.confidence<=AdjacentInterval.confidence

Merge the interval

One can use,

If Interval.density <= w_1 * AdjacentInterval.density

or Interval.confidence<= w_2* AdjacentInterval.confidence

Merge the interval

Where w1 and w2 are weights that need to be determined.

4.3 Alternative Sid Algorithms

The naïve and the SID[n-1] approaches can be viewed as approaches that represent the end points of the spectrum. The Naïve approach combines intervals from each pass with the input intervals and hence can generate the tightest intervals that satisfy the user-specified measure. On the other hand, the SID[n-1] approach combines the intervals from each of pass with only itself to from the intervals in the next pass. As a result, the intervals generated may be slightly larger than the ones generated by the naïve approach. In order to reduce the extra-fit of the SID approach, two more approaches are proposed that cover points in the spectrum formed by the naïve and the SID [n-1] approach.

In many situations, the application demands the discovery of all possible intervals with minimum extra-fit. The Naïve method accomplishes this task but at the cost of high computational time. Fortunately, SID can be modified to increase the number of intervals produced and reduce the extra-fit without listing all possible

intervals in more than one-way. This can be viewed as an option to the user to explore the data set in different ways so as to understand the domain characteristics and determine the input parameters. The user can decide, based on the required accuracy on one of the following approaches:

- The naïve approach to explore the intervals exhaustively (SID[1])
- The SID[n-1] approach to explore the intervals minimally
- The SID[1, n-1] approach to produce most of the intervals produced by Naïve and reduce the extra-fit as compared to SID[n-1]
- The SID[n-2, n-1] approach to produce improved coverage and reduced extra-fit compared to SID[n-1]

The value(s) in the parenthesis indicate(s) the intervals from the passes used for merging to form the intervals of the nth pass. Based on this convention, the naïve approach can be expressed as SID[1] as the first level intervals are always used to merge with the $(n-1)^{th}$ pass intervals. SID[1, n-1] uses intervals from first pass to merge with the intervals in the $n-1^{th}$ pass with priority given to intervals from the 1^{st} pass while merging.

4.3.1 SID[1, n-1] Approach

For the SID[1, n-1] approach, the SID[n-1] algorithm can be used with the same merging criteria (Figure 4.1) with minor modifications. SID[n-1] uses the current pass alone to create the next pass intervals. In this algorithm, the current pass [n-1] as well as the first pass [1] intervals are used for merging purposes. The four merging criteria cited in Figure 4.1 are used with higher priority given to the first pass over the

last pass intervals. The reasoning behind using first pass and its higher priority is, after reaching a certain *measure* level; the interval grows incrementally when merged with the first pass. In the initial passes the intervals grow faster since adjacent points may not always have the satisfying conditions to grow. However after a few passes, the intervals try to merge with the first level interval as compared to the last pass. Owing to this, the growth is curtailed, extra-fit is reduced and intervals produced increase as compared to SID[n-1].

Table 4.7: First Level Intervals

Start Interval	End Interval	Strength	Interval-Confidence	Density
1	3	5	.4	2.5
3	4	5	.4	5
4	8	10	.5	2.5
8	9	10	.5	10
9	12	15	.75	5

Table 4.7 represents the first level intervals produced from point data for 20 days. From Table 4.9 it can be seen that because of first pass intervals, the extra-fit by SID[1, n-1] reduces thereby increasing the number of intervals produced with confidence >=1. First iteration of interval-merge results in the formation of {[1-4], [3-8], [4-9]} for all approaches.

Table 4.8: After 2nd Pass of SID[n-1]

Start Interval	End Interval	Strength	Interval-Confidence	Density
1	9	30	1.5	3.3
3	12	40	2	2.5
4	12	35	1.75	2.1
8	12	25	1.25	3.5

SID[n-1] as seen from Table 4.8 produces {(1-9), (8-12)} as output. Intervals {(3-12), (4-12)} are eliminated because its subset (8-12) itself is above the threshold value (Lemma 1).

Table 4.9: After 2nd Pass of SID[1, n-1]

Start Interval	End Interval	Strength	Interval-Confidence	Density
1	8	20	1	2.8
3	9	25	1.25	4.1
4	12	35	1.75	2.1
8	12	25	1.25	3.5

SID[1, n-1], on the other hand, produces {(1-8), (3-9), and (8-12)} as output. Interval (4-12) is eliminated in favor of (8-12). Naïve approach with no extra-fit would produce {(1-8), (4-9), and (8-12)} with (3-9) eliminated in favor of (4-9). SID[1, n-1] results in increased number of intervals with reduced extra-fit. A salient point to note is that in either case, the coverage is complete with every interval produced by naive is encompassed by SID[n-1] and SID[1, n-1].

4.3.2 SID[n-2, n-3] Approach

SID[n-2, n-1] replaces the use of the first pass in SID[1, n-1] by the penultimate pass. This does increase the number of intervals greatly as compared to SID[n-1] but reduces the extra-fit and increases the coverage produced. This can be explained along the same lines as SID[1, n-1], since the interval growth is not completely exponential. The smaller interval of the penultimate pass is used to merge producing a better-fit interval instead of the corresponding larger last pass alone, which has the greatest length intervals. Figure 4.2 represents a subset from 9:30 to 14:40 over 30 days of data, mined for the discovery of intervals with interval-confidence >=0.8 with window=60. The

figure illustrates the difference in the intervals created and shows only one interval per area. Over the high density area of 9:55 to 10:11,

Naïve discovered: {9:55-10:06, 10:00-10:07, 10:02-10:08, 10:05-10:10, 10:07-10:11}

SID[n-1] discovered: {9:55-10:10, 10:04-10:11}

SID[n-2, n-1] discovered: {9:55-10:06, 10:00-10:08, 10:02-10:10, 10:03-10:11}

SID[1, n-1] discovered: {9:55-10:06, 10:00-10:07, 10:02-10:08, 10:03-10:09,10:07-10:11}

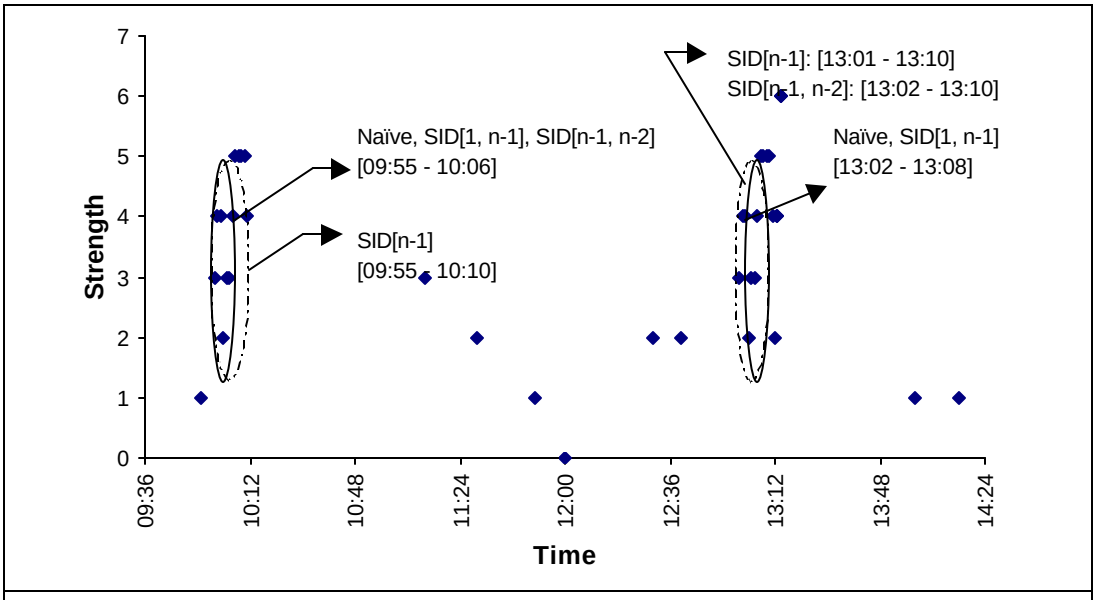

Figure 4.2: Comparison of Intervals with SID Variants

The following conclusions could be derived from the above: Naïve with n^2 combinations and larger number of passes produces all possible user-defined intervals from the dataset. It uses the latest and the first pass of intervals as input. SID[n-1] with n log n combinations uses the recently created intervals alone to generate the next set of

intervals. SID[1, n-1] increases the number of intervals substantially and uses the intervals in the first and the last pass as input. Finally, SID[n-2, n-1] reduces the extra-fit by using the last 2 passes alone as input. Both SID[1, n-1] and SID[n-2, n-1] give higher priority to a smaller length interval during merges.

4.4 Implementation of SID and Cluster Algorithm

The implementation of interval discovery can be broken into two parts: identification of intervals (SID approach) and identification of clusters from the intervals discovered (cluster identification).

4.4.1 Implementation of SID

The implementation of the interval discovery algorithm consists of the following main classes:

- InputStruc
- InputMavhome
- OutputStruc
- InputFormat
- IncrementInterval
- Semaphore
- Sync

InputStruc gives the method and attributes associated with the input object. This is a standard object, which can be used with any type of application. It is made of *date*, which signifies the field, to be converted into intervals and *support* that gives the strength at that field. Field can be either time-series or numerical data. InputMavhome

class provides information related to the MavHome application that is not given by InputStruc. Whereas InputStruc is the general structure of an input object, InputMavhome is an object that provides domain specific information. When the algorithm is to be used with a different application, the inputMavHome class can be replaced with the attributes associated with the specific domain.

OutputStruc refers to an interval class and is associated with the following attributes:

- Start time
- End time
- Total Supp
- Time Diff
- Density
- Interval-Confidence

It is also associated with an important function *supersetOf (outputStruc,Vector).* The function returns an integer value depending on whether the object is a subset/superset of the object passed as an input parameter to the function. Vector object is filled with the start and end times of itself and the object passed as a parameter converted to minutes format in case of time series data. This information is useful in determining the nature of the overlap in case of overlap during cluster identification phase.

InputFormat class reads the configuration file, identifies the input parameters and spawns objects of type IncrementInterval for each event. At the end of the last thread, it starts the sequential mining phase by invoking the MaximalSequence object.

To control the number of threads generated classes Semaphore and Sync are used. More details on the MaximalSequence object will be provided in the next chapter.

IncrementInterval forms the heart of the interval discovery algorithm, which accepts as input the vector of points and support associated with an event and all the parameters given in the configuration file. Point data is converted into intervals by joining adjacent points in FindCombinations(). Depending on the approach chosen, intervals are formed in an iterative manner. From the set of intervals created in the last pass, only those satisfying the threshold measure form the set of significant intervals. Since in every pass, an interval is extended with the same start point, its position in the vector remains unchanged. This property can be used while implementing SID[1,n-1] and SID[n-2, n-1] and the search to find the adjacent interval in a vector is done once. Otherwise the time spent for this search can become significant with large datasets. Once the position of the adjacent interval is found, the vector of the previous pass in SID[1, n-1] and SID[n-2, n-1] can be indexed by the same position and a second search can be avoided.

4.4.2 Implementation of Cluster Algorithm

The design and implementation of cluster identification module requires an algorithm that produces a set of cluster intervals generated from the significant intervals. The current implementation generates clusters between 2 adjoining significant intervals. For example, if 1-3 and 3-8 are the 2 significant intervals discovered, the algorithm generates a cluster of 1-8. Flexibility can be added to the cluster identification step by allowing intervals within x units of each other to form a cluster.

Also the cluster implementation can be modified to create overlapping or disjoint clusters. The data structure used to find a cluster has the following 3 attributes associated with it:

Name: A unique identification of the cluster

Head: The superset interval of the cluster. All the intervals associated with the cluster form subsets of this interval.

Cluster-Intervals: All the intervals including the head form the cluster. From this set, a subset of tightest intervals representative of the cluster can be obtained.

Head values of all the clusters identified as output are representative of the clusters within the dataset.

The psuedocode for finding clusters from the significant intervals is as follows:

```
Identify the intervals from increasedintervalOutput whose
measure >= user_specified measure
For each interval (I) in the finalset
     For all Heads (H) of the existing cluster ( C )
          If (I is a subset of H)
               I ∈ C
          Else if (I is a superset of H)
               C.Head ∈ C
               C.Head = I
          Else
               Form a new cluster C with C.Head = I
```

```
C.Name = 'Cluster ' +Max(C_Name)+1;
```

The above results in a set of clusters, which are overlapping in nature. Ordering criteria can be used in the form of length of the cluster, its density, interval-confidence and number of non-overlapping representatives with their *measures* above the threshold. After this step only the points with sufficient support either in terms of interval-confidence or length are chosen for sequential mining. This is the main advantage of SID [n-1] and its variants over the traditional approach.

Over the same data used for Figure 4.2 (30 days over 9:30–14:30), cluster identification algorithm with SID[n-1] produced the clusters shown in Figure 4.3. The current implementation of the algorithm illustrates overlapping cluster, which can be modified to produce disjoint clusters alone.

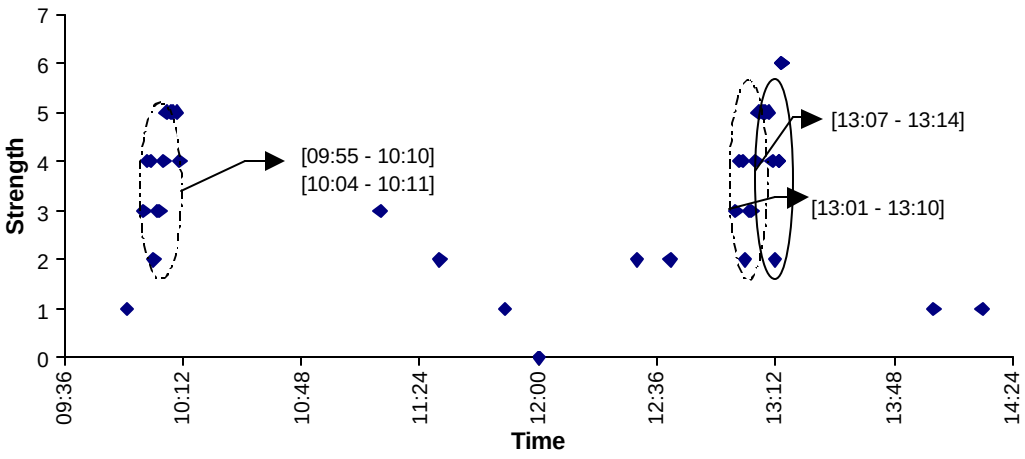

Figure 4.3: Identification of Clusters

In addition, the algorithm can be run on a representative data set and the number of clusters obtained by the algorithm can be fed to a traditional clustering algorithm

such as k-means [1], [2] to get improved results. Similarly, the average density of the intervals produced can be fed to a density-based clustering algorithm such as (DBSCAN) [3], [4] to improve results.

4.5 Experiment Results

A number of experiments were performed with the MavHome data to verify the correctness and scalability of SID and its variants with respect to the Naïve approach. Experiments were also performed in order to observe the effect of parallelization on response time. In the MavHome scenario, data collected for all devices is partitioned into events for each device and the intervals of occurrences are discovered for each device independently. This can be done for most of the data sets even if the data is collected as a single set. Since the present focus is not on the interaction between different events, each event can run as an independent thread. The user can input a parameter, which controls the number of threads generated. A number of experiments have been performed with the data collected from MavHome ranging from 1-3 months. Several experiments were conducted on the same dataset, comparing both the Naïve and SID approaches on the time taken, passes needed and more importantly intervals missed by SID[n-1], SID[1, n-1] and SID[n-2, n-1] as compared to Naive. The time recorded for all the experiments was calculated as an average of four runs.

It was found that by increasing the number of threads, and thereby introducing parallel computation, the time taken to discover the intervals reduced, as can be seen from Figure 4.4. The time taken was averaged over four runs for each dataset. The dataset ranged from 46000 records in 1 month to 140,000 records for 2 months and

201,000 records in 3 months. The x-axis indicates the number of threads used. The experiments suggest a significant improvement with increase in the number of threads. However, as the data set size increases, beyond a number of threads, the improvement does not continue which was observed for the 3-month data for 15 threads. Another observation is that the time taken by SID did not increase dramatically from 1 to 3 months, suggesting that with large data sets, the convergence is faster than small data sets. This also means that the computation time does not increase proportional to the data size (can be inferred from 1 thread response time for the three data sets).

It is important to understand the effect of SID alternatives on the number of passes. As seen from Figure 4.5 the number of passes taken by the Naïve approach for 3 months of data is much more than that taken by SID[n-1] or SID[1, n-1]. The largest improvement is from 75 passes to 12 (for D5-ON) and the smallest improvement is from 32 to 9 (for F5-ON). The largest improvement is 600% and the smallest is 300+ %. Even among the SID alternatives (not including the naïve), there is significant reduction in the number of passes. The reason SID[n-2, n-1] has not been shown in the graph is that it always lies between SID[1, n-1] and SID[n-1] in the number of passes.

In order to compare the coverage produced by SID[n-1], the intervals produced by the Naïve algorithm whose start times were not included within the start and end times of at least one interval produced by SID[n-1] were counted. The same calculation was performed with the end times produced by the Naïve algorithm to verify the presence of at least one interval in SID[n-1], which encompassed it. The number of points missed by SID[n-1] was less than 2% of the intervals generated by the Naïve

approach. Given the substantial reduction in the number of passes and the insignificant missed percentage in the coverage of the alternatives, there is no advantage to use the naïve approach for most of the data sets.

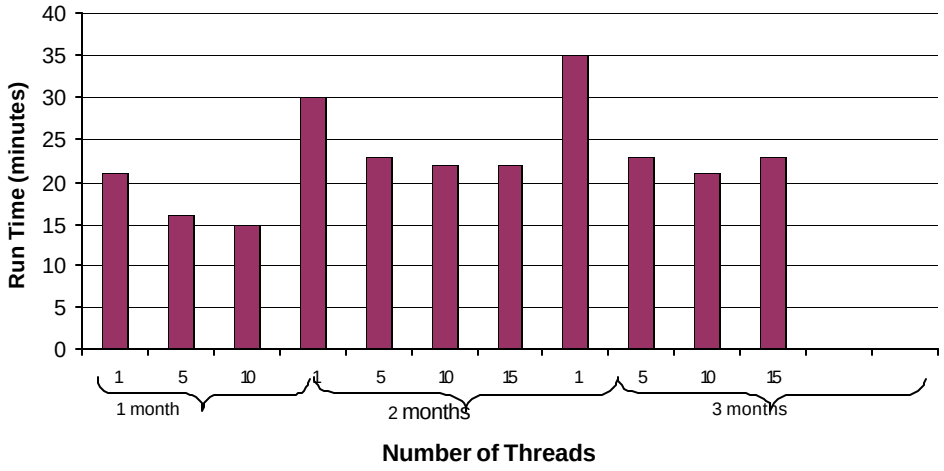

Figure 4.4: Response time of SID[n-1] for 1-3 months with daily periodicity

In addition to the number of passes, the number of intervals produced by the SID alternatives is also of interest. It was observed that SID[n-1] produced about 1/3 the intervals produced by Naïve and SID[1, n-1] about 2/3 as can be seen from Figure 4.6. There is not only a significant reduction in the number of passes, but also in the number of intervals generated. The reduction in the intervals by 33% and 66% with the use of SID[1, n-1] and SID[1] approaches respectively without sacrificing the coverage implies that these alternatives could be considered as useful substitutes to improve upon the performance of the naïve approach.

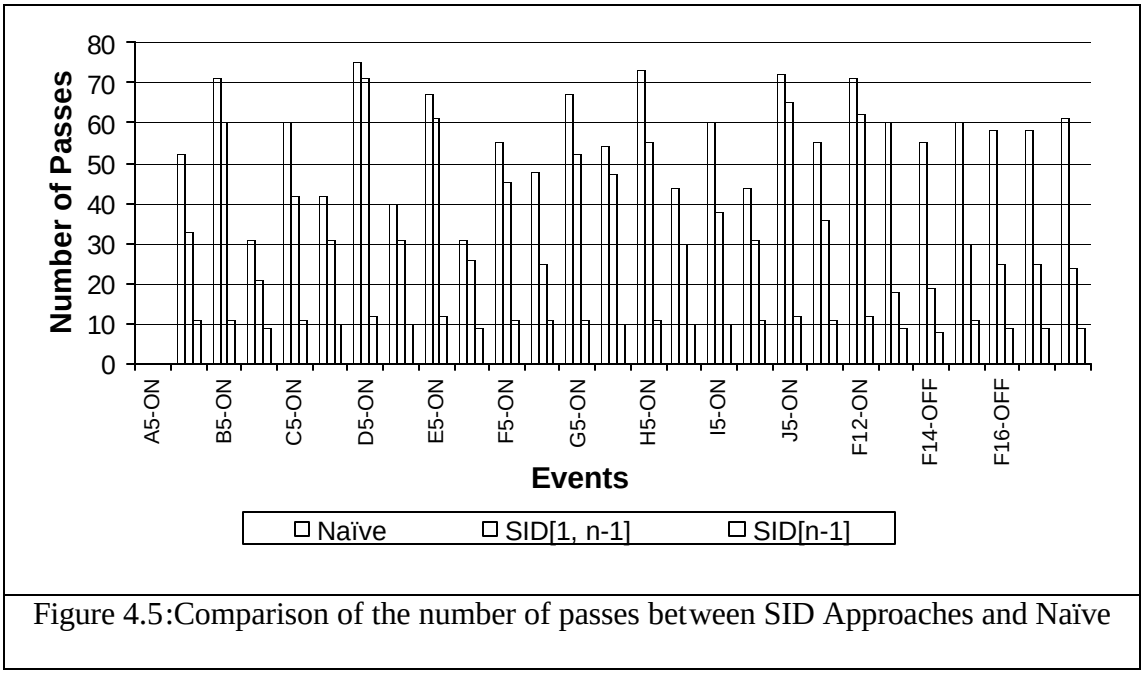

Figure 4.5:Comparison of the number of passes between SID Approaches and Naïve

The Graph shown in Figure 4.7 enumerates the percentage deviation in the interval length of the significant intervals between Naïve and various SID approaches.

Percentage Deviation = Avg((Sid[*i*].endtime-Naïve.endtime)/Naïve.Starttime-Naïve.endtime)

when Sid[*i*].starttime=Naïve.starttime and *i* is the SID alternative

Percentage deviation indicates the amount of extra-fit produced by SID and its variants as compared to Naïve. The first month gave an average of 14.5% whereas for the 3 months it reduced to 5.43 % when SID[n-1] was used. This shows that with larger datasets, SID[n-1] improved on performance as well as accuracy. From the graph we can infer that the percentage deviation not only improves with SID alternatives, but converges to the naïve approach when the data size increases.

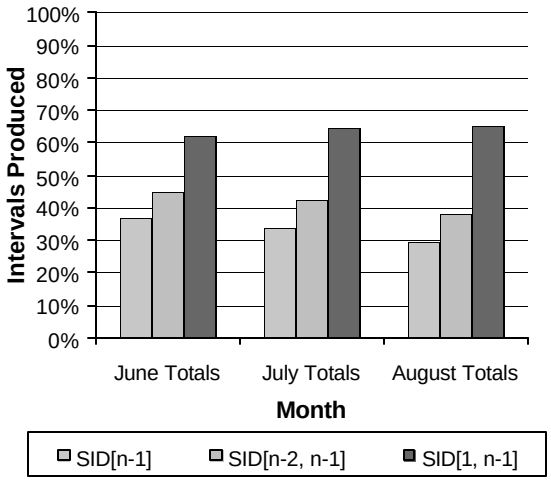

Figure 4.6:Comparison of Intervals produced by all approaches to Naïve method

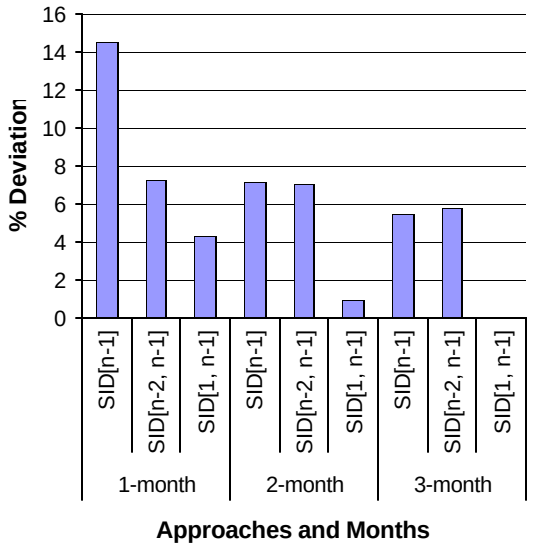

Figure 4.7:Percentage Deviation of SID variants Vs naive

The effect of domain specific parameters such as minimum support and window on the number of intervals has also been studied. Table 4.10 shows that reduction of the window parameter or specifying minimum constraint reduces the number of intervals discovered and changes interval characteristics. We can observe from Table 4.10 that window made a greater impact on the June dataset as compared to August. This is because in August, enough data had been accumulated with interactions at every point, because of which window value of 10 did not make much of an impact. Minimum Support however made a comparable difference in intervals irrespective of the sizes of the dataset.

This parameter reduced the number of valid intervals that were formed in the first pass due to the strength criteria and caused a rippling effect. Since these parameters

have a great impact on the intervals discovered, they should be set after careful consideration. The results in Table 4.10 was observed by running the algorithm with 1-3 months of data with the following parameters, Measure of 2 and interval-confidence as 0.8.

Table 4.10: Impact of Minimum Support and Window on #of intervals discovered

Parameters	**Month**	**#of intervals**
Window=10	June	3310
Window=60		3621
Minimum Support=3		2361
Window 60	July	5235
Window=10		5079
Minimum Support=4		4144
Window 60	Aug	5152
Window=10		5101
Minimum Support=5		4422

Naïve and SID variants produce a set of intervals for each event. It was of great interest to verify the one interval chosen by all approaches in a given range with the same ordering criteria applied over the set. This would be very useful in situations where the next occurrence of a device was to be automated.

The experiment was performed with June data for daily events and all approaches were run with the same configuration file before posing the question. As can be seen from Table 4.11, since there was more than one interval around 9:30 to 10:00, all approaches gave similar answers.

Table 4.11: Comparison of best intervals by all approaches over 09:00-10:00 for B5 - ON

SID[1]	SID[1, n-1]	SID [n-2, n-1]	SID[n-1]
09:53-10:00	09:53-10:00	09:58-10:04	09:53-10:00

The ordering criteria used by all approaches was to pick the interval with highest density followed by smallest interval length and lastly highest confidence in case of a tie.

Table 4.12: Comparison of best intervals by all approaches over 09:00-11:00 for B5 – ON

SID[1]	SID[1, n-1]	SID [n-2, n-1]	SID[n-1]
10:48-10:54	10:49-10:55	10:49-10:55	10:49-10:55

Even as the range over which the question was posed increased, the answer produced by different approaches did not differ by much.

4.6 Conclusion

In conclusion, the performance of SID[n-1] is significantly better for large data sets and is comparable in accuracy (deviation and coverage) to that of the Naïve approach. If one wants to reduce the deviation further, one of the other two alternatives (SID[1, n-1] or SID[n-2, n-1]) can be used. Our experiments indicate that the Naïve approach is not needed for the data sets. The interest in the naïve approach is to use it as a reference for benchmarking the performance and accuracy of other SID approaches. Of course, for smaller data sets, the naïve approach may still be acceptable from the

performance and accuracy point of view. Of the others, SID[n-1] is good for most of the data sets as it reduces the number of passes dramatically and the number of intervals by almost 66%. As this serves as input to the clustering and sequence mining algorithms, computation time of clustering and sequential mining will also reduce proportionally. Moreover the processing time does not increase much with SID variants (SID[1,n-1] and SID[n-2. n-1]) due to the inherent indexing property available to the algorithm for reducing extra-fit. Finally, it could be concluded that the algorithm finds the best intervals that can be formed from each point. Many of these intervals are weeded out due to the fact that a smaller interval is found, starting at a later point but ending at or before the previous interval. Many other domain specific constraints can also be added to the algorithm to reduce the number of intervals selected from the final pass. As an example, when two significant intervals start within x units of each other, the less denser or the larger interval among the two can be eliminated. These conditions can be added seamlessly in the algorithm after the final phase either in the main memory side or on the database side. On the main memory side, the final output can be sorted based on the start times and density in a vector. Binary search can be used to find and eliminate the intervals which fall within x units of the each other in the vector. On the database front, a stored procedure can be written to order the intervals based on density and select only those records with the maximum density among the records within x units of its start time. Finally as compared to previous approaches, our approach allows the user the option to input domain specific constraints or run the algorithm with no domain knowledge. In addition to the significant intervals, the algorithm also provides

certain statistics on the data, which can be used as input to other traditional algorithms such as DBSCAN, K-means, etc. In interval discovery phase, each event was considered in isolation. The next step would be to consider the events together and discover frequently occurring patterns among them. This problem is addressed in the next chapter since most of the current algorithms do not deal with interval-based input.

CHAPTER 5

HYBRID-APRIORI SEQUENCE MINING

This chapter forms the second phase of the algorithm, which takes the significant intervals as input and discovers frequent sequences from the data set. Section 5.1 describes the general representation of a sequence defining some of the commonly used terms. Section 5.2 explains hybrid-apriori algorithm, support counting in hybrid-apriori and how and why it differs from the traditional approach. Discovery of maximal and frequent sequences are explained in Section 5.3 followed by performance evaluation in section 5.4. Log files are maintained for interval discovery as well as sequence mining are described in Section 5.5.

5.1 Sequence Definitions

A general sequential pattern can be expressed as a Directed Acyclic Graph (DAG). This graph is formed of nodes and directed edges where the former represents events or event-sets and the latter represents the order of their occurrence. During the pattern discovery process, edges are added by introducing nodes dynamically or shrunk by collapsing one of its incident nodes into another. In our approach nodes and edges are added in succession upon their discovery extending the graph. For the universal formulation of sequences some of the concepts and figures are adopted from [8]. Edges can be associated with a set of constraints called edge or timing constraints, which can be translated into:

1. Maximum Span (ms): The maximum allowed time difference between latest and earliest occurrences of events in the entire sequence,
2. Event-set Window size (ws): The maximum allowed time difference between latest and earliest occurrences of events in an event-set,
3. Maximum Gap (xg): The maximum allowed time difference between the latest occurrence of an event in an event-set and the earliest occurrence of an event in its immediately preceding event-set, and
4. Minimum Gap (ng): The minimum required time difference between the earliest occurrence of an event in an event-set and the latest occurrence of an event in its immediately preceding event-set.

An example of the discovered pattern <(A) (C, B) (D) (F, E.G)> is shown in the Figure 5.1. Hybrid-Apriori uses CDIST_O (described in Introduction) as support counting for interval-based sequential mining. CDIST_O considers the maximum number of all possible distinct occurrences of a sequence over all objects; that is, the number of all distinct timestamps present in the data for each object. The primary difference between the approach presented in this study (Hybrid-Apriori) for interval based sequential mining and traditional mining algorithms, lies in the use of time-intervals instead of timestamps. As an ordering criterion among sequences, intervals with the maximum interval-confidence are chosen between sequences with the same interval boundaries. Similarly, among sequences with the same start point and interval-confidence, the sequence with the earliest end point is chosen.

Thus, in Hybrid-Apriori, more importance is placed on sequences with greater interval-confidence and smaller lengths, thereby extracting the tightest sequential pattern.

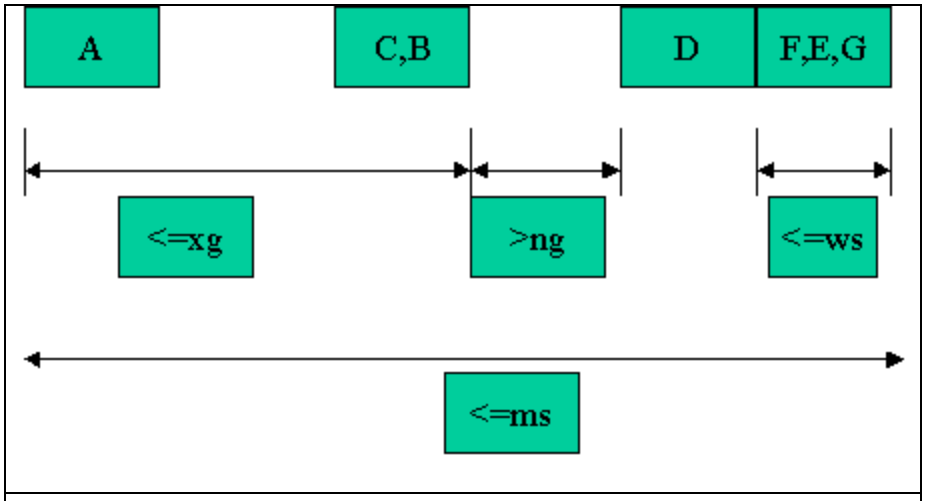

Figure 5.1: Timing Constraints

5.2 Characteristics of Traditional and Interval-Based Sequential Mining

This section outlines the differences between the traditional Apriori approaches used in traditional sequential mining techniques with the interval-based Apriori approach.

5.2.1 Apriori based Sequence Mining

Traditional mining algorithms primarily follow a standard sequence. From the entire dataset, given a window, all possible candidate item sets are identified. An efficient support counting method is used to prune the candidate sets with support below minimum support, and the remaining forms the frequent item sets. Pruning is based on the subset property to find the final frequent item sets for the current pass. The

current set of frequent items is used as the seed to form the frequent items of the next pass. This process continues as long as frequent item sets are generated in the following passes. In the Apriori approach, candidate and frequent item sets are represented as relations containing a set of attributes, each representing an item. In the k^{th} pass, the set of candidate itemsets C_k is generated from the frequent itemsets F_{k-1} (generated in the $(k-1)^{th}$ pass) as shown below:

```
Insert    into Ck
Select    I1.item1, … ,I1.itemk-1, I2.itemk-1
From      Fk-1 I1, Fk-1 I2
Where     I1.item1 = I2.item1 and
                :
          I1.item k-2 = I2.itemk-2 and
          I1.item k-1 < I2.itemk-1
```

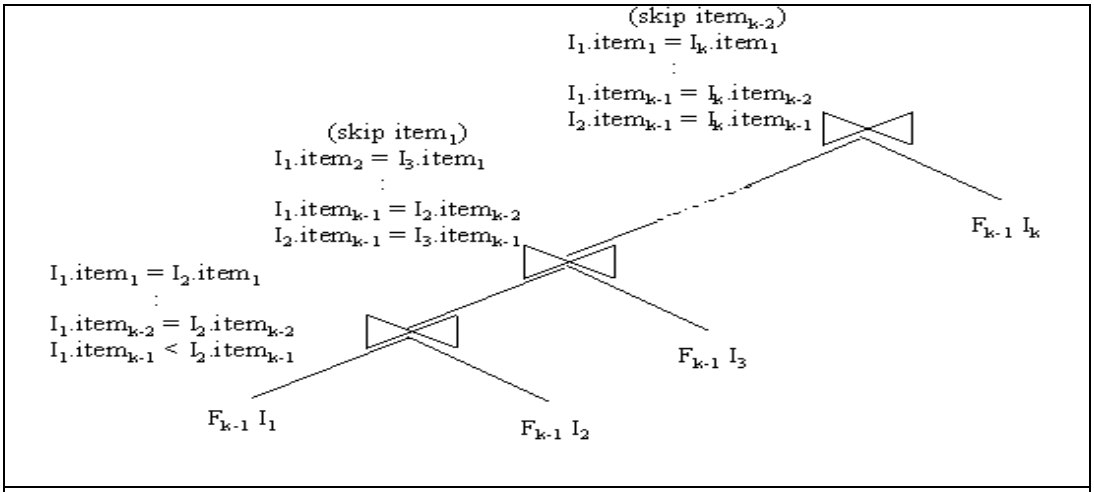

Figure 5.2: Candidate Generations for Any k

The number of candidate itemsets generated in each pass, by the above step is reduced by pruning out all itemsets $c \in C_k$ where some (k-1)-subsets (itemsets of length k-1) of c are not in F_{k-1}. This is based on the subset property that in order for an itemset

to be a frequent item, all subsets of that itemset have to be frequent. The tree diagram for this process is shown below in Figure 5.2.

5.2.2 Hybrid-Apriori based Sequence Mining

Hybrid-Apriori algorithm follows a similar sequence eliminating some of the steps on the way because of the presence of the SID intervals instead of the entire dataset of information. Application of a SID algorithm resulted in partitioning the dataset and extracting only the intervals with sufficient interval-confidence. Therefore the points, which would have been eliminated in the support counting phase, have already been eliminated before the start of sequential mining. Since the intervals for each event was found independently without using the sequential information, a statistical independence is assumed between events in the discovered patterns. They act as potential candidates including both positive and negative causality patterns. This approach has an advantage over the traditional mining approaches in its efficiency and lack of storage requirements. The actual causality can be found by using the potential patterns over the stream of future data, thereby eliminating the need to store the past data. In this process the validity, pattern-confidence and number of occurrences of the potential candidates can be extracted. On the other hand, if the application requires identifying the set of actual patterns in the data beforehand, making one pass over the input data can help eliminate patterns with events, which were discovered due to temporal proximity and not actual occurrence. Even with this extra computation, the hybrid-apriori is efficient as the approach makes use of partitioning, parallelization and makes only one pass over the entire dataset. Pattern-confidence replaces support

counting in the hybrid-apriori algorithm, which represents the average number of occurrences of the sequence within the interval. The pattern-confidence of a sequence within an interval is the product of the interval-confidence of its events within the same interval thereby assuming the events to be independent of each other. With frequently occurring patterns, pattern-confidence underestimates the actual probability of the events occurring together but retains its significance or order relative to the other patterns discovered. If interval-confidence >= 1, the event is considered certain to occur within the interval. Interval-confidence is always > 0, since an interval cannot exist without at least one occurrence in the history. Any interval-confidence between 0 and 1 provides a degree of certainty with which the event occurs within the interval. Therefore it can be said that events with interval-confidence >1 can be treated as certain events and can have their value as 1. The amount of over-occurrence within the interval is immaterial because pattern discovery relies on whether its events are certain to occur within the interval and if not their degree of certainty. For the events with interval-confidence between 0 and 1, the value gives the average number of occurrences within an interval over a period of time and therefore remains unchanged. The greater the interval-confidence, the greater is the chance of the event occurring within the interval. For example, when event A occurs within an interval (3-8) with interval-confidence of 0.9 and event B within (5-9) with interval-confidence of 0.8, the pattern-confidence of the occurrence of event AB within (3-9) is 0.72 (since both A and B are independent events). Instead of using K- copies of F1 for support counting, the confidence of an event set can be found by a two-way join of F_{k-1} and F_1. As the base case,

when k=2,

F_2.pattern-confidence= F_1.item1.confidence * F_1.item1*confidence,

where F_1.item1<F_1.item1

For K>2

$F_k = F_{k-1}$. pattern-confidence*F_1.item1.interval-confidence

where F_1.item1 < last item of F_{k-1} and F_1.item1.start-time and end-time is between start and end time of F_{k-1}.

The second property of WINDOW constraint automatically holds the subset property true because of which the pruning based on the subset property is not explicitly performed. As an example;

Let A (1,10), B (2,5), C(7,15), D(17,25) form the significant intervals generated from the SID[n-1] algorithm. The figures in the parenthesis indicate the intervals discovered for the events. Assuming a window of 10 units (maximum difference between start points of events), the first pass forms,

AB(1,10), AC(1,15), BC(2,15),CD(7,25)

The 3rd pass discovers ABC(1,15). Firstly if all subsets are above threshold pattern-confidence, ABC generated in the 3rd pass implies because of the window property that all of its subsets occur in the second pass. To explain it further A combined with B because B started within 10 units of start of A. A also combined with C because C started within 10 units of start of A. This automatically implies that B combines with C since B started after A. ABC was discovered in the 3rd pass, because AB and AC were found in the second pass. Secondly if we assume that the pattern-

confidence of sequence BC is below threshold, the pattern-confidence of the subset ABC automatically goes below threshold and can be pruned out automatically.

5.2.2.1. Interval Semantics

Given a window parameter, interval based mining provides two types of interval semantics which can be used to generate k-item set from the k-1 items. Most of the traditional sequential mining techniques deal with events that occur at a specific point of time. If the data point lies within window units of the start of the sequence, it is included in the generation of the maximal sequence. Dealing with an interval instead of a point results in two cases,

1) if the interval starts within window units of the start of the sequence, it can be included to form a maximal sequence

2) if the interval starts and ends within window units of the start of the sequence, it is included to form a maximal sequence.

Different domains require different semantics. Semantics-start (termed as semantics-s) generates all possible combinations of events, which occur within window units of its start time. Semantics-end (termed as semantics-e) on the other hand generates combinations with events, which complete within the window units of its start time. This automatically implies that events which occur with an interval greater than the window do not participate in the generation of maximal sequences of semantics-end. Using semantics-s results in many more sequences as compared to semantics-e as the latter is a subset of the former. With MavHome data, closely related interactions have more dependence on each other. Therefore semantics-e gives an agent more information

on the frequently occurring patterns. An agent can use the first event as the start indicator based on which the entire sequence can be automated.

5.3 Generation of Maximal Sequences

The algorithm is iterative in nature and generates greater length sequences in each subsequent pass. The terminating condition is either a stop level, which can be explicitly specified by the user or implicit when no more sequences are generated. The pseudo code of the algorithm is as follows:

```
1) Create table FKJOIN as
        Select * from F1
2) Insert into FKJOIN
        Select      item1,starttime+1,endtime+1,pattern-
        confidence
         From F1
3) For ( k=2; All Done !=1; k++)
{
     4)//Create all possible combinations based on a
     specific interval semantic
        create table MAXCONF as
           select a.item1,a.item2,…a.itemk-1,b.itemk-1,
           least(a.starttime,b.starttime),
           greatest(a.endtime,b.endtime),      b.pattern-
           confidence
           From F[k-1]  a* JOIN FKJOIN b
           where a.item1= b.item1 and ….
```

a.item$_{k-2}$=b.item$_{k-2}$ and a.item$_k$ <b.item$_k$ and b.starttime < a.start time + Window

5) //For the same interval, extract the interval with maximum confidence

create table MaxConf1 as

select item$_1$,. . . item$_k$, starttime, endtime, max(pattern-confidence)

from MAXCONF

group by item$_1$, …item$_k$, starttime, endtime

6)//For the same start time pick the end time associated with maximum confidence

create table MINENDTIME as

select item$_1$, … item$_k$, starttime, min(endtime) as endtime, pattern-confidence

from MAXCONF

group by item$_1$,.. item$_k$, starttime, pattern-confidence

7)//FMINENDTIME table contains patterns with maximum Pattern-confidence and the end time associated with it

create table FMINENDTIME as

select I2.item$_1$, … , I2.item$_k$, I2.dtstarttime, I1.dtendtime, I2.confidence

from

(select item$_1$, …, item$_k$, starttime, max(pattern_confidence) as confidence from MINENDTIME group by item$_1$,… item$_k$, starttime) I2 join MaxConf1 I1 on I1.item1=I2.item1 and … and I1.item$_k$ = I2.item$_k$ and I1.pattern-confidence=I2.pattern-confidence and I1.starttime=I2.starttime

8)//From a start point identify the smallest interval associated with the maximum confidence

create table F$_k$ as select item$_1$,…, item$_k$, starttime, min(endtime) , avg(pattern-confidence) from FMINENDTIME group by item$_1$, …, item$_k$, starttime

9) //Updates the pattern-confidence of the new FK sequence

*Update Fk pattern-confidence= pattern-confidence * (select max(confidence) from F1 where Fk.itemk-1=F1.item1 and F1.starttime>=Fk.starttime and F1.endtime <=FK.endtime)*

10) Eliminate Psuedo duplicates (will be explained below)

11) Create FKJOIN from F_k (same as above)

12) F_{k-1}Temps = Identify the sequences in F_{k-1} which are not subsets of any sequence in F_k

13) FreqItems$_k$= select item$_1$,…, item$_k$, #occurences

from F_k

group by item$_1$,…item$_k$

}

The algorithm uses the significant intervals produced by the SID algorithm as its collection of 1-item frequent sets (F1 table). 1-item sets combine to form 2-item sets that in turn combine to form 3-item sets and so on until the terminating condition is met.

Two F_{k-1} sequences combine in the k^{th} pass to form a k-length sequence if the following conditions are met:

- first k-2 items are the same
- a.item$_{k-1}$ < b.item$_{k-1}$
- b.item$_{k-1}$ occurs before a.item$_{k-1}$.startime + window.

The elements are arranged in a chronological order identified by *id* to ensure that events merge with other events that succeed them and the sequence grows in one direction only. All intervals produced by SID and its variants are associated with the default date of first of the current month along with the start and end times representing the tight boundaries. Because of the same date and the fact that time-series data have an

inherent wrap around property, the chronological order cannot be ascertained directly. An adjustment has to be made to convert this data into a format in which a definite order can be established. As an example assume an event with 23:00-23:30 can form a valid interval with another event at 00:00-00:20 or 23:20-00:00. Since an interval can have only one date, 23:30-00:00 has to be represented a little differently as it spans two days. Unless this is represented properly, max, min, least and greatest operators cannot be used on the data accurately. To accomplish the above FKJOIN is created, which is a copy of F_{k-1} with some additional records. The intervals, which start before window units of the smallest starting point of an interval from F_{k-1} are added to FKJOIN with their start and end dates incremented by 1. Using the above example if window is defined as 60 units, interval 00:00-00:20 will occur twice in FKJOIN, once with the same date as interval 23:00-23:30 and once with its date incremented by 1. Therefore in the k^{th} pass, interval 23:00-23:30 can merge with 00:00-00:30 of the next day to form a valid sequence with interval 23:00-00:30.

```
insert into FKJOIN (select item1, dtstarttime+1,
dtendtime+1
from Fk where
to_date(to_char(dtstarttime,'hh24:mi'),'hh24:mi') <=
to_date('hh:mm','hh24:mi'))
```

where ‘hh’ is the hour and ‘mm’ minutes associated with the window.

From the above algorithm, it is possible to identify the various aspects of interval based sequential approach, which differs from the traditional Apriori-K-way

join approach. The differences are: the use of pattern-confidence is used for support counting, lack of subset-based pruning, the different types of interval semantics available and to defer the usage of causality between the events to the end. In order to identify all possible frequent items sets at the end of the algorithm, two data structures are used: all frequent item sets from the last pass (F_k) and the frequent item sets from the prior passes, which could not join any further to form larger sequences. The latter items sets are stored in F_{k-1}TEMPS (Line 11) relation, partitioned on the length of the item set. The SQL formulation for the above is given on line 11 of the algorithm.

Line 4 generates all possible combinations of events by choosing events within the same block along with an added condition such that the new event added to the sequence has its start time within window units of the start time of the first interval. The new interval generated has the same start time as that of the starting interval. The end time however is the greater of the end times of the intervals being merged. For example when event A (3-5) merges with B (4-9), the sequence AB (3-9) is formed. In spite of the fact that SID reduces the number of frequent intervals generated, there might be more than one interval that covers a single area of high activity. These intervals form *almost* subsets of each other, differing from each other by a very small margin. As an example within an area of 10:00-10:15 , SID[n-1] might output {10:00-10:13, 10:01,10:14, 10:03-10:15} as the set of significant intervals. Line 4 creates duplicates when the above intervals are merged with another sequence. They are eliminated by keeping the sequence with the greatest confidence for the same event sets, start time and end time. These sequences are stored in MAXCONF1 (Line 5). As an example, when

A(1-8) joins with B (2-5) and B (5-8), two sequences namely AB (1-8) and AB (1-8) are created which may or may not be duplicates based on the confidence. The join performed on Line 4 also results in several rows with the same start time but varying end times. For example, when (1-6) joins with B (2-5) and B (5-8), two sequences are created viz. AB (1-6) and AB (1-8). During such a situation, the sequence with the maximum confidence is chosen because this sequence has greater probability of being involved in further passes. The SQL formulation for the above is given in Line 7. Line 8 picks the smallest length sequence with the greatest confidence within a particular event set and creates the sequences associated with the next pass i.e. F_k and line 9 updates the confidences of the newly formed intervals.

For example, when an event [ABC interval: (3-8), pattern-confidence: 9] merges with another [ABD interval: (4-8), pattern-confidence: 8], the pattern-confidence of the newly formed sequence [ABCD interval: (3-8) c: 8] is initially the same as that of the latter sequence. Line 9 updates it correctly by ABCD.pc= 0.8* C.ic where 0.8 gives us the pattern-confidence of A, B and D events together.

Line 11 creates FKJOIN for the subsequent pass, which is used to find frequent item sets for the next pass. Line 12 identifies the sequences in the prior pass, which failed to enter the frequent itemset of the current pass and stores them in F_{k-1}temps. In order to identify the set of sequences, which did not enter the next pass, the following measures are taken:

1) Get all sequences from Fk-1 whose item1..itemk-1 are the same as item1…itemk-2 of F_k that is identify all subsequences such as ABC, ABD in F_3 which were responsible for ABCD in F_4

2) Delete the above sequences from the list of probable candidates in FkTemp[k-1].

3) Perform the above operation for F_{k-1}.item1=F_k.item to delete any remaining subsets which start with the same event as $F_{k-1.}$ In order to remove the subset, i.e. the sequence F_{k-1} of F_k such that $F_{k-1.}$ item1= F_k .item2 and $F_{k-1.}$ item2= F_k .item3 and $F_{k-1.}$ itemk-1= F_k .itemk, the one-step computation given below is executed. As an example for a sequence ABCDE, subset BCDE is eliminated by the one-step Cartesian product given below. Currently in addition to the above subsets, all subsets with the same item1 are also removed by the below statement. Performing a Cartesian product at this stage or a corresponding join, does not affect performance since most of the sequences have been eliminated.

The reason for the step-wise operation is that very few sequences are left behind in each pass and all subsets of k-length sequence within an interval have to be eliminated. The above method is more efficient compared to a one-step computation given below as it performs a Cartesian product:

```
insert into FKTEMPS_23
     select       I1.item1,       I1.item2,       I1.item3,
     I1.dtstarttime,I1.dtendtime,I1.confidence
     from F_23 I1  where NOT EXISTS
```

```
( SELECT * FROM F_24 I2
     WHERE
     I1.item1 in ( I2.item1, I2.item2, I2.item3,
     I2.item4) AND
     I1.item2 in ( I2.item1, I2.item2, I2.item3,
     I2.item4)
     AND I1.item3 in ( I2.item1, I2.item2,
     I2.item3, I2.item4)
     AND I1.dtstarttime BETWEEN I2.dtstarttime
     AND I2.dtENDtime
     AND I1.dtendtime BETWEEN I2.dtstarttime AND
     I2.dtENDtime)
```

In addition to the frequent sequences occurring at specific points, the identification of sequences with maximum interactions between devices is also achievable. This implies that there may be several instances of a specific pattern, each occurring at different time intervals. Line 13 identifies all such patterns and their number of occurrences. This information is as valuable as the exact occurrences of each pattern, as the start event of the high frequency patterns can be used as triggers to automate the sequence. The remainder of the section discusses the formation and elimination of pseudo-duplicates of line 10. Due to the use of least and greatest functions, pseudo-duplicates, as given below, can be generated:

Table 5.1: Sample Input for formation of Pseudo-duplicates

ID	Item1	Item2	Start time	End time
1	A	B	1	5
2	A	C	2	5
3	A	B	1	4

Table 5.2: Pseudo-Duplicates

ID	Item1	Item2	Item3	Start time	End time
1	A	B	C	1	5
2	A	C	B	1	5

Let Table 5.1 represent a set of sequences created in F2. F3 produced from F2 in Table 5.2 consists of two patterns that represent the same interval with the same set of events with the order of items interchanged. These are therefore classified as pseudo-duplicates as only one of them needs to be retained. Since all characteristics except for the order of occurrence of the items are same, any one of the patterns can be retained from this set. This is achieved in two steps:

- The F_K table is converted into a vertical format of Id and Item
- K-copies of this table are joined in a specific order to produce all items in the same order

For the above example the following sql statements eliminate all pseudo-duplicates. Let us assume items contains the items and Id of F_23.

```
create table single_items as
select id,item from items order by item,id
//Form 3-item sets in ascending order
create table order_items as
```

```
     (select I1.Id,I1.item as item1 , I2.item as item2,
     I3.item as item3
     from single_items I1 join single_items I2 on
     I1.id=I2.id
               join single_items I3 on I3.id = I2.id
     where  I1.item < I2.item  and I2.item < I3.item)
//Get the corresponding start and endtimes associated with
each pattern
create table FreqTemps as
     select I2.*,dtstarttime, dtendtime, confidence
     from F_23 I1 join order_items I2 on I1.id=I2.id
drop table F_23
//Create F_23 again, without any duplicates.
create table F_23 as
     select rownum as ID, X.*
     from
          (select       item1,       item2,       item3,
          dtstarttime,dtendtime,           max(confidence)
          confidence
          from FreqTemps
```

```
group by item1, item2, item3,
dtstarttime,dtendtime order by
dtstarttime,dtendtime) X
```

Depending on the interval semantics chosen, the SQL query has following selection condition added to its already existing selection conditions.

b.endtime < = a.starttime + window

Classifying the sequences as parallel and serial can be done after the discovery of all frequent item sets. The pattern start and end times can be used to join with F1 intervals to identify the relative order of the events within the sequence. This allows for the discovery of parallel and serial episodes in one pass, instead of maintaining two algorithms for their discovery [3].

As a quick note on implementation, MaximalSequence.java and GeneraModule.java form the heart of hybrid-apriori. MaximalSequence is the class, which executes the above steps in an iterative manner. GeneralModule class provides the sql statements needed in each pass in order to dynamically run the algorithm for any number of iterations.

5.4 Experiment Results

The results produced by the hybrid apriori algorithm presented in this thesis is compared with Episode Discovery [ED] algorithm [19] to verify and evaluate the sequences generated. ED algorithm has the ability to discover events with different periodicity. Candidate sets in Ed are represented as a combination of sets of events, sets of episodes and a significance value. $I=<\{e_1,e_2,e_3..e_j\},\{P_1,P_2..P_k\},V>$, where each event

e_j has an occurrence in each episode P_k. V is the significance value of the candidate item set with respect to the frequency, length and regularity of the episode. ED was run with a fixed window of 15 units and an added parameter of number of occurrences within the window (window width) of 5. In order to simulate a window best, hybrid-apriori was run with a measure of 4 ensuring significant intervals of length <= interval length specified are produced when mined for patterns with semantics-e. Semantics-e performs better for comparison with ED, because it prevents the pattern-length from outgrowing the window, thereby producing comparable episodes with respect to Ed. Since pattern discovery can be split into two independent parts, SID/hybrid-apriori can be independently re-run with modified parameters. Data collection during the month of March for MavHome was used to identify all possible patterns and interactions between device operations. ED did not discover any daily patterns but discovered 7 weekly patterns with a total of 40 occurrences among the 7 patterns. SID with a measure of 4, interval-confidence=0.7, interval-length =15 and interval semantics-e also did not produce any daily patterns. SID[1]/Naïve discovered 50 significant intervals when run with a weekly periodicity. Furthermore comparison of intervals produced with different approaches on March data did not reveal any difference in the intervals produced. However with small datasets SID[1]/Naïve is the most suitable approach since it is imperative that all the intervals are enumerated. The pattern discovery algorithm discovered 7 patterns and 23 occurrences missing two patterns discovered by Ed. The differences between hybrid-apriori approach and ED were due to differences in the input parameters and the inability to form an exact match of the input parameters.

SID[1] allowed an event to form a significant interval only if it occurred in at least 3 weeks of march. ED on the other hand allowed events that occurred twice a month to form a part of the pattern. Reducing the threshold interval-confidence of SID[1] increased the number of significant intervals to 146 and identified the patterns discovered by ED. They however had very low pattern-confidence since all the events forming the sequence had around 0.5 of interval-confidence to start with. Mining on the same dataset using semantics-s (semantics 1) produced greater number of patterns and increased the pattern length correspondingly. Figure 5.3 below illustrates the number of intervals generated in each pas as well as the final output intervals from each pass using semantics-e. Figure 5.4 on the other hand compares the number of output intervals and intervals generated in each pass using semantics-s. F_k where k is the last pass and all Fktemps from $Fktemps_{k-1}$ to $Fktemps_1$ form the output where as F1 to Fk form the total number of intervals generated in each pass. It can be seen that semantics-e produced more number of 2-length patterns, which failed to grow to 3-length as compared to patterns produced by semantics-s. As seen in Figure 5.4 and Figure 5.3, semantics-s generated large number of patterns, but increased the pattern-length correspondingly.

A six-month synthetic data with predefined number of daily and weekly patterns was used to further compare the two algorithms. Ed produced 9 daily and 4 weekly events with a total of 41 occurrences. SID[n-1] with Hybrid Apriori and semantics-e produced 8 daily events but missed two events due to the strict merge criteria (Figure 4.1) described in chapter 4. SID[n-1] however when run with weights of 0.9 produced

intervals for all events which when mined for patterns, extracted all the all the patterns defined in the dataset.

The events were missed because the number of points for the event missed were very few, allowing for the existence of only one interval in that area. However as seen in Chapter 4, with large datasets, SID[n-1] performs as well as Naïve with regards to coverage.

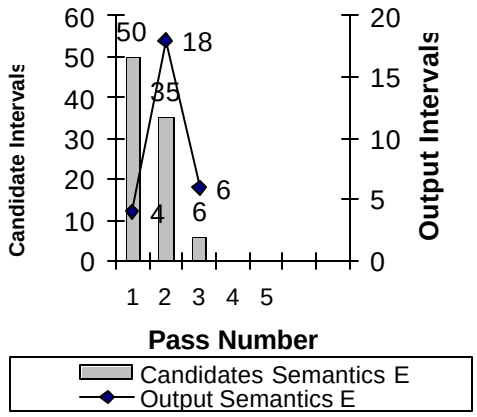

Figure 5.3: Patterns generated in each pass using semantics-e (weekly)

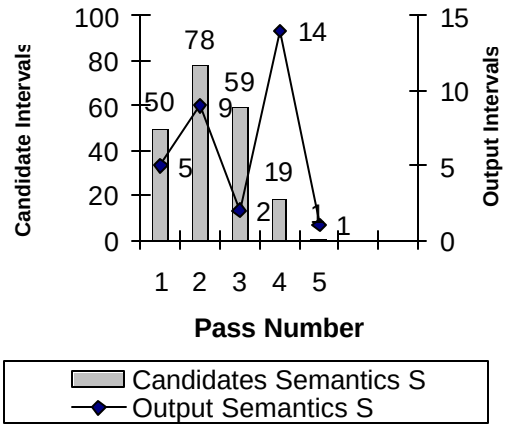

Figure 5.4: Patterns generated by Semantics-s (weekly)

Weekly episodes discovered by SID[n-1] in its strict form with weights=1 included all the daily episodes in addition to two events discovered which were purely weekly. Running SID[n-1] with weights=0.9 produced 2 weekly events missed from before. The algorithm was run on the dataset with both the semantics to compare on the number of intervals as well as the interval length when run with measure=4, interval-length=15 and interval-confidence=0.7. As seen with the MARCH dataset, semantics-s produced much more information and greater length patterns as compared to semantics-

e. Figure 5.6 illustrates the comparison in pattern-length between the two semantics when run with six-month dataset on a weekly periodicity.

In addition, the pattern-discovery algorithm was run with semantics-s and SID[n-1] approach was run with a measure type of 2 and interval confidence =0.7 alone with no restriction on the interval length. This resulted in greater number of intervals as compared to running with measure of 4 and an added constraint of interval-length=15.

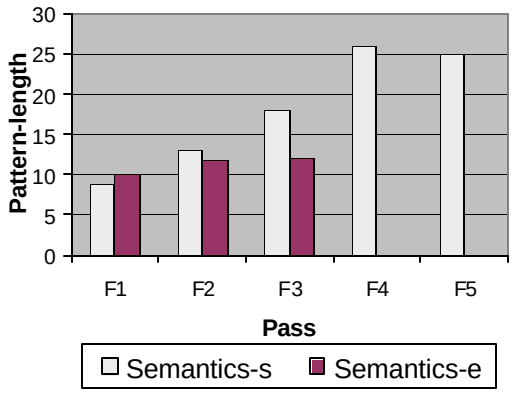

Figure 5.5: Pattern-lengths by semantics 1 & 2 in march (weekly)

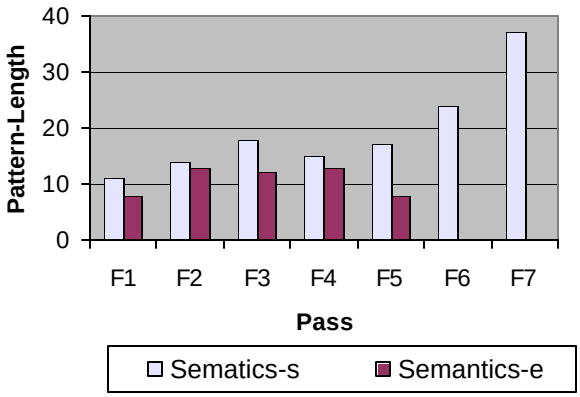

Figure 5.6: Pattern lengths by semantics 1 & 2 for 6-month synthetic data (weekly)

The number of patterns occurrences discovered with semantics-s (semantics 1) was 422 with a maximum length of 9 item-sets, whereas as seen from Figure 5.6 semantics-e and semantics-s with measure=4 produced maximum of 81 different patterns occurrences with the maximum length of 7-item set. We can thus conclude that the choice of interval semantics depends on the amount of domain knowledge available as well as the number of significant intervals generated. It can be inferred from above that when dealing with large number of intervals, semantics-s produces more number of

patterns as compared to semantics-e. Moreover on account of lack of any restriction on the end time, the pattern length increases as well. For such a scenario, mining for patterns using semantics-e produces better results as can be seen with the synthetic six-month and MARCH dataset.

In order to compare the time taken and the number of patterns discovered from the 3 months of MavHome data, we ran the dataset through SID[n-1] with the following parameters: Measure=4, interval-confidence=0.9 and interval-length=15 Period=Daily. Response time was measured as the time taken over an average of four runs over the same dataset with the same parameters. All the experiments were run with the algorithm producing a maximum of 10-item-length patterns. Figure 5.7 illustrates the time taken during the pattern-discovery alone between the 1-3 months of dataset used.

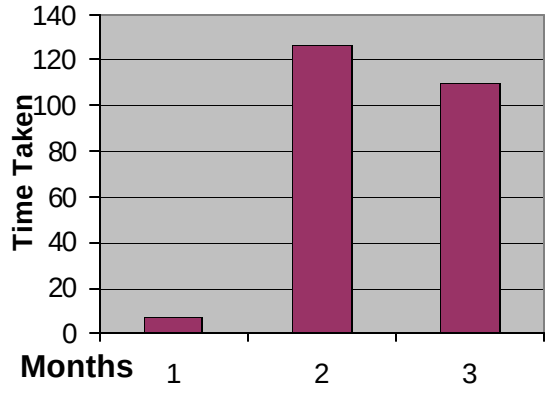

Figure 5.7: Time taken for hybrid-apriori from 1-3 months with daily periodicity

Months	1	2	3
# of Rows	46000	140000	201000
INPUT	2181	3686	3526
F1	33	28	14
F2	56	35	25
F3	66	32	44
F4	97	46	46
F5	107	23	64
F6	123	56	55
F7	121	95	103
F8	89	111	151
F9	43	169	119
F10	7	9710	15853

Figure 5.8: Number of patterns produced for 1-3 months with daily periodicity

Figure 5.8 enumerates the number of output intervals produced in each pass i.e. FkTemps1, FkTemps2,..,F_10. For the month of June, the intervals were fewer in number and the dataset could be clustered into distinct regions. This was the reason for

the small amount of time taken and the number of intervals generated in each pass was few. The maximum size of the intermediate pass was 20324. When mined for 2 months, the number of intervals generated increased as seen from INPUT row of Figure 5.8. Moreover the number of distinct overall clusters reduced, thereby increasing the number of merges that can be performed in each pass. The maximum size of the intermediate pass was 173711, almost 10 times as that of June data. By three months, the data consistency improved a touch, thereby producing slightly fewer intervals and taking similar time as compared to the two-month run.

5.5 Writing Log File

Data mining is a time-consuming process and at times it happens that for certain mining configurations, mining a given dataset may take 10 to 15 hrs or even more. In order to compare the performances of SID approaches and the sequential mining algorithm described above, after a given time limit, if the algorithm does not complete, the mining process has to be killed. Also for the purpose of studying these algorithms, it is necessary to know of their progress in terms of the number of iterations completed, cardinality of the intermediate results etc. Hence it is very important to collect the time taken at each step of the algorithm and produce a log file containing pertinent information. This log file can then be processed to generate the useful information such as the number of passes completed, time taken for each pass, intermediate relations generated and cardinality of each of them, even if the mining process is killed before it completes. The algorithm generates two log files for interval discovery and one for the sequential mining.

Ouput_interval log and event_name+'periodicity' log, are written at the end of interval discovery process. Output_Interval log contains the time stamp of when a particular step of each event in the approach started and completed. Event_name log contains the time taken for each pass along with the results generated in each pass, which is specific for each event. Contents of the Output_Interval log file are given below.

Window :60

Measure :4

Interval-Confidence :0.8

Interval-Length :15

NumDays:182

Interval Semantics :1

Maximum Number of threads 10

Approach used :1

The Program started at Tue Sep 30 12:44:12 CDT 2003

Period :Daily

Finished batch for BathRm2Light and Off and of size 126

The Program started at Tue Sep 30 12:44:13 CDT 2003 for BathRm2LightOff

End of Find combinations and Start of Repeat Combinations at Tue Sep 30 12:44:13

CDT 2003 for BathRm2LightOff

Started All Combinations Approach for BathRm2LightOff

Finished Repeat Combinations for BathRm2LightOff with 16 passes at Tue Sep 30 12

:44:14 CDT 2003

Finished Identify Clusters for BathRm2LightOff at Tue Sep 30 12:44:14 CDT 2003

The above example illustrates the messages output for one event BathRm2Light Off. All events have a similar set of messages associated with their execution. This gives an overall summary of information on the intervals produced for each device. To get detailed information on the time taken for each pass and the number of intervals created in each pass, the event_name+periodicity log can be viewed. For the BathRm2Light Off, the file BathRm2LightOff.out for daily periodicity or BathRm2LightOff_Monday or BathRm2LightOff_Tuesday etc for weekly periodicity is created. We now take a look at BathRm2LightOff.out.

Numdays is: 182

Interval-confidence is: 0.8

Interval length :15

Measure: 4

Window is: 60

Type: T

Approach: 1

Complete Debug: 0

The set of points :

00:19 1

01:09 2

01:10 1

01:11 2

01:12 5

01:13 8

…

End of Find combinations and Start of Repeat Combinations 1 pass at Tue Sep 30 1

2:44:13 CDT 2003

Start of Repeat Combinations2 pass at Tue Sep 30 12:44:13 CDT 2003

Start of Repeat Combinations3 pass at Tue Sep 30 12:44:13 CDT 2003

…

Start of Repeat Combinations15 pass at Tue Sep 30 12:44:14 CDT 2003

End of Repeat Combinations is at Tue Sep 30 12:44:14 CDT 2003

The Significant intervals found before forming clusters are

//Set of Significant intervals created

19:52 20:04 12.750 12 153 0.841

19:51 20:03 12.667 12 152 0.835

19:50 20:02 12.667 12 152 0.835

Time Taken to find all Clusters is 1 Seconds

End of Identify Clusters is at Tue Sep 30 12:44:14 CDT 2003

The Cluster intervals found are

19:50 20:02 12.667 12 152 0.835

19:51 20:03 12.667 12 152 0.835

19:52 20:04 12.750 12 153 0.841

End of Repeat Program is at Tue Sep 30 12:44:18 CDT 2003

Depending on the debug parameter set in configuration file, the amount of information logged changes. When debug is set to 'no', the intervals created in each pass of repeat combinations phase or the cluster groups are not printed as shown above. Only the final set of significant and cluster intervals are printed in full.

To see the results at the end of every step the debug switch has to be set to 'yes' in the configuration file.

Maxseq.out is the log file associated with generation of maximal sequences. This log file prints the SQL steps completed and ends with the cardinality of each relation created during the process. Depending on the interval semantics chosen, standard table names are associated with each pass. The convention for naming the tables is as follows:

Frequent items set created at each pass are stored in F_iK where i represents the interval semantics chosen. As an example, with interval semantics=2 (semantics-e), the frequent 2-item sets are stored in F_22 and frequent 5-item sets are stored in F_25. All frequent k-item sets which did not merge further to form k+1 item sets are stored in Fktemps_ik where i represents the interval semantics. As an example, using interval semantics-e (semantics 2), all frequent 2-item sets that did not extend further to form 3-item sets are stored in Fktemps_22. The frequent k-item sets, which always occur together irrespective of the time, are stored in FreqItems_ik, using the same notation as above. Sample of Maxseq.out logfile is given below:

Completed: DROP TABLE F_21

Completed: DROP TABLE F_22

Completed: DROP TABLE F_23

Completed: DROP TABLE F_24

Completed: DROP TABLE FKTEMPS_23

Completed: DROP TABLE FKTEMPS_24

Completed: DROP TABLE FREQITEMS_22

Completed: DROP TABLE FREQITEMS_23

Completed: DROP TABLE FREQITEMS_24

Completed: create table F_21(ID int, item1 varchar(20), dtstarttime date, dtendTime date, confidence float)

Completed: insert into F_21 select rownum as ID, X.* from (select txtdeviceId||txtStatus as item1 ,dtstarttime, dtendtime, numconfidence from TBSIGINTERVAL order by dtstarttime) X

Completed: commit

Completed: update F_21 set confidence=1 where confidence >1

Completed: delete from F_21 where (dtendtime-dtstarttime)*1440 > 15

Completed: drop table FKJOIN

Completed: create table FKJOIN as (select * from F_21)

Completed: insert into FKJOIN (select ID, item1, dtstarttime+1, dtendtime+1, confidence from F_21 where to_date(to_char(dtstarttime,'hh24:mi'),'hh24:mi') <= to_date('0:15','hh24:mi'))

Completed: drop table F_21ALL

Completed: create table F_21ALL as select * from FKJOIN

Next Pass Fri Sep 19 10:43:30 CDT 2003

Completed:

create table TEMP(item1 varchar(20) not null, item2 varchar(20) not null, dtstarttime date not null , dtendtime date not null, confidence float, primary key(item1 , item2, dtstarttime, dtendtime,confidence))

Completed:

insert into TEMP select distinct I1.item1, I2.item1, least(I1.dtstarttime,I2.dtstarttime) , greatest(I1.dtendtime, I2.dtendtime),I2.confidence from F_21 I1, FKJOIN I2 where I1.ID < I2.ID and I1.item1 <> I2.item1 AND I2.dtstarttime >= I1.dtstarttime AND I2.dtendtime <= I1.dtstarttime +15/1440

Completed:

drop table TEMP1

Completed:

create table temp1 as select item1, item2 ,dtstarttime,dtendtime,max(confidence) as confidence from temp group by item1, item2, dtstarttime,dtendtime

Completed:

drop table MINENDTIME

Completed:

create table MINENDTIME as select item1, item2 ,dtstarttime,min(dtendtime) as endtime,confidence from TEMP group by item1, item2,dtstarttime,confidence

Completed:

create table TEMP2 as select I2.item1, I2.item2, I2.dtstarttime, I1.dtendtime,I2.confidence from (select item1, item2, dtstarttime,max(confidence) as confidence from MINENDTIME group by item1, item2, dtstarttime) I2 join Temp1 I1 on I1.item1=I2.item1 and I1.item2 = I2.item2 and I1.confidence=I2.confidence and I1.dtstarttime=I2.dtstarttime

Completed:

create table F_22 as select rownum as ID, X.* from (select item1, item2, dtstarttime, min(dtendtime) dtendtime,avg(confidence) confidence from TEMP2 group by item1, item2, dtstarttime order by dtstarttime) X

Completed:

update F_22 I2 set I2.confidence=I2.confidence * (select max(I1.confidence) from F_21ALL I1 where I1.item1=I2.item1 and I1.dtstarttime >= I2.dtstarttime and I1.dtendtime<= I2.dtendtime)

Completed:

create table items (Id int, item varchar(30))

Completed:

insert into items(Id,item) select id,item1 as item from F_22

Completed:

insert into items(Id,item) select id,item2 as item from F_22

Completed:

create table single_items as select id,item from items order by item,id

Completed:

create table order_items as (select I1.Id,I1.item as item1 , I2.item as item2 from single_items I1 join single_items I2 on I1.id=I2.id where I1.item < I2.item)

Completed:

delete from order_items where item1=item2

Completed:

create table FreqTemps as select I2.*,dtstarttime,dtendtime,confidence from F_22 I1 join order_items I2 on I1.id=I2.id

Completed:

drop table F_22

Completed:

create table F_22 as select rownum as ID, X.* from(select item1, item2, dtstarttime,dtendtime, max(confidence) confidence from FreqTemps group by item1, item2, dtstarttime,dtendtime order by dtstarttime,dtendtime) X

Completed:

drop table FKJOIN

Completed:

create table FKJOIN as (select * from F_22)

Completed:

insert into FKJOIN (select ID,item1, item2, dtstarttime+1, dtendtime+1,confidence from F_22 where to_date(to_char(dtstarttime,'hh24:mi'),'hh24:mi') <= to_date('0:15','hh24:mi'))

Completed:

create table FREQITEMS_22(item1 varchar(20) not null, item2 varchar(20) not null, Count int, primary key(item1, item2))

Completed:

insert into FREQITEMS_22(select item1, item2, count(*) as Count from F_22 group by item1, item2)

Next Pass Fri Sep 19 10:43:35 CDT 2003

…

The Program started at Fri Sep 19 10:43:33 CDT 2003

, F_21:50

, FKJOIN :50

, 2 pass ended at Fri Sep 19 10:43:35 CDT 2003

, FKJOIN :55

, F_22:55

, FKTEMPS_21:0

, FREQITEMS_21:0

, 3 pass ended at Fri Sep 19 10:43:36 CDT 2003

, FKJOIN :19

, F_23:19

, FKTEMPS_22:10

, FREQITEMS_22:1

, 4 pass ended at Fri Sep 19 10:43:37 CDT 2003

, FKJOIN :3

, F_24:3

, FKTEMPS_23:6

, FREQITEMS_23:6

Finally a vector storing the time taken for each pass and the cardinality of each relation created by the program is printed.

5.6 Conclusions

From the above it can be concluded that the efficiency and scalability of the hybrid-apriori algorithm depends heavily on SID and its variants. Better the quality of the intervals produced by SID, better is the pattern-discovery. Since the output generated between the two semantics greatly differs in quantity, semantics-s can be used to run with representative data sets so as to gather more information on the average pattern-length, size and so on. The process can then be run with semantics-e on the actual dataset, by setting parameters such as stop-level and window-length appropriately. It can be summarized that in addition to the output patterns and their time of occurrences, the above algorithm also produce the frequently occurring patterns irrespective of their time of occurrence from FreqItems relation. As an example, LivRmLamp1On, ComputerOn are the most frequent 2-item patterns, occurring several times in a day in the six-month dataset. In addition, the number of occurrences of each pattern within the interval can be estimated by finding its maximum and minimum number of occurrences. Taking the minimum of the occurrences of each of its event within the interval from Tbsiginterval (set of significant intervals) gives an upper

bound on number of occurrences of the pattern. The lower bound on the number of occurrences of a pattern AB within a time interval can be obtained as :

0 if the pattern-confidence of A or B <0.5

pc(A)+pc(B)-1 if pattern-confidence of A and B >0.5

For patterns with k-length >2, the approach can be recursively performed by taking the minimum of the number of occurrences of its 'k' k-1 length subsequences if the pattern-confidence of all its events is greater than 0.5.

Current implementation of the hybrid-apriori retains only those frequent patterns ($FreqItems_k$) from each pass, which do not have an instance in the following pass. However this can be modified to obtain the most frequent 2-item pattern, 3-item pattern and so on irrespective of the further growth. The most frequently occurring patterns in each pass can be arranged as per their relative order of occurrence to act as triggers for automating device interactions. Currently this is not performed because the emphasis is more on extending the frequent patterns to their maximum length.

CHAPTER 6

CONCLUSION AND FUTURE WORK

With large numerical and time series data, events occur with a high degree of certainty not at specific points but within tight intervals (set of points). This implies a need for the discovery of such intervals, which are representative of points of occurrence of events, based on no, little or reasonable knowledge of the dataset. The SID algorithmic variants proposed in this thesis succeed in discovering the set of tightest intervals based on user input for each event. A suite of algorithms has been formulated from which the best approach satisfying the input and output requirements of the dataset can be chosen. The intervals discovered are used to identify clusters of interest. Our work presents a different and novel approach towards discovery of intervals and clusters. The biggest drawback to traditional clustering algorithms has been the input parameters, which are difficult to set due to lack of knowledge on the nature of the dataset. The number of clusters generated as output from our work can be used as an upper bound and its characteristics such as length and density can be used as input to traditional clustering algorithms for improved results. Currently the implementation identifies overlapping clusters thereby providing a maximum limit on the number of clusters present in the data set. This set of clusters can be further changed on domain requirements to be disjoint in nature. The algorithms are easily extensible to domain specific features. Identification of disjoint or overlapping clusters, use of Naïve

or SID variants, patterns with different periodicities, reduction in response times based on system characteristics are all parameters, which can be modified as required. The approach is flexible enough to work with none, few and several domain-specific inputs provided by the user. The algorithms provide a rich set of tools to explore and understand the data set.

The SQL formulation of the interval-based sequential mining proposed in this thesis does not depend on the window parameter as heavily as traditional approaches. The patterns discovered using semantics-s identifies all sequences within window units irrespective of the lengths of the event sets, thereby discovering patterns with lengths greater than a given window size. In addition to patterns and their occurrences, the algorithm identifies the set of potential sequences/subsequences that can occur as a group irrespective of the time of occurrence. Identification of the sequences that actually occur as a group can be performed by looking either at the past or future data. Identification of parallel and serial episodes is performed together by determining the complete set of frequent episodes followed by evaluating the relative order of the event/event-set within every sequence. Joining the set of frequent sequences with the set of significant intervals can discover the relative order within a sequence. Bounds on the number of occurrences of each pattern within an interval can also be found. Efficiency in terms of response time and reduction in the amount of storage needed to discover patterns are the advantages of the proposed approach.

As can be seen from the Hybrid-Apriori experiments, the scalability of the algorithm depends heavily on the intervals created using SID approaches. Currently if

there exists 10 tight intervals within a region, Naïve enumerates all of them where as SID[n-1] enumerates around three. Reducing this to one, thereby identifying the best interval in the region, taking into account intervals produced for other devices so as not to lose a frequent pattern, can be an excellent extension to this work.

In addition to the above, SID[n-1] and its alternatives can also be implemented as a pure SQL based algorithm to improve scalability and efficiency. Incremental versions of the interval and pattern discovery algorithms will provide better scalability with large and continuous data (data streams). Currently the algorithm when run on the higher level of periodicity discovers the patterns with lower level periodicities automatically. As an example, when run with weekly periodicity, the algorithm discovers daily patterns as well. An additional layer can be added to this, so as to seamlessly drill down and retrieve patterns with all periodicities when run at the highest level.

REFERENCES

1. Thuraisingham, B., *A Primer for Understanding and Applying Data Mining.* IEEE, 2000. **Vol. 2, No.1**: p. 28-31.
2. Thomas, S., *Architectures and optimizations for integrating Data Mining algorithms with Database Systems*, in *CSE*. 1998, University of Florida: Gainesville.
3. Mannila, H., H. Toivonen, and I. Verkamo. *Discovering Frequent Episodes in Sequences*. in *Proc of the 1st Intl. Conference on Knowledge Discovery and Data Mining*. 1995. Montreal, Canada.
4. Srikant, R. and R. Agrawal. *Mining Sequential Patterns:Generalizations and Performance Improvements.* in *In 5th Intl. Conf. Extending Database Technology*. 1995. Avignon, France: IBM.
5. Bell, T.C., J.C. Cleary, and I.H. Witten, eds. *Text Compression*. Advance Reference. 1990, Prentice Hall.
6. Bruce L Bowerman, R.T.O.C., *Time Series Forecasting*. Second Edition ed. 1990: PWS Publishers. 25-120.
7. Gilchrist, W., *Statistical Forecasting*. 1976: John Wiley and Sons. 115-148,77-90.

8. Joshi, M., G. Karypis, and V. Kumar, *A Universal Formulation of Sequential Patterns*. 1999, Department of Computer Science, University of Minnesota: Minnesota. p. 20.

9. Agarawal, R. and R. Srikant. *Mining Sequential Patterns*. in *Proc. of 11th Int'l Conference on Data Engineering*. 1995. Taipei, Taiwan.

10. Zaki, M.J., *SPADE:An Effecient Algorithm for Mining Frequent Sequences.* Machine Learning Journal,special issue on Unsupervised Learning (Doug Fisher, ed.), 2001. **42 No.1/2**: p. 31-60.

11. Zaki, M.J. *Sequence Mining in Categorical Domains: Incorporating Constraints.* in *9th Int'l Conference on Information and Knowledge Management*. 2000. Washington DC.

12. B. Ozden, S. Ramaswamy, and A. Silberschatz. *Cyclic Association Rules*. in *Proceedings of the IEEE International Conference on Data Engineering*. 1998. Orlando, FL.

13. Han, J., W. Gong, and Y. Yin. *Segment-Wise Periodic Patterns in Time Related Database*. in *Proc 1998 Int'l Conference on Knowledge Discovery and Data Mining*. 1998. New York City: AAAI Press.

14. Das, S.K., et al., *The Role of Prediction Algorithms in the MavHome Smart Home Architecture,* in *IEEE Wireless Communications Communications Special Issue on Smart Homes*. 2002. p. 77-84.

15. Kanungo, T., et al., *An Efficient k-Means Clustering Algorithm: Analysis and Implementation*, in *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*. 2002. p. 881-892.

16. Han, R.N.a.J., *Clarans: A method for clustering objects for spatial data mining*, in *IEEE Transactions on Knowledge and Data Engineering*. 2002. p. 1003--1016.

17. A. Hinneburg and D. A. Keim. *An Efficient Approach to Clustering in Large Multimedia Databases with Noise*. in *Proc. 4rd Int. Conf. on Knowledge Discovery and Data Mining*. 1998. New York.

18. Ester, M., et al. *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*. in *In Proceeding of the 2nd International Conference on Knowledge Discovery and Data Mining*. 1996. Portland, OR.

19. D. J. Cook, M.Y., E. Heierman, K. Gopalratnam, S. Rao, A. Litvin, and F. Khawaja. *MavHome: An Agent-Based Smart Home.* in *to appear in Proceedings of the Conference on Pervasive Computing*. 2003.

20. Anwar, E., L. Maugis, and S. Chakravarthy, *A New Perspective on Rule Support for Object-Oriented Databases*, in *1993 ACM SIGMOD Conf. on Management of Data.* 1993: Washington D.C. p. 99-108.

21. Chakravarthy, S., et al. *ECA Rule Integration into an OODBMS: Architecture and Implementation*. in *ICDE*. 1995.

22. Chakravarthy, S., *Early Active Databases: {A} Capsule Summary.* 1995. **7**(6): p. 1008--1011.

23. Chakravarthy, A. and E. Anwar, *{Exploiting Active Database Paradigm For Supporting Flexible Transaction Models}.* 1995, University of Florida Computer and Information Science and Engineering Department.

24. Datar, B.M., R. Motwani, and J. Widom. *Models and issues in data stream systems.* in *In Proc. of the 2002 ACM Symp. on Principles of Database Systems.* 2002.

25. Madden, S. and M.J. Franklin. *Fjording the Stream: An Architecture for Queries over Streaming Sensor Data.* in *In Proceedings of the 18th International Conference on Data Engineering (ICDE).* 2002. San Jose, CA.

26. Carney, D., et al. *Monitoring Streams.* in *Monitoring Streams - A New Class of Data Management Applications*. 2002. Hong Kong, China.

27. Chen, J., et al. *NiagraCQ: A scalable continuous query system for internet databases*. in *In Proc. of the 2000 ACM SIGMOD Intl. Conf. on Management of Data.* 2000. Dallas, Texas.

28. Randy, H.K., et al., *A Project on High Performance {I/O} Subsystems.* Computer Architecture News, 1989. **17**(5): p. 24--31.

29. Sonune, Satyajeet, *Data Stream Management System*, Thesis, University of Texas at Arlington, 2003

BIOGRAPHICAL INFORMATION

Ambika Srinivasan was born on May 28, 1977 in Mumbai, India. She received her Bachelor of Engineering degree in Electrical Engineering from University of Bombay, Maharashtra, India in September 1998. She worked for Tata Consultancy Systems from August 1998 to May 2001 as a software analyst. In the Fall 2001, she started her graduate studies in Computer Science and Engineering at The University of Texas, Arlington. She received her Master of Science in Computer Science and Engineering from The University of Texas at Arlington in December 2003. In her graduating semester, she started working for a company specializing in creating sales forecast and prediction related models for the retail industry. Her research interests include Decision Support Systems, Prediction Algorithms, Data Mining and Business Intelligence.