SQL-BASED APPROACH TO SIGNIFICANT INTERVAL DISCOVERY IN TIME-SERIES DATA

by

SUNIT SHRESTHA

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

August 2005

TO MY PARENTS, FAMILY AND FRIENDS

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere gratitude to my advisor, Dr. Sharma Chakravarthy, for his encouragement, guidance and support throughout my thesis. We had endless arguments and fruitful discussions about various issues and always reached logical conclusions. I would like to thank Dr. Mohan Kumar and Dr. Ramez Elmasri for serving on my committee.

I am grateful to Raman Adaikkalavan for endless advice, Vihang Garg for his faith and confidence on me and Dhawal Bhatia for invaluable suggestions. I owe my sincere thanks to Ambika Srinivasan for clearing a lot of doubts and confusion in spite of her busy schedule and family responsibilities. I am thankful to Niroj Manandhar for his invaluable help and advice during the implementation of this work. I acknowledge friendship and companionship shared by all the friends of ITLAB.

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 thank my family members for their endless love and support throughout my academic career. July 25, 2005

ABSTRACT

SQL-BASED APPROACH TO SIGNIFICANT INTERVAL

DISCOVERY IN TIME-SERIES DATA

Publication No. _____

Sunit Shrestha, M.S.

The University of Texas at Arlington, 2005

Supervising Professor: Dr. Sharma Chakravarthy

With time-series data, events (like turning off a light, opening garage door, turning on TV) occur with a high degree of certainty not at specific time points but within time intervals (sequence of time points). So, it is useful for applications to consider data as contiguous time points. The smallest interval that satisfies the criteria of interval-confidence (i.e., ratio of total support of participating time points and the number of days) is termed as Significant Interval (SI). Significant Interval Discovery (SID) algorithm finds SIs from time-series data.

The main focus of this thesis is on the improvement of existing SID algorithms and the design and development of new SQL-based algorithms which work directly on Relational Database Management System (RDBMS). The experiments compare the performance of the main memory SID against SQL-based SID. The larger goal of this thesis is to achieve scalability.

TABLE OF CONTENT

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Dwindling cost of storage and advances in data warehousing technology have allowed corporations to collect huge amounts of data, and hence data mining has become important to extract useful knowledge from the data collected and to leverage it for business purposes. Large data sources suitable for mining are growing in number and size literally every passing moment. The initial work in the field of data mining was primarily focused on developing algorithms for new mining techniques (e.g., association rules) and to scale existing mining techniques (e.g., classification, clustering) to large data sets. Most early works were developed for data stored in file systems and specialized data structures and buffer management strategies were devised for each algorithm. Recently, a lot of work [15, 18, 17, 16, and 19] has been done for scaling data mining techniques to very large data sets. Integrating mining with databases is an important research effort in this direction. The authors in [14] have classified research on database integration of mining into two categories- one which proposes new mining operators and the other which leverages the query processing capabilities of current RDBMS. In the former category, there have been language proposals to extend SQL with specialized mining operators. A few examples are DMQL, M-SQL and the Mine rule operator. These proposals do not address processing techniques for these operators inside a database engine. In the second category,

researchers have addressed the issue of exploiting the capabilities of conventional RDBMS to execute mining operations. This entails transforming the mining operations into database queries and in some cases developing newer techniques that are more appropriate in the database context. SETM algorithm, the formulation of association rule mining as “query flocks” and SQL queries for mining all aim at tighter database integration. The focus of this thesis is to achieve tighter database integration for SID algorithm.

Roddick et al. in [11] explain that the mounting recognition of temporal data has resulted in the prospect of temporal data mining. It is an important extension of data mining as it has the capability of mining activity rather than states and thus, inferring relationships of contextual and temporal proximity, some of which may also indicate a cause-effect association. In particular, the accommodation of time into mining techniques provides a window into the temporal arrangement of events. Thus, it provides an ability to suggest cause and effect that are overlooked when the temporal component is ignored or treated as a simple numerical attribute. Mover over, it has the ability to mine the behavioral aspects of (communities of) objects as opposed to simply mining rules that describe their states at a point in time. In other words, there is the promise of understanding *why* rather than merely *what*. Consider an association rule stating that gloves and coffee are bought together during winter. This is a temporal association rule - the static equivalent would simply associate the two products. The temporal aspect “during winter” is important. The association may be rare during the

rest of the year, so the association between the two products may go undetected if the analysis concentrates on static association rules only.

A number of algorithms have been developed [11] to process time-series data. Time-series is a sequence of values of a given variable ordered by time. Existing mining techniques treat these values as unique events (i.e., events are considered to occur at particular time points). With time-series data, events occur with a high degree of certainty not at specific time points but within intervals (set of time points). So, events are better understood in terms of intervals rather than time points for many applications as explained in [4]. Some work has been done to find association rules over ranges of values, which are explained in [20, 21]. For example, it is useful to extract information from telephone logs in terms of time periods of high activity to understand the network use. For Smart Home [2], it is useful to consider periods of high activity of the devices rather than their actual usage at a particular time, to infer the usage patterns of each device as well as interactions between different devices. Several numerical domains, such as a magazine subscription company wanting to mine information about its subscribers with highest subscriptions, can use intervals of age groups, instead of processing for each and every subscription. Representing events with intervals has several advantages: First, it provides an opportunity to explore and identify significant intervals, and in the process provides a better understanding of the underlying data. Second, it reduces the size of the data to be used for discovering sequential patterns by mining algorithms [3].

Time-series data is known for its volume and discovering useful segments of information from them is a challenging task. Most of the traditional sequential mining algorithms [1, 22] deal with events occurring at a point in time and they process the entire dataset repeatedly. For time-series or numerical data, the sheer size of the dataset makes running an algorithm that makes repeated passes on the entire dataset time consuming. The efficiency of these algorithms can be improved by reducing the dataset that these algorithms work with. Time-series point values can be compressed by using the notion of intervals.

1.1 Significant Interval Discovery Algorithm

Srinivasan [3] has laid out the foundation work for the detection of significant intervals. Intervals are represented as [T1, T2, s, l, d, ic] where *T1* and *T2* represent the start and end time of an interval, *s* represents the support of the interval, *l* denotes the length of the interval (T2-T1+1), *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. In a numerical domain, let *S* be the sum of the supports at all the points in the dataset.

- *Support (s): event count for a time point or sum of event count for an interval*
- *Length (l): T2-T1+1*
- *Density (d): s/l*
- *Interval-Confidence (ic): s/N (for time-series data) Or s/S (for numerical data)*

Significant Interval (SI) is defined as an interval that satisfies the user-specified criteria of maximum Interval-length (max-Len) and minimum Interval-confidence (min-

Conf). The author has explained how significant intervals can be identified from time-series data for a given max-Len and min-Conf. The author has proposed Significant Interval Discovery (SID) algorithm, which is a time-series data mining algorithm and uses the concept of intervals to find significant intervals. SID is a level-wise iterative algorithm and have three important phases. They are:

- **Preprocessing**: In this phase, support of the distinct time points are obtained by folding the data over the periodicity specified by the user. Currently, the algorithm processes each distinct event sequentially. After the folding of data, first level intervals are generated by merging adjacent time points.
- **Interval Formation**: First level intervals are expanded by merging them with adjacent intervals and significant intervals are selected from them.
- **Cluster Formation**: Overlapping significant intervals are consolidated to represent as clusters.

The algorithm is implemented in main memory [SIDH] and uses database to store the initial results and final results. SQL is used for counting support of distinct time point events and also for time wrapping (which is explained in Chapter 3.2.1.1). The algorithm is working for daily and weekly periodicities with minute's granularity. Based on the merging of intervals in interval expansion, suites of SID approaches have been proposed to identify significant intervals efficiently. They are SID[1], SID[1, n-1], SID[n-1], and SID[n-2]. These algorithms form a spectrum from the exhaustive to the most efficient without significant loss in the output generated. The numbers in the

brackets indicate how intervals from the n-1th iteration are combined to form intervals of the nth iteration.

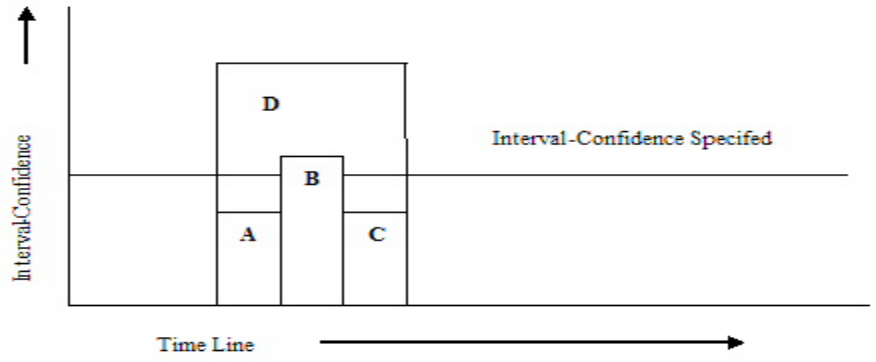

Figure 1.1 Significant interval subsumed

Upon closer investigation, a number of issues have been identified that needed improvement. One such area is the proper definition of significant intervals and their identification. Consider a situation in MavHome (which is the predominant problem domain for this work) where a user turns off his TV every night at 11:00 pm. SID algorithm failed to identify this event as a significant event. However, it is a significant event and the automated home systems (e.g., MavHome), should turn off TV every night at 11:00 pm exactly. Such events are called "*Unit Length Significant Interval*" where a time point that meets min-Conf criteria forms an interval.

Further, the algorithm detects the largest SI for a given value of max-Len and min-Conf. If there is a larger SI, which subsumes another smaller SI, the algorithm identifies larger interval as SI not the smaller one. This creates a problem in case of MavHome. Consider the Figure 1.1, B and D are SIs. Interval B is the smallest SI and

there are two insignificant intervals A and C. Then, the algorithm identifies that interval D, which encompasses A, B and C is significant even though the intervals A and C are not significant. Interval D is significant because it subsumes SI B. If interval B represents an interval with intense device activity in case of MavHome, the algorithm is predicting that interval D needs to be automated for that event. This introduces unnecessary overhead of keeping an event active in those intervals where it is not really significant at all.

The implementation of SID suite of algorithms in [3] is loosely coupled with database. Database is used as a container to store the initial raw data and final results. Rest of the processing is done by fetching data from the database and storing them in main memory data structures. The algorithm does not store transaction data in main memory. It only stores folded data (which is explained in Chapter 3.2.1.1), which is much smaller than transaction data. Even then, the algorithm is limited by the size of main memory to handle large amounts of data. Consider a situation pertaining to MavHome in which that the system is tracking the events of an application in second's granularity. Then, in the worst case there can be 31,536,000 transactions per year for a single device and there can be 86,400 transactions with folding on a daily basis or 604,800 with folding on a weekly basis. If the system is tracking a number of devices, then this number is going to be several times larger. With scaling experiments, it was observed that SID stopped due to main memory limitation when data size was very large.

1.2 Focus of the Thesis

With increase in the use of Relational Database Management Systems (RDBMS) to store and manipulate data, mining directly on RDBMS gives us the advantage of using the fruits of decades of research done in this field. Main memory always imposes a limitation on the size of data that can be processed. However, the use of RDBMS provides the benefits of using their buffer management systems specifically developed for freeing the user/applications from the size considerations of the data. Building mining algorithms to work on RDBMS also gives the advantage of mining over very large datasets, as RDBMS have been built to manage such large volumes of data. File based mining algorithms are those that work on data outside of the database. They generally have an upper limit on the number of transactions that can be mined. For example, the DBMiner has an upper limit of 64K on the number of unique transactions that it can process for mining. Main memory implementation of SID with new definitions of SI has an upper limit of 6.4 Million Transactions. With the user having a choice of RDBMS to use for his application, the mining algorithms should be developed using accepted standards so that the underlying system is not a limitation and should be portable to other RDBMS. Keeping this in mind, our focus in this thesis is to develop SQL-based SID algorithms that provide scalability in addition to the core functionality of the algorithm. We have tried to use queries that conform to the SQL-92 standard.

Several improvements to the earlier SID algorithms have also been incorporated. The current implementation fails to identify "*Unit Length Significant Intervals*" (i.e., time points with sufficient confidence which has same start and end

time). Also, it identifies significant intervals that contain another significant interval. In this thesis, we have refined the notion of SI and take into consideration disjoint and overlapping SIs.

The remainder of the thesis is organized as follows. Chapter 2 summarizes literature survey done. Chapter 3 describes Extended SID (SID). Chapter 4 provides the details of developing SQL-based algorithms for significant interval detection. It includes both the design and implementation aspects of SQL-based algorithms, which form the core contributions of this thesis. Chapter 5 has the experimental results. Finally, Chapter 6 concludes with future work.

CHAPTER 2

RELATED WORK

2.1 Mining Interval Time-series

Not many data mining algorithms discuss the formation of intervals on time-series data based on the interaction of events. Villafane et al. [5] propose a technique to discover temporal containment relationships using intervals. An item A is said to contain an item B if an event of type B occurs during the time span of an event of type A, and occurs frequently in the data set. As an example, let us consider a database application in which a data item is locked and then unlocked sometime later. Instead of treating the lock and unlock operations as two discrete events, it can be advantageous to interpret them together as a single interval event that better captures the nature of the lock. When there are several such events, an interval time-series is formed. Formally, let BeginTime(X) and EndTime(X) denote the start and end time of an event X, respectively. Event X is said to contain event Y if BeginTime(X) < BeginTime(Y) and EndTime(X) > EndTime(Y). SID focuses on finding intervals of significance for events rather than relationship between events. For the above case, SID finds the SIs over the time line for event (X) and event (Y). Then, these SIs are used to find association or relationship between those events.

2.2 Discovering Frequent Episodes in Time-series

As in case of WinEpi [1], SID also uses the concept of sliding window for support counting. SID calls it *folding* wherein the time-series data that spans months or years are represented in terms of days or weeks based on the periodicity of interest. If the periodicity of interest is “daily”, then the entire dataset is folded over that periodicity (24 hours) and the number of occurrences at each time point is considered as the support of the event at that point. WinEpi makes multiple passes over the data for counting the support of the candidates in each pass. SID is closer to MinEpi as SID obtains the support and confidence of events in a single pass over the data and uses them to obtain support and confidence for the intervals. Referring to the timing constraints, WinEpi and MinEpi [1] finds all sequences that satisfy the time constraint ms (Maximum Span: maximum allowed time difference between latest and earliest occurrences of events in the entire sequence) and whose support exceeds a user-defined minimum min_sup, counted with the CWIN (One occurrence per span window) method. Similar concepts are used in the SID, which are called max-Len and min-Conf.

2.3 Integrating Association Rule Mining with Relational Database Systems

Sarawagi et al. [7] compare different architectural alternatives for coupling mining with database systems. These alternatives include: loose-coupling through a SQL cursor interface; encapsulation of a mining algorithm in a stored procedure; caching the data to a file system on-the-fly and mining; tight-coupling primarily using user-defined functions; and SQL implementations for processing in the DBMS. The authors’ mention that Cache Mining provides the best performance and they mention

that SQL-based implementation is the worst. They have used Association Rule Mining as their case. However, in this work, it is found that though Cache Mining provides a good performance for relatively modest data set but there is a performance bottleneck due to the main memory and that Cache Mining does not scale well for large datasets. The authors mention various advantages of SQL-based implementation. One can make use of the database indexing and query processing capabilities thereby leveraging more than a decade of effort in making these systems robust, portable, scalable, and concurrent. One can also exploit the underlying SQL parallelization, particularly in an SMP environment. The DBMS support for check pointing and space management can be valuable for long-running mining algorithms. We have choosen SQL-based implementation of SID algorithms for the purpose of scalability.

2.4 Frequent Itemset Discovery with SQL Using Universal Quantification

Rantzau [8] asserts that there is a need to look at SQL-based approaches for finding frequent itemsets even though their performance is inferior to main memory algorithm, the reason being the current trend among database vendors to integrate analysis functionalities into their query execution and optimization components. A new approach called “Quiver” is proposed that employs universal and existential quantifications. Conceptually SID is similar to Frequent Itemset Discovery (FID) if time points and time-intervals are thought of as items. The intervals formed from merging time points and time-intervals will be denoted by start and end time. Basically a record represents a group of time points. Understanding of FID makes it easy to understand SID. However, there are few subtleties that differentiate it with FID. First, it uses the

concept of min-Conf instead of minimum support to decide where an interval is frequent (i.e., significant in case of SID) or not. Second, the candidate generation phase is marked by the limitation imposed by the max-Len constraint specified by the user.

2.5 Programming the K-means Clustering Algorithm in SQL

Ordonez in [9] concede that using SQL has not been considered an efficient and feasible way to implement data mining algorithms. However, he explains how to implement K-means clustering algorithm in SQL efficiently emphasizing on its correctness and performance. From a correctness point of view the author explains how to compute Euclidean distance, nearest-cluster queries and updating clustering results in SQL. From a performance point of view, he explains how to cluster large data sets by indexing tables, optimizing and avoiding joins, optimizing and simplifying clustering aggregations, and taking advantage of sufficient statistics. K-means uses Euclidean distance to find the nearest centroid to each input point. The algorithm has two main steps E and M. The E step determines the nearest cluster for each point and adds the point to it. That is, the E step determines cluster membership of the points. The M step updates all centroids by averaging points belonging to the same cluster. The algorithm iterates executing the E and M steps starting from some initial solution until cluster centroids become stable. K-means forms clusters and the number of clusters formed is determined by the parameter “K” specified by the user. SID is similar to clustering in the sense that it also forms clusters of adjacent time points as intervals. It is different from clustering as there is no limit set on the number of significant intervals to be discovered. Clustering does not maintain sequential order for data but clustering of

events has to maintain the sequential order as well. This is a significant departure from the traditional clustering algorithms. While K-means partitions all the participating points into one or another cluster, SID basically partitions the time points as being significant or insignificant. The author presents two main schemes. While Standard K-means implementation uses SQL, Optimized K-means is an implementation incorporating several optimizations. As in Standard K-means, our implementation of SID also uses standard SQL only. The author seems to be interested in avoiding join even though he believes that a solution based on joins is more elegant and simpler. We have used joins in some of our important SQL statements.

2.6 SQL-based Frequent Pattern Mining with FP-growth

Xuequn et al. in [10] explain that finding frequent patterns is a fundamental component in data mining, which has been extensively researched and studied. Most of the studies adopt an Apriori-like candidate set generation and test approach; if any length K pattern is not frequent in the database, its length (K+1) super pattern can never be frequent. The aim of SID is similar as it tries to find frequent tightest intervals. SID has candidate generation and selection steps wherein candidate intervals are generated and tested to check whether they are significant or not. It is different from Apriori approach as (K+1) significant intervals need not be generated from k^{th} significant intervals. In fact, SID tries to avoid the generation of such intervals because it is only interested in tightest significant interval. The authors mention that candidate set generation is costly. A novel method for frequent pattern mining known as frequent pattern growth (FP-growth) has been proposed. FP-growth method adopts the divide

and conquers strategy, uses only two full I/O scans of the database and avoids iterative candidate generation. The first scan accumulates the support of each item and then selects items that satisfy minimum support. In fact, this procedure generates frequent-1 items and then stores them in frequency descending order. The second scan constructs FP-tree. A FP-tree is a prefix-tree structure storing frequent patterns for the transaction database, where the support of each tree node is no less than a predefined minimum support threshold. Each node in the item prefix subtree consists of three fields: item-name, count and node-link. Node-link links to the next node in the FP-tree carrying the same item-name, or null if there is none. SID also avoids multiple scans of the database. In fact, it scans the database only once to generate support of event time points. As is the case of FP-tree, the results generated by SID are highly compact and is much smaller than its original database due to compression achieved with folding and representation of sequence of time points with intervals.

CHAPTER 3

EXTENDED SIGNIFICANT INTERVAL DISCOVERY

3.1 Extension of Significant Intervals Representations

Given a time sequence T, minimum confidence *min-Conf* and maximum interval-length *max-Len*, we define an interval w [ts, te] to be *Significant Interval* in T

(1) interval confidence of w, *ic* >= *min-Conf* and

(2) length of w, *l* <= *max-Len* and

(3) there is no other window w'= [ts', te'] in w for which conditions (1) and (2) hold. In other words, a significant interval cannot subsume another significant interval (of length one or more).

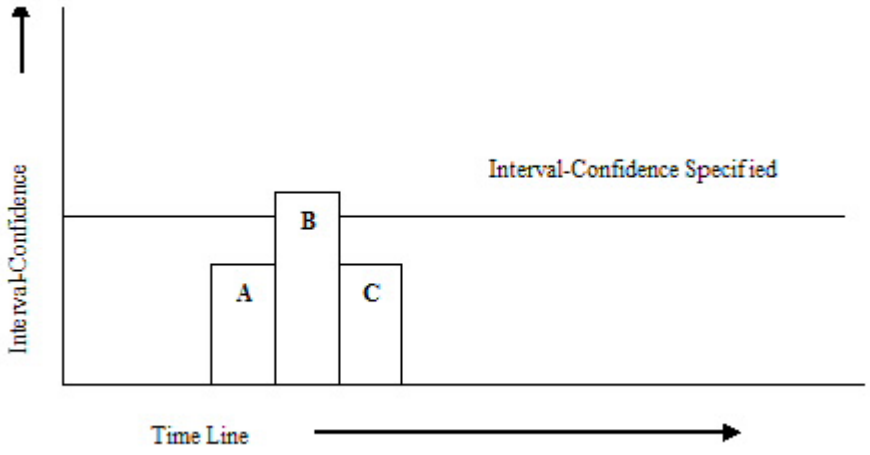

Figure 3.1 Improved significant interval

Significant intervals can be either disjoint or overlapping. If the start (end) time of an interval w falls within start and end time of another interval w' and the end (start) time of the interval is outside of the interval w', then the intervals w and w' are overlapping. Formally, *Overlapping Significant Intervals* are defined as two significant intervals w [ts, te] and w' [ts', te'] if (ts <= te'< te and ts' < ts) or (ts < ts'<= te and te'> te). *Disjoint Significant Intervals* are defined as two significant intervals (i.e., w [ts, te] and w' [ts', te']), which do not overlap (i.e., ts' > te and te' is not in the interval [ts, te] or te'< ts and ts' is not in the interval [ts, te]). *Unit Length Significant Intervals* are defined as significant intervals where ts = te.

As per the revised definitions of a significant interval (SI), the algorithm would find only interval B as significant interval as shown in Figure 3.1. This is a refinement of SID explained in [3].

3.2 Refined Significant Interval Discovery Algorithm

Given a time-series data, SID algorithm finds the time intervals, within which the events appear at least as frequently as a given threshold, called min-Conf. The user defined parameters such as time-granularity, periodicity, max-Len and min-Conf are read from a configuration file and all calculations are performed accordingly. *Time granularity* is the granularity of time-stamp. It represents whether events are recorded using second's or minute's granularity and the user has to specify granularity in the configuration file. *Periodicity* represents the kind and frequency of events that are being searched. The user specifies daily (or weekly) periodicity if he is looking for events that occur on a daily basis (or weekly basis). *Max-Len* specifies the maximum length for the

intervals searched and *Min-Conf* specifies the minimum interval confidence needed to classify an interval as significant.

SID is a level-wise iterative algorithm that consists of a sequence of steps that proceed in a bottom-up manner; the result of the k^{th} step is generated from the results of $k\text{-}1^{th}$ step. This algorithm also uses the Apriori approach of candidate generation and selection. It differs from [3] in the deletion of significant intervals that are performed in each iteration. Time points and time intervals that have already participated in interval formation are deleted from the input. This progressively reduces the size of the data processed by the algorithm. The algorithm can be partitioned into 3 phases:

- Preprocessing
- Interval Formation
- Post-processing

The Figure 3.2 below illustrates the SID algorithm graphically.

3.2.1 Preprocessing

The algorithm starts with the folding of time-series data and formation of first level intervals. Folding enables the accumulation of support for events at all the distinct time points in a given periodicity (and time granularity) and further it compresses the data.

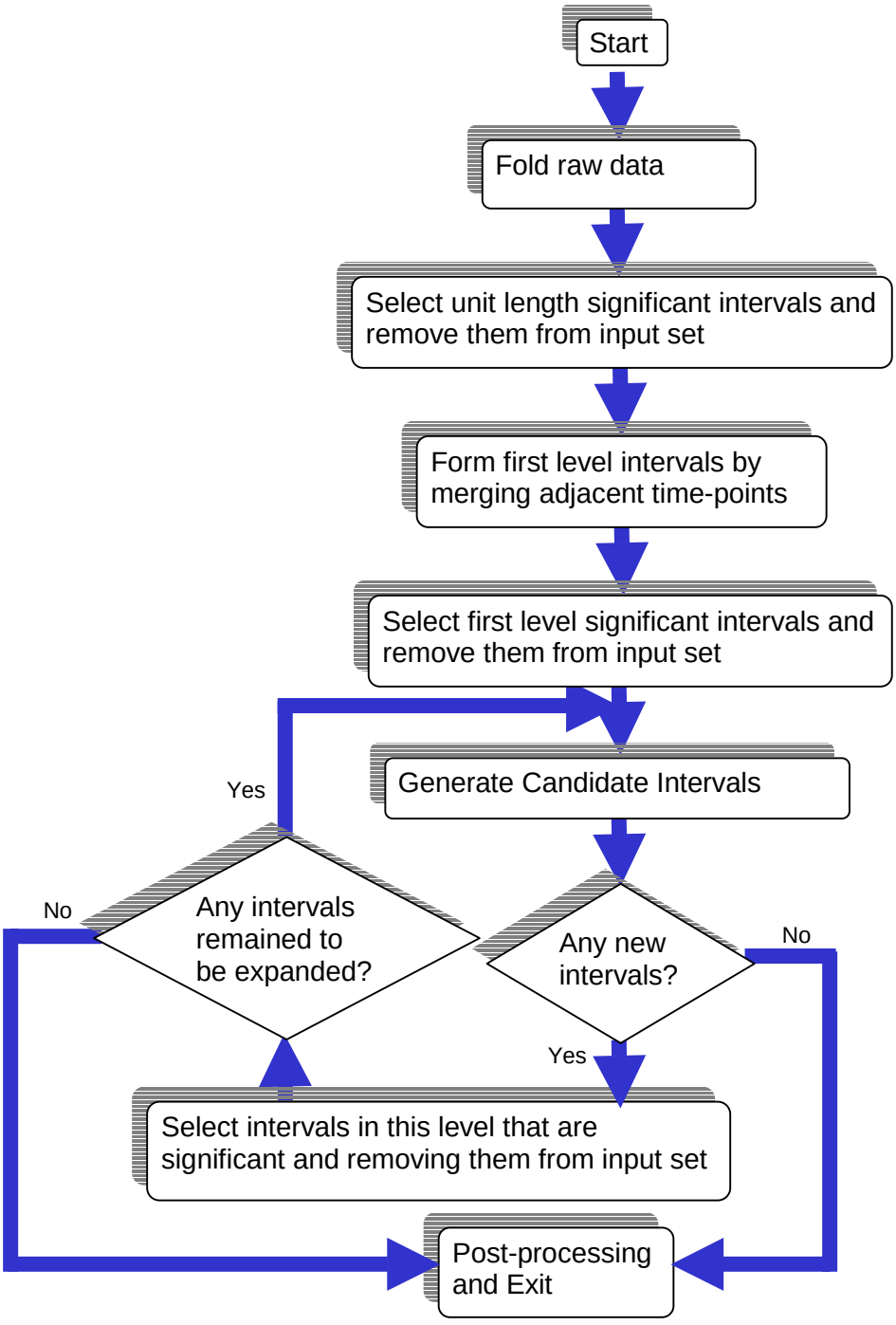

Figure 3.2 Flowchart of the SID algorithm

3.2.1.1 Folding

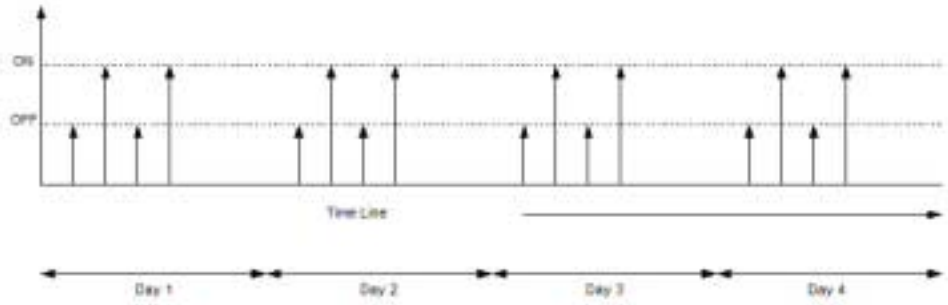

Figure 3.3 Time-series data

Time-series data consists of events recorded with a time-stamp. Consider the Figure 3.3 which shows the occurrence of events over a time line. This time line event activity can be considered to be consisting of distinct and disjoint windows of a period of a day or week (termed *periodicity*). Each event in a window represents its count. If the size of the window is chosen as 24 hours, then it is known as folding on a daily periodicity. If the time line extends for 10 days, the maximum count of any event on a daily periodicity can be at most 10. The events in the Figure 3.3 can be represented as shown in Table 3.1.

Table 3.1 Transaction data

Device	Status	Time-Stamp
Lamp1	On	8/10/2005 1:00
Lamp1	OFF	8/10/2005 2:00
Lamp1	On	8/10/2005 3:00
Lamp1	OFF	8/10/2005 4:00
Lamp1	OFF	8/11/2005 2:00
Lamp1	On	8/11/2005 3:00
Lamp1	OFF	8/11/2005 4:00
Lamp1	On	8/12/2005 1:00
Lamp1	OFF	8/12/2005 2:00

Table 3.1 shows the data collection for Lamp1. After folding on a daily periodicity, we obtain distinct time point events with their corresponding support as shown in Table 3.2 below.

Table 3.2 Input dataset after Folding

Device	Status	Time of occurrence	Support
Lamp1	On	1:00	2
Lamp1	OFF	2:00	3
Lamp1	On	3:00	2
Lamp1	OFF	4:00	2

The process of folding also compresses the data significantly. In the above example, 12 records have been reduced to 4 records that will be used for significant interval processing (a reduction of 66%). As another example, if there are 31,536,000 occurrences of an event over a period of one year recorded using second's granularity, it will reduce to 86,400 (reduction of 99%) occurrences with folding on a daily basis. With folding on a weekly basis, it will reduce to 604,800 (reduction of 98%). As the number of points is fixed given a periodicity, greater reduction is achieved if the data represents larger number of periodicity units (daily, weekly, monthly etc.).

3.2.1.2 Time wrapping

The compression of data due to folding should occur without the loss of any information of interest. With folding on a daily or weekly periodicity, all intervals that lie within the window of a day or week can be obtained but there can be some intervals which might span two days or weeks. Consider a situation where an inhabitant of home operates his TV between 11:00 pm to 2:00 am every night. Then, the interval 11:00 pm

to 2:00 am becomes significant interval for automation. However, folding of data on a daily basis will not lead to identification of such interval and some interesting intervals that span two days or weeks may not be discovered at all.

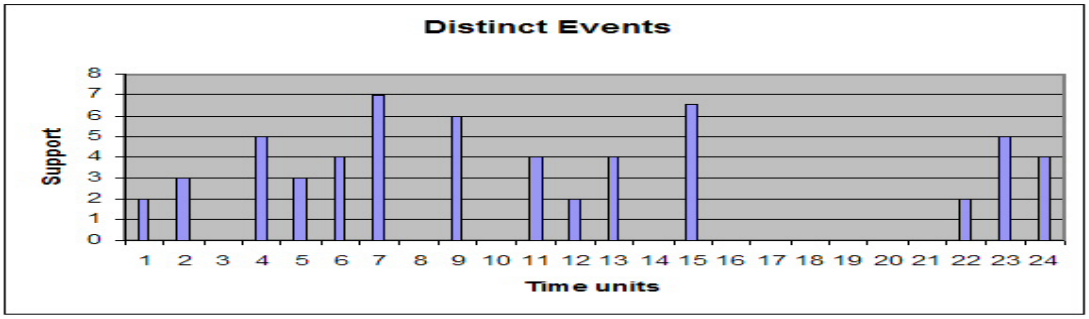

Figure 3.4 Distinct events with support obtained after folding on a daily periodicity

This is taken care of in the algorithm by means of Time wrapping. Time wrapping allows the formation of intervals that spans two days or weeks. All time-points that fall below the start of that day but lying below the max-Len are replicated and given a separate day value or week value. If max-Len is specified as 30, then time points, which is less than 00:30 can participate in the intervals spanning two days. After time wrapping, distinct events in Figure 3.4 will look like Figure 3.5 as below.

3.2.1.3 Identification of unit length SIs

Some of the time point events that are obtained after folding may have sufficient confidence to qualify as significant intervals. These are shown in Figure 3.5 with circles. There are 3 time points which have sufficient confidence. These time points form interval with themselves and they are known as "*Unit Length Significant Intervals*" (Unit SIs).

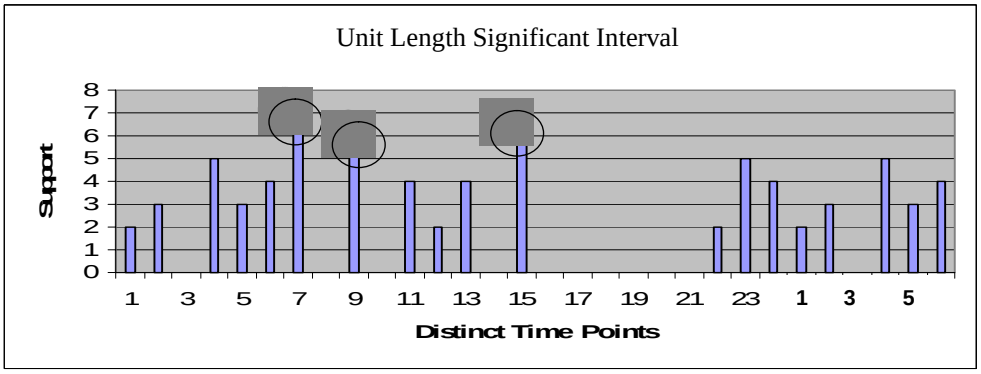

Figure 3.5 Unit length significant intervals

Unit SIs are removed from the input before first level intervals are formed. Figure 3.6 shows the remaining time points.

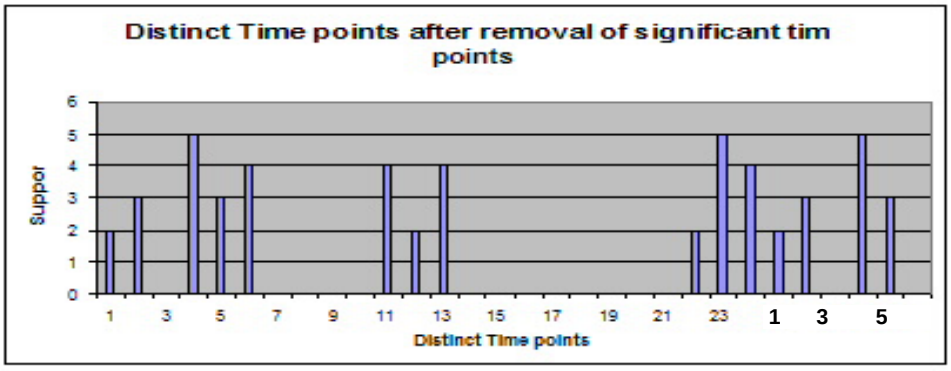

Figure 3.6 After removal of unit length significant intervals

3.2.1.4 Formation of first level intervals

SID is a level-wise algorithm which proceeds in a bottom-up approach. The lowest level is the distinct time point events and these distinct time point events are merged to form intervals. One way of merging the time points is to consider all possible intervals that can be formed by merging two time points. However, it is a very

exhaustive and computationally expensive approach. Consider time point 01:00, 02:00, 04:00, 05:00, 06:00 and others as shown in Figure 3.5 above. Time point event 01:00 can form interval with 02:00, 04:00, 05:00 and 06:00 assuming that max-Len specified is 6. Then, for each interval of 01:00, interval confidence can be compared and checked to determine whether it is less than min-Conf specified. If it is, then that interval can be selected as SI. The problem with this approach is that it generates all possible intervals for any time point and only a few out of these intervals will be selected as SIs. Furthermore, if interval 01:00-02:00 has been identified as SI, still all the intervals like 01:00-04:00 and 01:00-06:00 are calculated and selected as SIs, which violates the 3rd definition of SI. A better approach would be to start with a smallest interval and grow the interval with small units till significant intervals are obtained. The smallest interval that can be obtained between any two time points is the first level intervals.

Table 3.3 First level intervals

Device	Status	Start Time	End Time	Support
Lamp1	On	1:00	2:00	5
Lamp1	On	2:00	4:00	8
Lamp1	On	4:00	5:00	8
Lamp1	On	5:00	6:00	7
Lamp1	On	11:00	12:00	6
Lamp1	On	12:00	13:00	6
Lamp1	On	22:00	23:00	7
Lamp1	On	23:00	0:00	9

From the vertical database layout, first level intervals (shown in Table 3.3) are formed by coalescing adjacent points, which are within max-Len specified. If any of the first level intervals meet the min-Conf specified, then it is selected as SI and removed

from the input dataset. Figure 3.7 shows 5 first level SIs. First level SIs are removed from the input data set and the remaining first level intervals are shown in Figure 3.8 below.

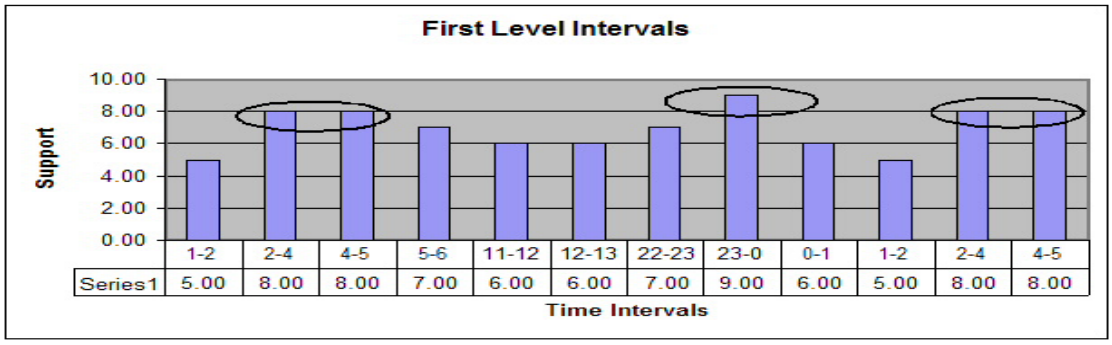

Figure 3.7 First level significant intervals

3.2.2 Interval Formation

The remaining first level intervals are expanded in interval formation stage. The algorithm follows an iterative process to grow first level intervals. In each iteration, the intervals obtained from previous iteration are extended by merging them with adjacent intervals from the first level (for naïve) and other levels (for other variants).

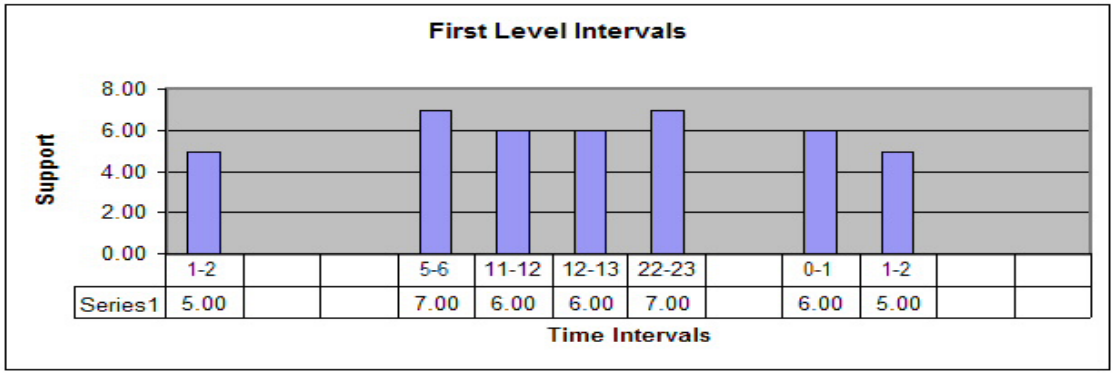

Figure 3.8 After removal of first level significant intervals

The condition for merging two intervals is that end time of one interval should coincide with start time of another interval. Interval Formation phase consists of three steps:

- Candidate interval generation: Every new interval that meets the max-Len constraint is considered as a candidate.

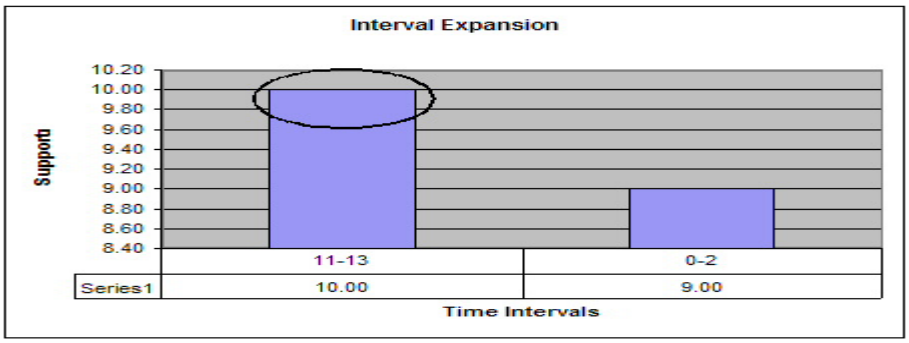

Figure 3.9 Expansion of first level intervals in interval formation phase

Figure 3.9 shows interval expansion of first level intervals in Figure 3.8. For example, intervals (11:00-12:00 and 12:00-13:00) can merge to form an interval (11:00-13:00). Intervals (11:00-13:00, 0:00-2:00) will be considered as candidate intervals. If the interval (11:00-13:00) is selected as SI, then it is removed from the input dataset and it won't participate further in any interval formation. Intervals (1:00-2:00, 5:00-6:00, 22:00-23:00) could not be expanded as they don't have any adjacent intervals, which can be used to expand those intervals. Revised SID algorithm follows the definitions of Naïve, SID[n-1], SID[n-2] approaches and corresponding merging criteria as explained in [3].

- Selection of significant intervals: New interval that meets min-Conf constraint is selected as a SI.
- Deletion of intervals: For a SI, the participating time-intervals are removed from the input set to disable the formation of any other interval, which might subsume this interval. This is a significant departure form the SID algorithm explained in [3]. Deletion of SIs not only prevents the formation of SIs that subsume another SI, it also reduces the data size to be processed in each iteration thereby completing the iteration quickly.

The algorithm iterates until there are no more intervals for merging or no new candidate intervals are generated.

3.2.3 Post-processing

Post-processing is done to remove redundant intervals introduced by the *Anomaly of Time Wrapping* and *Subsumption*. These are explained below.

3.2.3.1 Anomaly due to Time wrapping

Time wrapping prevents the loss of information but it may also introduce redundant SIs. This is known as *Anomaly of Time wrapping*. As we replicate time points to occur in two days (or weeks), they may lead to formation of intervals whose length is less than max-Len specified. In such cases, some intervals might be identified as SIs in two different days (or weeks). In Figure 3.10, intervals (2:00-4:00, 4:00-5:00) are repeated in two different days. If these intervals are identified as significant, then there will be two entries for the same interval with different dates in the database. The final result should contain only one set of records.

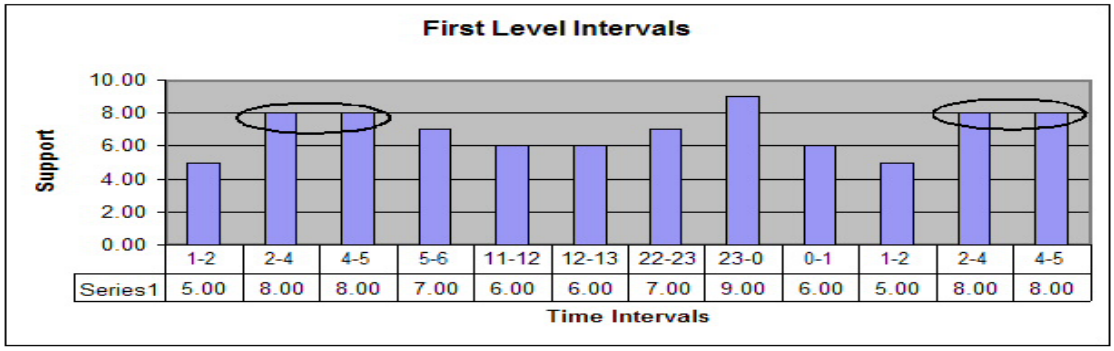

Figure 3.10 Anomaly introduced by time wrapping

3.2.3.2 Subsumption

Also, there are redundant significant intervals introduced by *Subsumption*. When distinct time points are found to be significant, they are selected as Unit SIs. But the remaining time points may lead to formation of intervals that subsume these Unit SIs. As shown Figure 3.11, time point 2:00 is found to be significant and is removed from the input set. However, time points 1:00 and 3:00 merges to form first level interval, which is also found to be significant. Then, interval 1:00-3:00 violates the third definition of SIs, which basically says that a SI cannot have another SI inside it. In case of 1:00-3:00 interval, it has another SI 2:00-2:00. So, interval (1:00-3:00) though significant has to be removed from the output as it subsumes another SI.

Similarly, in case of SID[n-1] and SID[n-2], SIs that subsume another SIs are generated due to merging of intervals at different levels to form new intervals. Consider the intervals (1:00-3:00, 2:00-4:00 and 3:00-5:00) that are obtained in second level of SID[n-1]. If the interval 2:00-4:00 is found to be significant, it will be removed from the

data. The remaining intervals 1:00-3:00 and 3:00-5:00 will merge to form a new interval 1:00-5:00 in third level. If 1:00-5:00 is also found to be significant then, it will subsume another significant interval 2:00-4:00 and violates the third definition of SI. So, 1:00-5:00 has to be removed from the output.

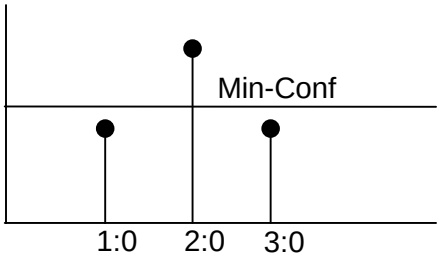

Figure A: Time points with their confidence

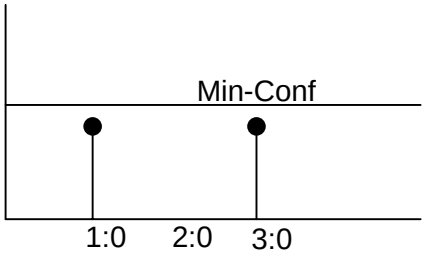

Figure B: After removal of unit length significant interval

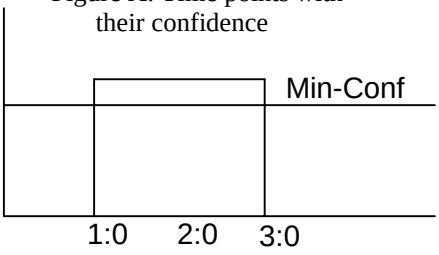

Figure C: Formation of first level interval which is significant

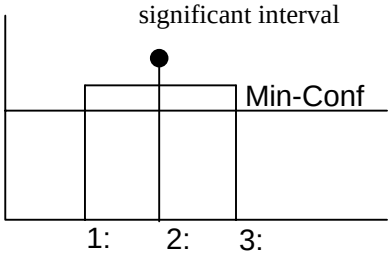

Figure D: Unit length significant interval subsumed

Figure 3.11 Subsumption for unit length significant interval

CHAPTER 4

SQL-BASED AND MAIN MEMORY APPROACHES

4.1 Comparison of Main Memory and SQL-based Approach

Table 4.1 Comparison of main memory and SQL-based approaches

	Main Memory Approach	SQL-based Approach
Coupling	Data is stored in a database. The algorithm has to load all the data into main memory to process them.	Data already resides in the database. No need to retrieve data into main memory as the algorithm is implemented in SQL.
Scalability	The amount of data it can process is limited by size of main memory.	Buffer manager of database enables it to handle large volumes of data.
Performance	Can use compression or special data structures to achieve performance gain. Can write optimized algorithm for fast processing	Database adds the overhead that can make it slow. Can use some features like indexing and optimized queries to have performance improvement.
Portability	Language used to implement data mining algorithm may be platform specific. So, portability can be an issue.	Can be implemented in SQL-92, which is an industry standard followed by all database vendors. So, the implementation will be portable to other databases.

Most performance experiments have shown that SQL-based approaches are inferior to main memory algorithms. However, the current trend of database vendors to integrate analysis functionalities into their query execution and optimization components (i.e., “closer to the data”) suggests revisiting these results and searching for new, potentially better solutions. The data generated by MavHome, which is our predominant problem domain, resides in a database (Oracle in our case). The results

generated by SID are used by another algorithm called Hybrid-Apriori [HA]. HA uses the SIs identified by the SID algorithm to determine the patterns of device activity. It is implemented in SQL. Therefore, it is desirable to implement SID in SQL so that we have a set of algorithms to generate sequential pattern tightly coupled with database. The advantage of main memory algorithms is that it can make use of compression or complex data structures, to enhance speed. However, it is limited by main memory in handling large volumes of data. Further, for each algorithm, a separate buffer management technique needs to be developed. SQL-based approach avoids the limitations of main memory as it has a built-in buffer management system. So, it is capable of managing huge volumes of data. Further, it is scalable, portable and stable.

4.2 SQL-based Approach [SIDQ]

SQL is an international standard followed by all database vendors. It is used to create, maintain & query relational databases. A fundamental difference between SQL and standard programming languages is that SQL is non-procedural. Various standards exist for SQL (SQL-86, SQL-89, SQL-92, SQL-99, SQL-2003) but we have tried to use SQL-92 features of Oracle database for our implementation as it is supported by most of the database vendors. SQL is not a computationally complete language and it does not have programming constructs such as loops and conditional check, which are commonly used in general programming language. As SQL is a non-procedural language, the data structures and algorithm that are used by general programming language cannot be used in SQL implementation. Looping and conditional checking can

be implemented in SQL through *JOIN* and *WHERE* clause. So, the implementation challenges observed are:

1. Implement the algorithm in a language that is not Turing complete.
2. Design of tables in the database.
3. Design of algorithms for SQL implementation.
4. Finding efficient implementation of SQL using indexes where possible.

SQL statements required for the execution of the algorithm are generated dynamically by a program written in JAVA and this program generates SQL statements based on parameters specified by the user. The parameters are: *approach_number*, *min-Conf*, *max-Len*, *numDays*, *granularity* and *periodicity*. *Approach_Number* is used to distinguish between SID suite of algorithms as explained by Srinivasan in [3]. The suites of SID algorithms implemented are SID[1], SID[n-1] and SID[n-2]. *NumDays* specifies the number of days for which the data is collected. *Max-Len* specifies the maximum length of the interval to be searched. *Min-Conf* specifies the minimum interval confidence that significant intervals should have. *Granularity* defines the granularity of the time-stamp for the events. Current implementation supports minutes and seconds granularity. *Periodicity* defines the periodicity of interest for the user which can be *Daily* or *Weekly*. The program uses JDBC to connect to Oracle database and all the SQL statements generated are executed in a sequential manner. Intermediate results are stored in tables. The algorithm starts with the processing of raw data stored in *TbTransload* table.

4.2.1 Design of Tables

The first issue in the design of SQL-based algorithm is the question about where to store the intermediate results. In case of main memory, various standard data structures are available and if they are not suitable, desired data structures can be created in the form of objects. In case of database, all the results have to be stored in the tables only. So, it was important to come up with ideal table structures. The following are the list of tables that have been used:

Table 4.2 COUNTSUP table

COUNTSUP TABLE		
COLUMN NAME	DATA TYPE	NULL
TXTDEVICEID	VARCHAR2(30)	NOT NULL
TXTSTATUS	VARCHAR2(31)	NOT NULL
DTTIME	VARCHAR2(10)	NOT NULL
RECORDCOUNT	NUMBER	NOT NULL

- ***CountSup***: This table stores support of all distinct time point events after folding is performed in daily periodicity. The field "*DTTIME*" is a character field and it stores times of the day as a character. The field "*RECORDCOUNT*" is a number field and stores the count of distinct time point events.

Table 4.3 COUNTSUPTEMP table

COUNTSUPTEMP TABLE		
COLUMN NAME	DATA TYPE	NULL
TXTDEVICEID	VARCHAR2(30)	NOT NULL
TXTSTATUS	VARCHAR2(31)	NOT NULL
DTTIME	DATE	NOT NULL
RECORDCOUNT	NUMBER	NOT NULL

- ***CountSupTemp***: This table is similar in structure to "*CountSup*" except for "*DTTIME*" field, which is a date field instead of a character field. For SQL-based implementation, it was important to maintain all the time points in some increasing time order to allow for time wrapping. This is achieved by assigning date to all the time points in "*CountSupTemp*" table.

Table 4.4 DISTINCTINT table

DISTINCTINT Table		
COLUMN NAME	DATA TYPE	NULL
TXTDEVICEID	VARCHAR2(30)	NOT NULL
TXTSTATUS	VARCHAR2(31)	NOT NULL
DTTIME	DATE	NOT NULL
TIMEDIFF	NUMBER	NOT NULL

- ***Distinctint***: This table stores distinct time points that will form first level intervals. It also stores the minimum time difference it will have with the adjacent time point to form first level interval. The field "*DTTIME*" stores the start time of the first level interval and "*TIMEDIFF*" stores the minimum time difference this start time will have with other time point to form first level interval.

Table 4.5 FIRSTLEVEL and CANDIDATE table

FIRSTLEVEL AND CANDIDATE TABLES		
COLUMN NAME	DATA TYPE	NULL
TXTDEVICEID	VARCHAR2(30)	NOT NULL
TXTSTATUS	VARCHAR2(31)	NOT NULL
STARTTIME	DATE	NOT NULL
ENDTIME	DATE	NOT NULL
TIMEDIFF	NUMBER	NOT NULL
INTERVALSUP	NUMBER	NOT NULL
STARTSUP	NUMBER	NOT NULL
ENDSUP	NUMBER	NOT NULL
INTERVALCONF	NUMBER	NOT NULL
DENSITY	NUMBER	NOT NULL

- ***FirstLevel*** and ***Candidate*** Tables: They have the same table structure. "*FirstLevel*" table stores all the first level intervals. A new "*Candidate*" table is created in each iteration inside interval formation. It stores all the candidate intervals generated in that iteration. Besides storing information about the intervals (like *strattime, endtime, time difference, interval support, interval confidence* and *density*), they also store two additional informations "*StartSup*" and "*EndSup*". "*StartSup*" represents the support of a time point that is the starting time of the interval and "*EndSup*" represents the support of a time point that is the ending time of the interval.

Table 4.6 FINALFREQUENTITEMS table

FINALFREQUENTITEMS TABLE		
COLUMN NAME	DATA TYPE	NULL
TXTDEVICEID	VARCHAR2(30)	NOT NULL
TXTSTATUS	VARCHAR2(31)	NOT NULL
STARTTIME	DATE	NOT NULL
ENDTIME	DATE	NOT NULL
TIMEDIFF	NUMBER	NOT NULL
INTERVALSUP	NUMBER	NOT NULL
INTERVALCONF	NUMBER	NOT NULL
DENSITY	NUMBER	NOT NULL

Consider an interval I1 (01:00 – 03:00) and support of 01:00 is 5 and support of 03:00 is again 5. Consider another interval I2 (03:00 - 05:00) and support of 05:00 is 5:00. Intervals I1 and I2 can merge to form a new interval I3 (01:00 – 05:00). While forming such an interval I3, the support of the new interval becomes sum of support of the two intervals less the support of the overlapping time point i.e.

(Support of I3 = Support of I1 + Support of I2 – Support of Overlapping time point). In this case the overlapping time point is 3:00 and its support is 5. So, support of I3 becomes 10 + 10 – 5 = 15. In this way, "*Endsup*" enables to calculate the support of newly formed interval.

- ***FinalFrequentItems***: This table stores SIs that are selected in different stages of the algorithm. Its structure is similar to "*FirstLevel*" and "*Candidate*" except that it does not have the columns "*Startsup*" and "*Endsup*".

Table 4.7 COUNTSUPWEEK table

COUNTSUPWEEK TABLE		
COLUMN NAME	DATA TYPE	NULL
TXTDEVICEID	VARCHAR2(30)	NOT NULL
TXTSTATUS	VARCHAR2(31)	NOT NULL
DTTIME1	VARCHAR2(10)	NOT NULL
DTTIME2	VARCHAR2(15)	NOT NULL
DATETIME	DATE	NOT NULL
RECORDCOUNT	NUMBER	NOT NULL

- ***CountSupWeek***: This table stores support of time points in weekly periodicity. The column "*DTTIME1*" is used to store the unique time of the day, "*DTTIME2*" is used to store unique day of the week and "*DATETIME*" field is used to maintain the time order among all the records. "*DATETIME*" field is initially populated with current system date and then later changed to appropriate unique date that is obtained from another table "*WeekTable*".

Table 4.8 WEEKTABLE table

WEEKTABLE TABLE		
COLUMN NAME	DATA TYPE	NULL
DAYS	VARCHAR2(15)	NOT NULL
DATETIME	DATE	NOT NULL

- ***WeekTable***: This table stores all the days of the week and they are assigned a unique date in time-order.

4.2.2 Design of Algorithm

- Fold time series data
- Select unit length significant intervals
- Delete time points that have been identified as significant
- Perform time wrapping
- Form first level intervals
- Select significant first level intervals
- Delete first level intervals that have been identified as significant
- While (true) {
  - Generate candidate intervals
  - Select significant intervals from candidates
  - Delete candidates that have been identified as significant
  - }
- Delete redundant intervals introduced by time wrapping and subsumption

Figure 4.1 SIDQ algorithm

Figure 4.1 shows the SQL-based algorithm for SID. It starts with counting of support of distinct time points through "*Folding*". Distinct time points with their support are obtained in "*CountSup*" table and time points that meet the min-Conf criteria are selected as Unit SIs. They are inserted into "*FinalFrequentItems*" table and removed from "*CountSup*" table.

"*Time Wrapping*" is performed by replicating time points that falls below max-Len specified and assigning them next date. Then, all the distinct time points are

assigned current date. In this way, all the time points are maintained in time order so that intervals can be formed by taking difference of two time points and assuming that start time is always greater than end time. It is possible to form intervals by taking absolute difference of time points however, it leads to a large number of tuples (after joining two tables) and it is computationally expensive. So, all time points (replicated and initial distinct time points) are maintained in time order by assigning current date for initial distinct time points and next date for replicated time points during "*Time Wrapping*".

After time wrapping, all the time points are obtained in "*CountSupTemp*" in an increasing time order. Formation of first level intervals in case of SQL-based algorithm is separated into two distinct steps. In the first step, all the distinct time points that will lead to formation of first level intervals are identified and they are stored in "*DistinctInt*" table with their minimum time difference to the adjacent time point. In the second step, first level intervals are obtained by using the information stored in "*DistintInt*" table (i.e. start time of first level interval and their time difference). When "*DistinctInt*" and "*CountSupTemp*" tables are joined on this information, first level intervals with all their attributes are obtained and they are stored in "*FirstLevel*" table. Those first level intervals that are significant are inserted into "*FinalFrequentItems*" table and they are removed from "*FirstLevel*" table.

The remaining first level intervals will participate in expansion of intervals. Intervals are expanded by merging them with adjacent time intervals. Two intervals can merge only if end time of one interval is equal to start time of another interval.

Srinivasan in [3] has proposed two approaches for merging. In the first approach, only max-Len is used to validate new intervals. SID[1] algorithm uses this approach and basically checks whether interval length of new interval is less than max-Len specified. If not, newly formed interval cannot be considered as valid interval. In the second approach, besides max-Len, characteristics of the intervals like interval confidence and interval density is taken into consideration to validate new intervals. This approach basically means that two intervals can merge only if their confidence or density can be improved through merging and the length of new interval is less than max-Len. SID[n-1] and SID[n-2] use this approach. SID[1], SID[n-1] and SID[n-2] differ in the way intervals are merged for expanding. While n-1th level intervals are expanded with first level intervals in case of SID[1], they are expanded with intervals in the same level in case of SID[n-1]. In case of SID[n-2], the intervals in the level n-2 are expanded with intervals from the level n-1. SID[1], SID[n-1] and SID[n-2] have been implemented in SQL and they are referred as SIDQ[1], SIDQ[n-1] and SIDQ[n-2].

New intervals that meet the criteria of max-Len are known as candidate intervals and they are inserted into nth candidate table. From these candidate intervals, SIs are selected and inserted into "*FinalFrequentItems*" table. The expansion of intervals will continue until there are candidates generated in each iteration and there are intervals to be expanded at the end of each iteration. Once, the algorithm comes out of the loop of interval formation, redundant significant intervals that are introduced due to *Anomaly of Time Wrapping* and *Subsumption* is removed from "*FinalFrequentItems*" table.

4.2.2.1 Preprocessing

This is the first phase of the algorithm. The algorithm reads all the parameters specified by the user from the configuration file. Transaction data is folded to obtain the count of distinct time points. The following SQL is used for folding on daily basis and it uses function COUNT to obtain support of distinct time points (assumed to be in minute granularity). Grouping is done on deviceid, status and time of day and the combination of these three represent the distinct time point event. The folded data is ordered by time and inserted into "*CountSups*" table.

```
INSERT INTO countsups
SELECT txtdeviceid, txtstatus ,TO_CHAR(dttransdatetime,'HH24:MI'),COUNT(*)
FROM tbtransload
GROUP BY txtdeviceid, txtstatus, TO_CHAR(dttransdatetime,'HH24:MI')
ORDER BY txtdeviceid, txtstatus, TO_CHAR(dttransdatetime,'HH24:MI')
```

From the distinct time points, unit SIs are selected and inserted into "*FinalFrequentItems*" table using the following SQL in which time points with confidence greater than min-Conf are selected and they are assigned same start and end time. Length of the interval is assigned as 1 and support of the time point becomes the density for such intervals.

```
INSERT INTO finalfrequentitems
        SELECT txtdeviceid, txtstatus, TO_DATE(SYSDATE||' '||dttime, 'DD-MM-YY HH24:MI'),
                TO_DATE(SYSDATE||' '||dttime, 'DD-MM-YY HH24:MI'), 1,
                recordcount,TRUNC(recordcount/ NumDays,3), recordcount
        FROM countsups
        WHERE TRUNC(recordcount/NumDays, 3) >= Min-Conf
```

Once, unit SIs are selected, they are removed from "*CountSup*" table and the remaining time points are used for time wrapping. All time points that fall below max-

Len are selected and inserted into "*CountSupTemp*" table with day greater than the current date in order to create time-ordered data.

```
INSERT INTO countsuptemp
SELECT txtdeviceid, txtstatus, TO_DATE(SYSDATE+1||' '||dttime, 'DD-MM-YY HH24:MI'), recordcount
FROM countsups
WHERE dttime < 'Max-Len'
```

Then, all the time points in the "*CountSup*" table are again inserted into "*CountSupTemp*" table with current date.

```
INSERT INTO countsuptemp
SELECT txtdeviceid, txtstatus, TO_DATE(SYSDATE||' '||dttime,'DD-MM-YY HH24:MI'),  recordcount FROM countsups
```

After this, first level intervals are obtained by merging adjacent time points. Finding first level intervals from time points is a trivial problem in case of traditional programming language. All the distinct time points can be time ordered and two consecutive time points can be considered for the formation of first level intervals. If they meet the max-Len specified, then they can form first level intervals. But this is not the case when the distinct time points are in the form of rows of a table. Cursor allows looping through and considering two consecutive rows (assuming that all the rows are time ordered) at a time but it was inefficient. Join can be used to form intervals between time points but it can lead to the formation of large number of intervals. Consider the time points (1:00, 2:00, 3:00, and 4:00). With self-join, the first level intervals obtained would be (1:00-2:00, 1:00-3:00, 1:00-4:00, 2:00-3:00, 2:00-4:00, 3:00-4:00) but we are only interested in generating (1:00-2:00, 2:00-3:00 and 3:00-4:00) as the desired first level intervals. We tried to achieve this using function MIN in time difference so that only intervals that have shortest time difference are selected (i.e., only nearest points

will lead to the formation of first level interval). When MIN function is used, grouping on rest of the attributes is not possible with all the calculations involved. So, formation of first level intervals is divided into two steps:

```
INSERT INTO distinctint (
        SELECT t1.txtdeviceid,t1.txtstatus,t1.dttime, MIN(((t2.dttime-t1.dttime)*24*60)+1)
        FROM countsup t1, countsup t2
        WHERE t1.dttime < t2.dttime
        AND (((t2.dttime-t1.dttime)*24*60)+1) <=Max-Len
        AND t1.txtstatus = t2.txtstatus
        AND t1.txtdeviceid = t2.txtdeviceid
        GROUP BY t1.txtdeviceid ,t1.txtstatus, t1.dttime
)
```

The first step is to find distinct time points with their minimum time difference. This is achieved with a following SQL which populates "*DistinctInt*" table.

```
INSERT INTO firstlevel  (
SELECT t1.txtdeviceid, t1.txtstatus, t1.dttime, t2.dttime,
        ((t2.dttime-t1.dttime)*24*60)+1, t1.recordcount+t2.recordcount, t1.recordcount, t2.recordcount,
        TRUNC((t1.recordcount+t2.recordcount)/NumDays,3),
        TRUNC((t1.recordcount+t2.recordcount)/(((t2.dttime-t1.dttime)*24*60)+1),2)
FROM countsuptemp t1, countsuptemp t2, distinctint d1
WHERE  (((t2.dttime-t1.dttime)*24*60)+1) <= Max-Len
AND t1.dttime < t2.dttime
AND t1.txtdeviceid = d1.txtdeviceid
AND t1.txtstatus = d1.txtstatus
AND t1.txtdeviceid = t2.txtdeviceid
AND t1.txtstatus = t2.txtstatus
AND t1.dttime = d1.dttime
AND (((t2.dttime -t1.dttime)*24*60)+1) = d1.timediff
)
```

The second step is to find first level intervals by using information collected in "*DistinctInt*" table. It performs two joins between two copies of "*CountSupTemp*" and "*DistinctInt*". The query calculates all the measures for the interval (like interval support, density and interval confidence) using expressions and finds first level intervals. Support of the start time of interval is set as "*StartSup*" and support of end time of interval is set as "*EndSup*".

Once the first level intervals are obtained, SIs are selected and inserted into "*FinalFrequentItems*" table and they are removed from "*FirstLevel*" table. The remaining intervals will participate in interval expansion.

4.2.2.2 Interval Formation

First level intervals are expanded in interval formation phase. This takes place inside the loop. Intervals are expanded by merging them with adjacent intervals. Intervals can merge if end time of one interval is equal to start time of another interval. This condition is specified in the WHERE clause. In each iteration, a new candidate table is generated and this is populated using the following SQL:

```
INSERT INTO C1
        SELECT I1.txtdeviceid, I1.txtstatus, I1.starttime, I2.endtime, ((I2.endtime-I1.starttime)*24*60)+1,
                I2.intervalsup + I1.intervalsup - I2.startsup,I2.startsup, I2.endsup,
                TRUNC((I1.intervalsup + I2.intervalsup - I2.startsup)/NumDays,3),
                TRUNC((I1.intervalsup + I2.intervalsup - I2.startsup)/(((I2.endtime-I1.starttime)*24*60)+1),4)
        FROM firstlevel I1, firstlevel I2
        WHERE I1.endtime = I2.starttime
        AND ((I2.endtime-I1.starttime)*24*60)+1 < Max-Len
        AND I1.txtdeviceid = I2.txtdeviceid
        AND I1.txtstatus = I2.txtstatus
```

From these candidate intervals, SIs are selected and inserted into "*FinalFrequentItems*" table. Then, candidate table is checked to see whether any intervals are left for expanding. If there are intervals to be expanded, the loop iterates and repeats the same process. The loop stops if there are no new candidates generated. All SID approaches: SIDQ[1], SIDQ[n-1] and SIDQ[n-2] have the same flow except for the difference in merging of intervals to form candidate intervals. Till the generation of 1st level candidates, all SID suites of algorithms have the same flow and their difference can be observed after the 2nd level candidate generation.

```
INSERT INTO C2
    SELECT I1.txtdeviceid, I1.txtstatus, I1.starttime, I2.endtime,
        ((I2.endtime-I1.starttime)*24*60)+1,
        I2.intervalsup + I1.intervalsup - I2.startsup,I2.startsup, I2.endsup,
        TRUNC((I1.intervalsup + I2.intervalsup - I2.startsup)/NumDays,3),
        TRUNC((I1.intervalsup + I2.intervalsup - I2.startsup)/(((I2.endtime-I1.starttime)*24*60)+1),4)
    FROM C1 I1, firstlevel I2
    WHERE I1.endtime = I2.starttime
    AND ((I2.endtime-I1.starttime)*24*60)+1 < Max-Len
    AND I1.txtdeviceid = I2.txtdeviceid
    AND I1.txtstatus = I2.txtstatus
```

The above SQL generates 2nd level candidates for SIDQ[1] and it can be seen that n-1th level intervals are merged with first level intervals to form new candidate intervals.

The SQL given below generates 2nd level candidates for SIDQ[n-1] and it can be seen that n-1th level intervals are merged with intervals of the same level to form new candidate intervals. This SQL is different from one used for SIDQ[1] as there is one additional condition that is checked before two intervals are merged to form new

interval. This condition ensures that the densities or confidences of the merging intervals are in increasing order.

```
INSERT INTO C2
    SELECT I1.txtdeviceid, I1.txtstatus, I1.starttime, I2.endtime,
        ((I2.endtime-I1.starttime)*24*60)+1,
        I2.intervalsup + I1.intervalsup - I2.startsup, I2.startsup, I2.endsup,
        TRUNC((I1.intervalsup + I2.intervalsup - I2.startsup)/NumDays,3),
        TRUNC((I1.intervalsup + I2.intervalsup - I2.startsup)/(((I2.endtime-I1.starttime)*24*60)+1),4)
    FROM c1 I1, c1 I2
    WHERE I1.endtime = I2.starttime
    AND ((I2.endtime-I1.starttime)*24*60)+1 < Max-Len
    AND I1.txtdeviceid = I2.txtdeviceid
    AND I1.txtstatus = I2.txtstatus
    AND ((I1.intervalconf <= I2.intervalconf) or (I1.density <= I2.density))
```

The same condition is applied in case of SID[n-2] as well. The following SQL is used to generate 2^{nd} level candidates and in this case, n-2^{nd} level intervals merge with n-1^{th} level intervals to form new candidate intervals.

```
INSERT INTO C2
    SELECT I1.txtdeviceid, I1.txtstatus, I1.starttime, I2.endtime,
        ((I2.endtime-I1.starttime)*24*60)+1,
        I2.intervalsup + I1.intervalsup - I2.startsup, I2.startsup, I2.endsup,
        TRUNC((I1.intervalsup + I2.intervalsup - I2.startsup)/NumDays,3),
        TRUNC((I1.intervalsup + I2.intervalsup - I2.startsup)/(((I2.endtime-I1.starttime)*24*60)+1),4)
    FROM firstlevel I1, c1 I2
    WHERE I1.endtime = I2.starttime
    AND ((I2.endtime-I1.starttime)*24*60)+1 < Max-Len
    AND I1.txtdeviceid = I2.txtdeviceid
    AND I1.txtstatus = I2.txtstatus
    AND ((I1.intervalconf <= I2.intervalconf) or (I1.density <= I2.density))
```

In case of SIDQ[n-2], for the generation of 2nd level candidates, first level intervals are expanded with intervals in C1 and for the generation of 3rd level candidates, intervals in C1 are expanded with intervals in C2. The tables to be used for merging the intervals are controlled by a function, which provides appropriate tables based on the selection of "*Approach_Number*" in the configuration file.

4.2.2.3 Post-processing

Redundant SIs, generated due to *Anomaly of Time Wrapping* and *Subsumption*, are removed in this phase. Due to time wrapping, there may appear two intervals with same start time and end time as SIs on two different days. Only one should be retained. The following SQL selects significant intervals that are same in all respects except that they appear in two different days. MAX function is used to obtain maximum date of an interval which appears in two days and it is deleted.

```
DELETE FROM finalfrequentitems
WHERE (txtdeviceid, txtstatus, TO_CHAR(starttime,'dd-mm-yyyy'), TO_CHAR(starttime,'HH24:MI'))
IN (
        SELECT f1.txtdeviceid, f1.txtstatus, TO_CHAR(MAX(f1.starttime), 'dd-mm-yyyy'),
                TO_CHAR(f1.starttime,'HH24:MI')
        FROM finalfrequentitems f1, finalfrequentitems f2
        WHERE TO_CHAR(f1.starttime,'HH24:MI') = TO_CHAR(f2.starttime,'HH24:MI')
        AND TO_CHAR(f1.endtime,'HH24:MI') = TO_CHAR(f2.endtime,'HH24:MI')
        AND TO_CHAR(f1.starttime,'dd-mm-yyyy') != TO_CHAR(f2.starttime,'dd-mm-yyyy')
        GROUP BY f1.txtdeviceid, f1.txtstatus, TO_CHAR(f1.starttime,'HH24:MI')
)
```

Due to unit SIs and also in case of SIDQ[n-1] and SIDQ[n-2], a SI may subsume unit SI or another SI. So, these redundant SIs have to be removed to meet the

third definition of SIs. For every SI, the following SQL statement checks whether there are any other SI which subsumes it. If it finds those SIs, they are deleted.

```
DELETE FROM finalfrequentitems
WHERE (txtdeviceid, txtstatus, starttime, endtime)
IN (    SELECT t2.txtdeviceid, t2.txtstatus, t2.starttime, t2.endtime
        FROM finalfrequentitems t1, finalfrequentitems t2
        WHERE t2.starttime <= t1.starttime
        AND t1.endtime <= t2.endtime
)
```

4.2.3 Implementation of Weekly Periodicity

```
for (int i = 1; i < 7; i++){
        INSERT INTO weektable
                SELECT TO_CHAR (SYSDATE+i,'DAY'),SYSDATE+i  FROM DUAL
}
INSERT INTO weektable
        SELECT TO_CHAR(SYSDATE, 'DAY'), SYSDATE  FROM DUAL
```

For weekly periodicity, the algorithm uses same set of tables except for *Folding*. In case of weekly periodicity, when *Folding* is done, besides the time of the day, the day of the week information also has to be stored and furthermore, the time points in different days of the week have to be maintained in a time order. So, "*CountSupWeek*" table with additional field for storing day of the week is used for *Folding* and "*WeekTable*" is populated with the following SQL.

```
INSERT INTO countsupweek
        SELECT txtdeviceid, txtstatus , TO_CHAR(dttransdatetime,'HH24:MI'),
                TO_CHAR(dttransdatetime,'DAY'), SYSDATE, COUNT(*)
        FROM tbtransload
        GROUP BY txtdeviceid, txtstatus, TO_CHAR(dttransdatetime,'HH24:MI'), TO_CHAR(dttransdatetime,'DAY')
        ORDER BY txtdeviceid, txtstatus, TO_CHAR(dttransdatetime,'HH24:MI'), TO_CHAR(dttransdatetime,'DAY')
```

Folding is performed in weekly periodicity with the above SQL statement, which uses COUNT function with grouping on event (i.e., the combination of deviceid, status, day of week and time of day). Then, "*CountSupWeek*" table is updated with corresponding date for their days by joining with "*WeekTable*" as follows:

```
UPDATE countsupweek cw
        SET datetime = ( SELECT TO_DATE(w.dttime||' '||cw.dttime1,'DD-MM-YY HH24:MI')
                        FROM weekTable w
                        WHERE cw.dttime2 = w.days
                )
```

Unit SIs are selected from "*CountSupWeek*" table with the following SQL and inserted them into "*FinalFrequentItems*" table. After time points have been selected as SIs, they are removed from "*CountSupWeek*" table.

```
INSERT INTO finalfrequentitems
        SELECT txtdeviceid, txtstatus, datetime, datetime, 1, recordcount,
                TRUNC(recordcount/numDays,3), recordcount
        FROM countsupweek
        WHERE TRUNC(recordcount/numDays, 3) >=Min-Conf
```

Time Wrapping is performed in case of weekly periodicity with the following SQL. It duplicates those time points that fall below max-Len specified and "*First-Day*" of the week. The first day of the week is the first day of the data collection and it is inferred from the data.

```
INSERT INTO countsuptemp
        SELECT txtdeviceid, txtstatus, datetime, recordcount
        FROM countsupweek
        WHERE dttime1 < Max-Len
        AND TRIM(dttime2) = First-Day
```

Rest of the algorithm is similar as in case of daily periodicity, which has been described earlier.

4.3 Existing Main Memory Approach [SIDH]

Srinivasan [3] implemented SID algorithms in main memory using data structures to store intermediate results. Important data structures used are:

- InputStruc – data structure to store the inputs

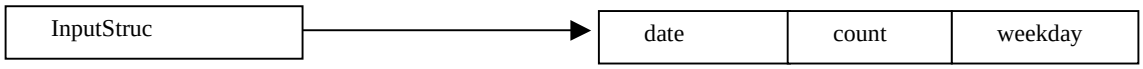

Figure 4.2 INPUTSTUC data structure

- InputMavhome – domain specific data structure
- OutputStruc – data structure to store the outputs

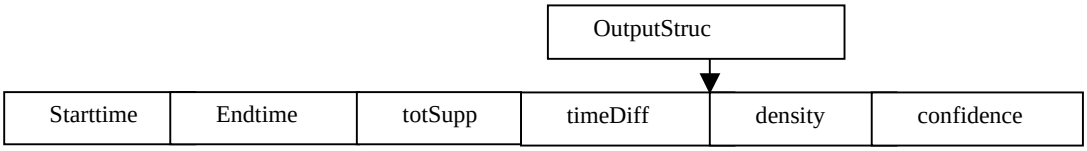

Figure 4.3 OUTPUTSTRUC data structure

```
SELECT TO_CHAR(dttransdatetime,'HH24:MI'), COUNT(*)
FROM tbtransload
WHERE txtdeviceid = 'DeviceId'
AND txtstatus = 'Status'
GROUP BY txtdeviceid, txtstatus, TO_CHAR(dttransdatetime, 'HH24:MI')
ORDER BY txtdeviceid, txtstatus, TO_CHAR(dttransdatetime, 'HH24:MI')
```

First, the algorithm finds out distinct events (combination of deviceid and status) and stores them in main memory data structures. For each deviceid and status, data is fetched from the database using the above SQL statement.

"*InputFormat*" and "*IncrementInterval*" are the two most important classes. "*InputFormat*" class reads the configuration file, identifies the input parameters, and finds count for each event. "*IncrementInterval*" finds first level intervals and then finds SIs. Post-processing phase is carried out for *Time Wrapping*. SIs having start time greater than end time are assigned different dates for start time and end time within the same interval. This is done to represent formation of intervals spanning two days. This is done with a single SQL as follows:

update TBSIGINTERVAL set dtendtime = dtendtime + 1 where dtstarttime >dtendtime

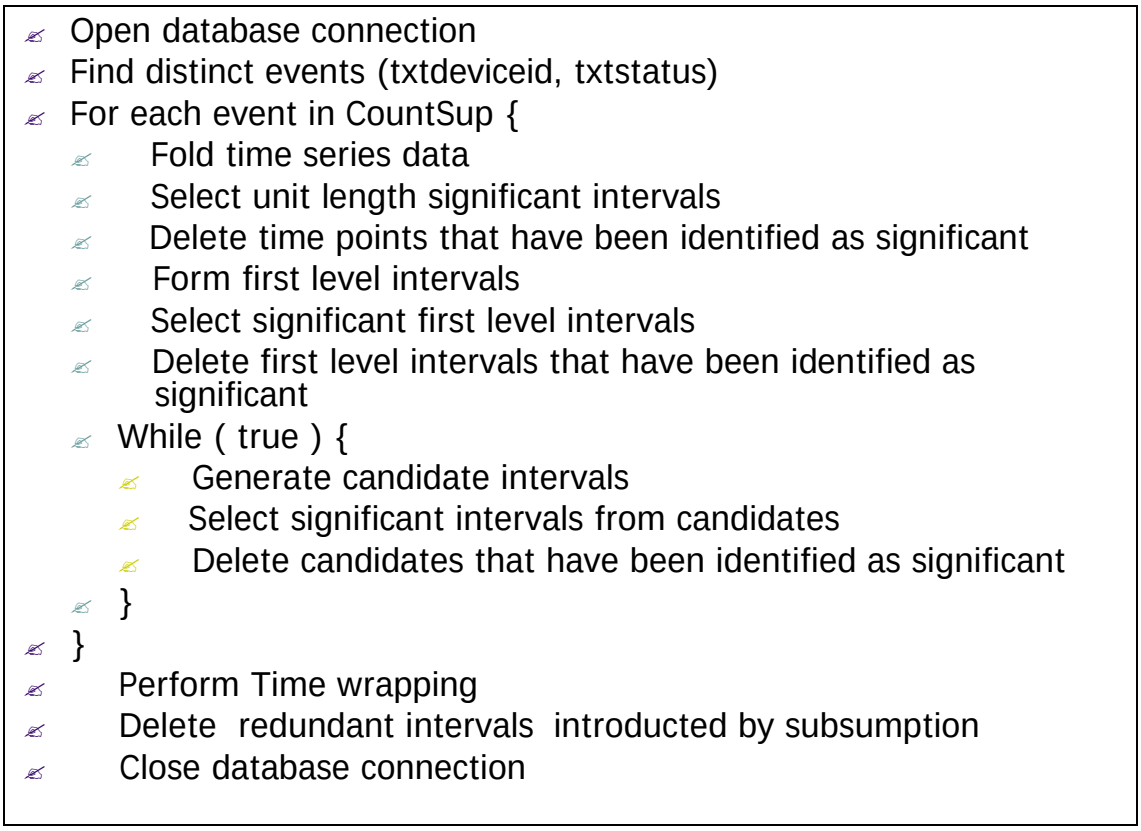

Figure 4.4 SIDH algorithm

The basic flow of the algorithm has been maintained, however a number of changes has been made in the algorithm to incorporate the revised definitions of SIs. This revised main memory algorithm which reads data from database is named as Significant Interval Discovery – Hybrid (SIDH).

4.4 Refined Main Memory Approach [SIDM]

Previously described SIDH algorithm reads data from a database using SQL statements although it is a main memory algorithm. Furthermore, it executes the same SQL statement multiple times to fetch data for different events from the database. The SID algorithms find SIs for each event separately. The performance of the main

memory algorithm is improved when it read data from flat files. So, a file reading interface has been implemented in lieu of the SQL interface, which reads data from .csv files. This makes it a pure main memory algorithm and hence the nomenclature SIDM as opposed to SIDH used for the hybrid version. A new data structure "*CountSup*" has been created to maintain the count of all the distinct time points. It uses a double Hashtable, one for distinct events (deviceid and status) and another Hashtable for unique time points of the distinct event. The data structure is shown below.

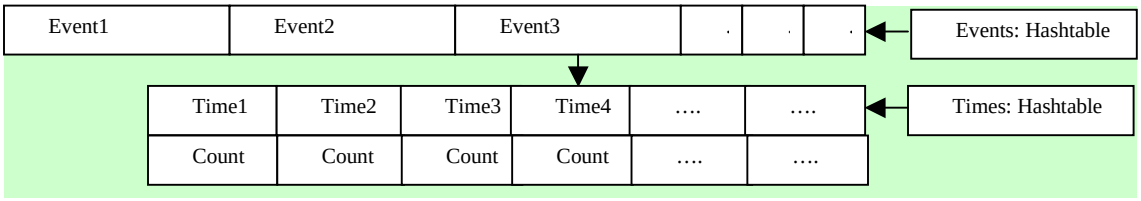

Figure 4.5 COUNTSUP data structure

As data is read from the file, the data structure is updated continuously to record count of the distinct time point events. Once the file is read completely, time points stored in hashtable are sorted in time order. This is done by first converting Hashtable into Collections type and using sort function of collections. Then, each distinct event is processed sequentially as before. The implementation has been refined as per the revised definitions and some changes have been made in the Post-processing phase to prune redundant SIs to satisfy the *Subsumption* condition. The revised algorithm is shown below.

- Open data file
- Create a double hashtable (CountSup) with (txtdeviceid, txtstatus) as key
- While (!EOF){
  - Populate CountSup
- }
- Sort CountSup in time order
- For each key in CountSup {
  - Select unit length significant intervals
  - Delete time points that have been identified as significant
  - Form first level intervals
  - Select significant first level intervals
  - Delete first level intervals that have been identified as significant
  - While (true) {
    - Generate candidate intervals
    - Select significant intervals from candidates
    - Delete candidates that have been identified as significant
  - }
- }
- Perform time wrapping
- Delete of redundant intervals introducted by subsumption

Figure 4.6 SIDM algorithm

CHAPTER 5

EXPERIMENTAL RESULTS

5.1 Experimental Setup

Experiments were conducted using a Linux cluster where each node is a dual processor of 2.4GHz 533MHz P4 Xeon CPUs with 512KB cache 533MHz FSB and Intel E7501 Plumas chipset. The nodes are running Redhat 7.3 with kernel extensions for GFS Linux version 2.4.20-18.7.gfs520p002smp as the operating system. Each node has 1GB of main memory and the operating system is scheduling processes on both the processors. No other processes were allowed to run on these machines when the experiments were conducted except for the routine database operation and operating system processes. Oracle 9i 9.2.0.1.0 database is used. A configuration file for the program is provided for the user to specify the values for different parameters. The program is written in Java, which generates SQL statements dynamically based on the user specific configuration. Java HotSpot (TM) Client VM (build 1.4.2_03-b02, mixed mode) is used to run java program. The program uses JDBC connection to connect to Oracle database.

Data set generated by the MavHome is used for the experiments. 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. Synthetic datasets were created to verify the correctness of the algorithm in terms of meeting revised

definitions of Significant Intervals and boundary conditions (folding of data, time wrapping, generation of first level intervals, interval expansion, anomalies of time wrapping, Subsumption, disjoint significant intervals, overlapping significant intervals and unit length significant intervals). The output of SIDQ is compared against the output of SIDM to verify the correctness of both the algorithms.

Number of Transactions in (thousand)	13.42	26.84	40.26	53.68	67.1	80.52	93.94	107.36	120.78	134.2	147.62	161.04
Number of devices	1	2	3	4	5	6	7	8	9	10	11	12
Experiment Number	1	2	3	4	5	6	7	8	9	10	11	12

Number of Transactions in (thousand)	174.46	201.78	403.56	807.12	1614.24	3228.48	6456.96	9685.44
Number of devices	13	15	30	60	120	240	480	720
Experiment Number	13	14	15	16	17	18	19	20

Figure 5.1 Data set used for scaling experiments

Experiments on scalability are meant to test the ability of the algorithm to handle large volumes of data. Synthetic data is generated for these experiments based on the data obtained from MavHome. The maximum size of the data generated has 9685440 transactions and there are 720 distinct events in this dataset. The configuration used for all the scaling experiments are: min-Conf = 0.57, max-Len = 40, numDays = 92, granularity = min, periodicity = daily. The experiments were conducted multiple times and average of the three runs is taken. Time taken is measured by the System. currentTimeMillis() function provided by JAVA.

5.2 Comparison of SIDH and SIDM

As shown in Figure 4.6 and Figure 4.4, the difference between SIDH and SIDM is that in case of SIDH, folding is performed inside the loop and the same SQL query is executed for each event. This translates to scanning the entire table multiple times once for each query. So, SIDH is slower as compared to SIDM. In case of SIDM, transactions are available in the form on .csv file. The algorithm reads the file only once and populates the data structures to hold the count of all the distinct time points. In case of SIDH, database connection and execution of query is an overhead which has been replaced in SIDM with the overhead of maintaining the data structure to hold the support of distinct time points.

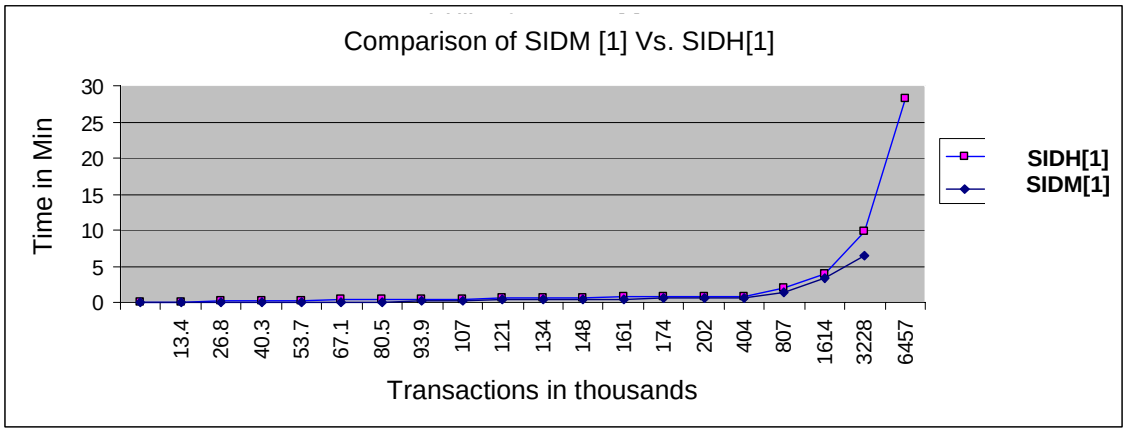

Figure 5.2 Comparison of SIDM[1] and SIDH[1]

When SIDH[1] and SIDM[1] were run for different sizes of data with the same configuration, it was observed (shown in Figure 5.2) that the performance of the algorithm increased marginally when a file reading interface was introduced. However,

it decreased the size of the largest data set it can process. The experiments were conducted on the dataset as shown in Figure 5.1.

The experiments confirmed our presumption that main memory algorithm can be improved with a file reading interface to replace the recurring SQL statement.

5.3 Effect of max-Len and min-Conf

The number of significant intervals generated by SID algorithm is affected by the values chosen for max-Len and min-Conf. Max-Len specifies the length of an interval which basically translates to the number of time points that can converge to form an interval. As the max-Len is increased, more number of time points will converge to form an interval and consequently, the support of the interval grows. This leads to detection of more number of significant intervals. Min-Conf is used to select significant intervals from intervals generated. When min-Conf is assigned a small value, more number of intervals will qualify as significant intervals and when it is increased, the number of significant intervals detected decreases.

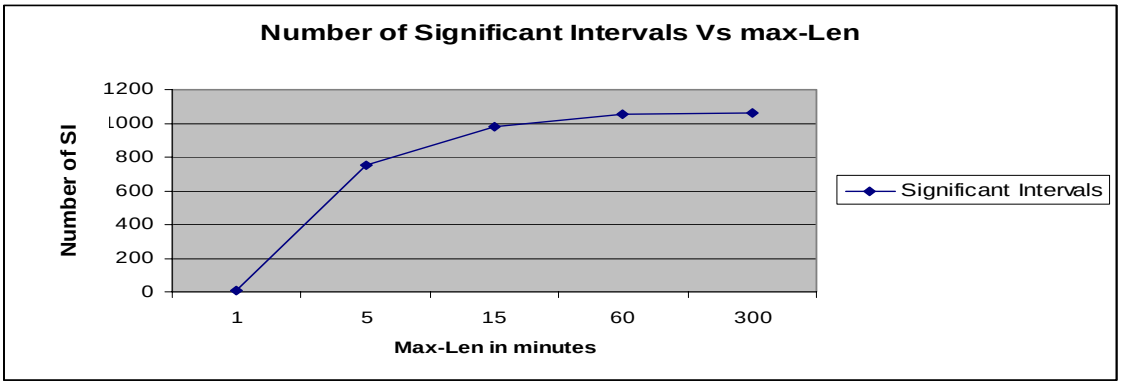

Figure 5.3 Number of significant intervals vs max-Len

An experiment is conducted to observe the number of significant intervals generated by SIDQ[1] algorithm as max-Len and min-Conf are changed. This is conducted on the first dataset of Figure 5.1. In the first experiment as shown in Figure 5.3, max-Len is increased keeping the rest of the parameters same. The rest of the parameters used are: min-Conf = 0.3, numDays = 92, granularity = min and periodicity = daily. It is observed that the number of significant intervals generated increased as max-Len is increased but the number tends to be constant after some value. SID algorithm finds tightest significant intervals only and the number of tightest significant intervals is a constant for any dataset for a particular min-Conf irrespective of max-Len. As max-Len is increased, more number of tightest significant intervals qualifies as significant intervals but this number is tending towards the maximum value for that particular dataset and min-Conf.

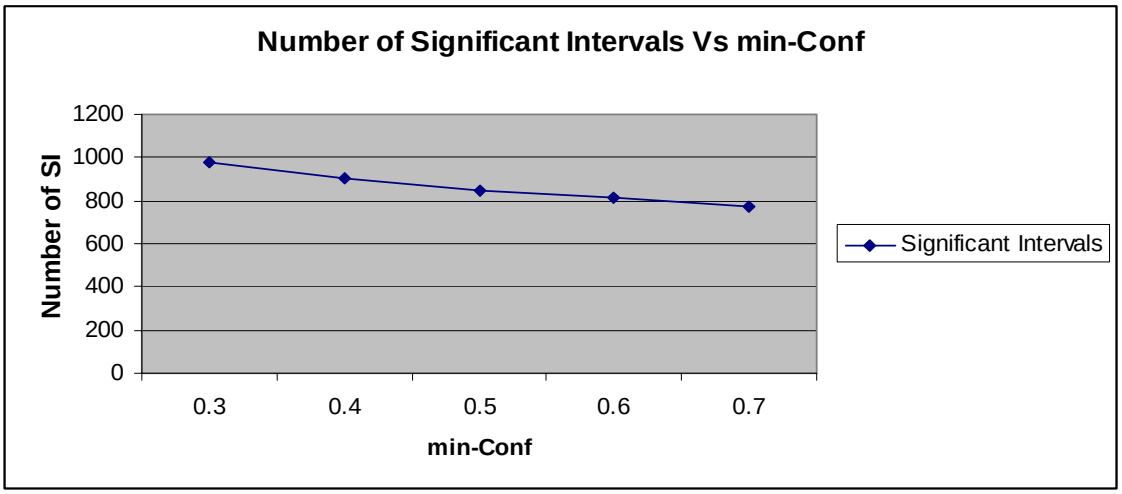

Figure 5.4 Number of significant intervals vs min-Conf

In the second experiment, max-Len is kept constant while min-Conf value is increased and the number of significant intervals generated is plotted against different values of min-Conf as shown in Figure 5.4. It was observed that as the min-Conf value is increased, the number of significant intervals generated keep on decreasing. This is because when min-Conf is increased, this is only restricting the selection of significant intervals but the overall support of the intervals are not increasing at all as max-Len is kept constant. So, it is only natural that if more number of significant intervals are obtained for less value of min-Conf, less number of significant intervals will be obtained for large value of min-Conf. Significant intervals are still obtained when min-Conf exceeds 1 as interval confidence for an interval is not associated with probability but it is just a ratio of total support of an interval and the number of units (days or weeks) of data collection.

The experiments confirmed our belief that max-Len and min-Conf affects the number of significant intervals discovered. So, it is important to choose those parameters intelligently.

5.4 Analysis of SIDQ[1]

SQL-based algorithm is implemented with various SQL statements. It is interesting to observe how various SQL statements of the algorithm affect the overall performance of the algorithm. We have used Join in a number of SQL statements. Finding distinct time points, finding first level intervals and finding candidate intervals are the SQL statements that use Join. While SQL statements for finding distinct time points and candidate generation uses join between two tables, SQL statement for

generation of first level use join between three tables. So, these statements should take the most amount of time as they have to process a large number of tuples.

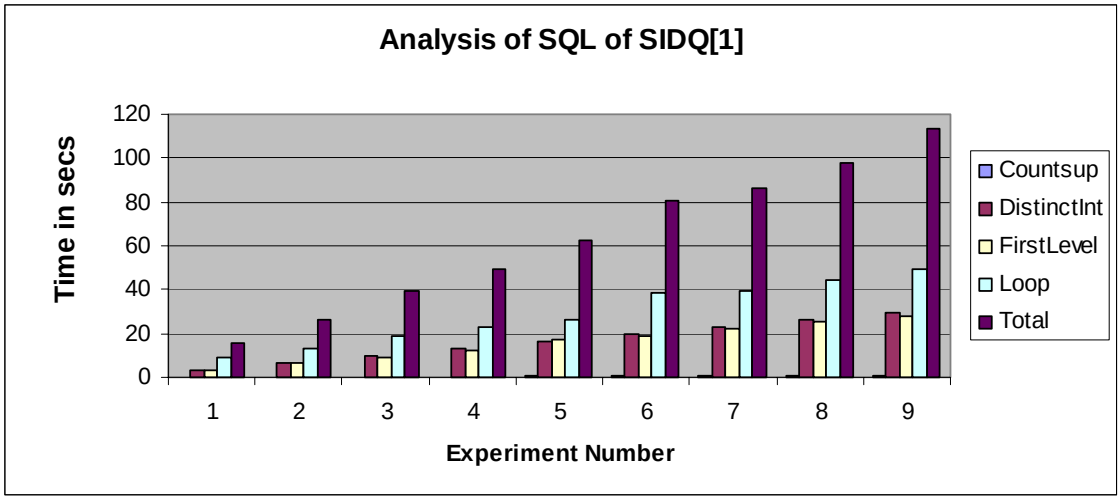

Figure 5.5 Time taken by major steps of SIDQ [1]

An experiment was conducted to analyze the performance of various steps of the SIDQ[1] algorithm. Dataset and configuration used for the experiment is same as used for the scaling experiments. Time taken to execute each query was noted and it was observed that generation of "*CountSup*", generation of "*DistinctInt*", generation of "*Firstlevel*" and generation of candidates took the most time. So, we have plotted the time taken by these SQL statements only in Figure 5.5. Experiments were conducted with different sizes of data and each experiment number corresponds to a data size as explained in Figure 5.1. It was observed that SQL statement for support counting took almost the same time with no significant change as the data size is increased. However, time taken by SQL statements for generation of "*DistinctInt*" and "*FirstLevel*" intervals

increased with the increase in data size. Time taken by these two SQL statements is significantly more as compared to the time taken for all iterations combined.

The experiments confirmed out belief that a SQL statement with join is affecting the overall performance of the SQL-based algorithm and optimization of these SQL statements will have a positive effect on the overall performance of the SQL-based algorithms.

5.5 Experiment for Scaling

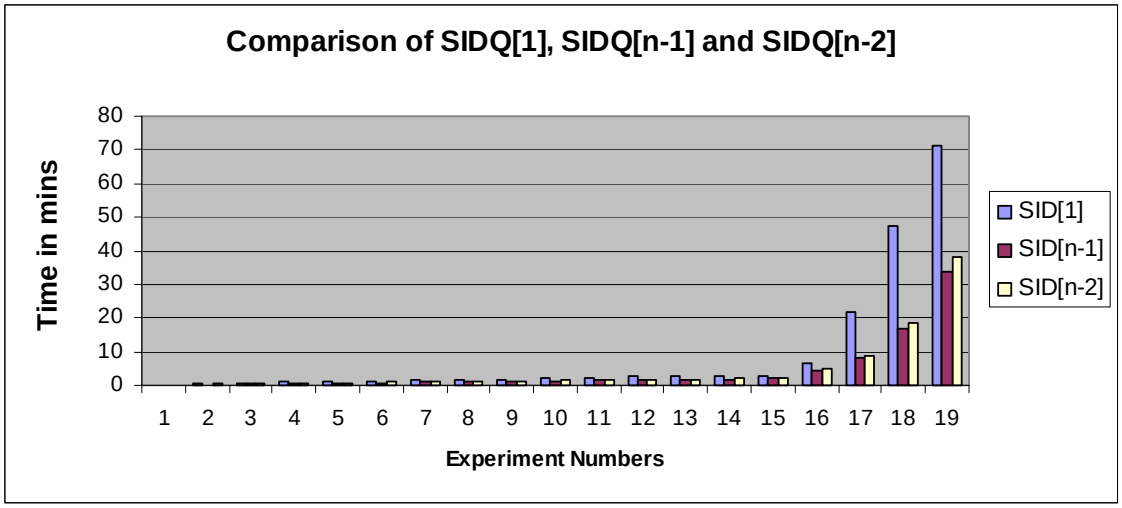

Figure 5.6 Comparisons of SIDQ [1], SIDH [1] and SIDM [1]

A number of experiments were conducted to observe the performance and scalability of various SID approaches. Comparison were made between various SID approaches implemented in SQL [SIDQ], implemented in main memory with file reading interface [SIDM] and implemented in main memory with database connection [SIDH]. Algorithms implemented in main memory are supposed to be faster compare to SQL-based as they used specialized data structures and optimized algorithms. Time

taken by the execution of the algorithms formed the basis of comparison. The experiments were conducted on the dataset shown in Figure 5.1.

Figure 5.6 shows the comparison of SIDQ[1], SIDH[1] and SIDM[1] algorithms. While SIDM[1] algorithm is observed to be fastest among all, it is not very scalable as it failed to process data beyond 18th experiment. SIDH[1] was a bit scalable compared to SIDM[1] but it also has a limitation on scalability. However, SIDQ[1] continued to process data for very large data sizes.

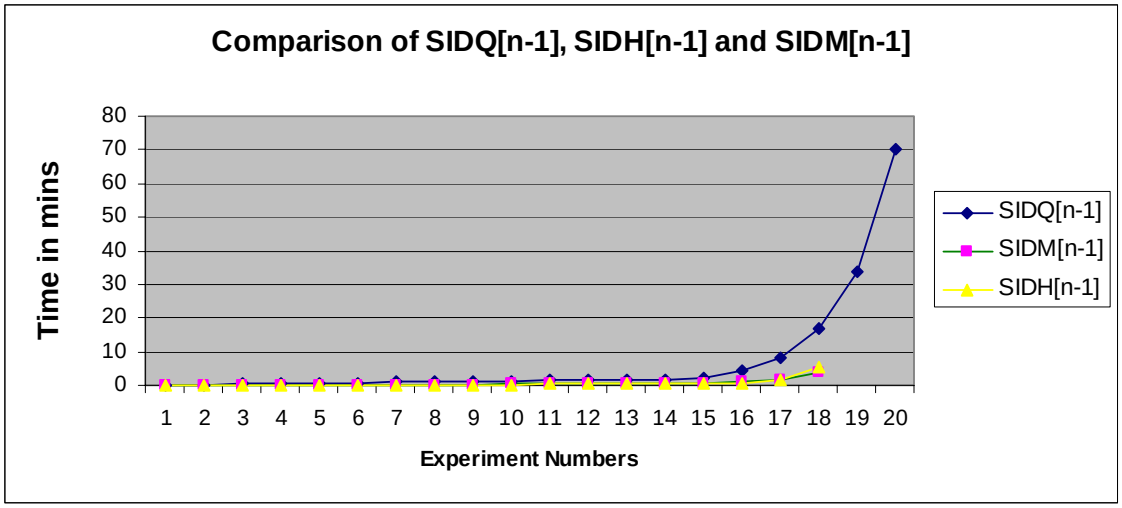

Figure 5.7 Comparison of SIDQ[n-1], SIDH[n-1] and SIDM[n-1]

Figure 5.7 shows the comparison of SIDQ[n-1], SIDH[n-1] and SIDM[n-1] algorithms and Figure 5.8 shows the comparison of SIDQ[n-2], SIDH[n-2] and SIDM[n-2]. In both the cases, main memory algorithms were observed to be faster compared to SQL-based while SQL-based algorithms were observed to be scalable.

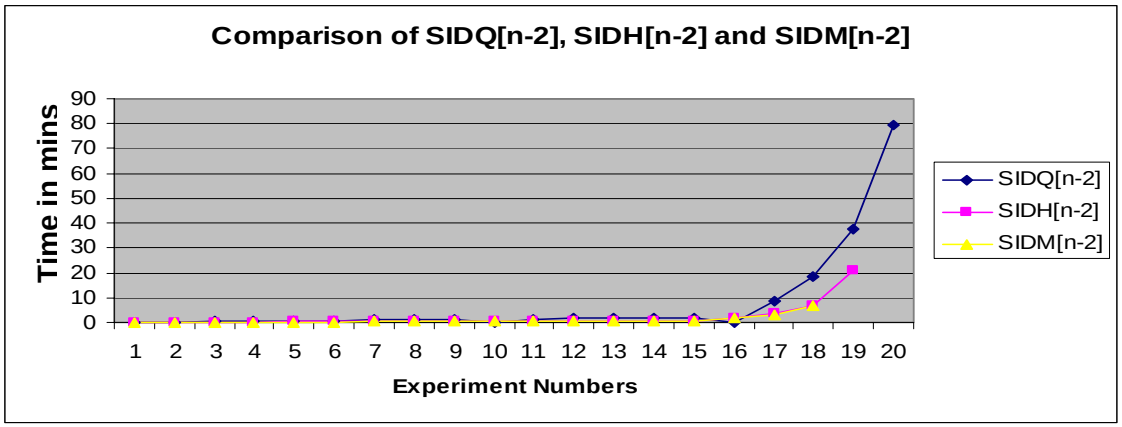

Figure 5.8 Comparison of SIDQ[n-2], SIDH[n-2] and SIDM[n-2]

The experiments confirmed that main memory algorithms are faster but they are not scalable while SQL-based algorithms are slower compared to main memory but they are scalable. So, based on the performance and scalability requirement, suitable algorithm can be chosen to find significant intervals from time-series data.

5.6 Comparison of SIDQ[1], SIDQ[n-1] and SIDQ[n-2]

Srinivasan in [3] has made the comparison of SID suite of algorithms in terms of accuracy (deviation & coverage), extra-fit and performance (number of iterations, time taken). Accuracy is expressed in terms of deviation and coverage. Coverage is defined as the set of all distinct points included within the significant intervals produced by Naïve. The amount, by which SID algorithm extends the output interval for the same measure level as Naïve, is referred to as extra-fit. Percentage deviation indicates the amount of extra-fit produced by SID variants as compared to Naïve. Comparison of suites of SID suites of algorithms is made and it is found that these algorithms are comparable in terms of coverage.

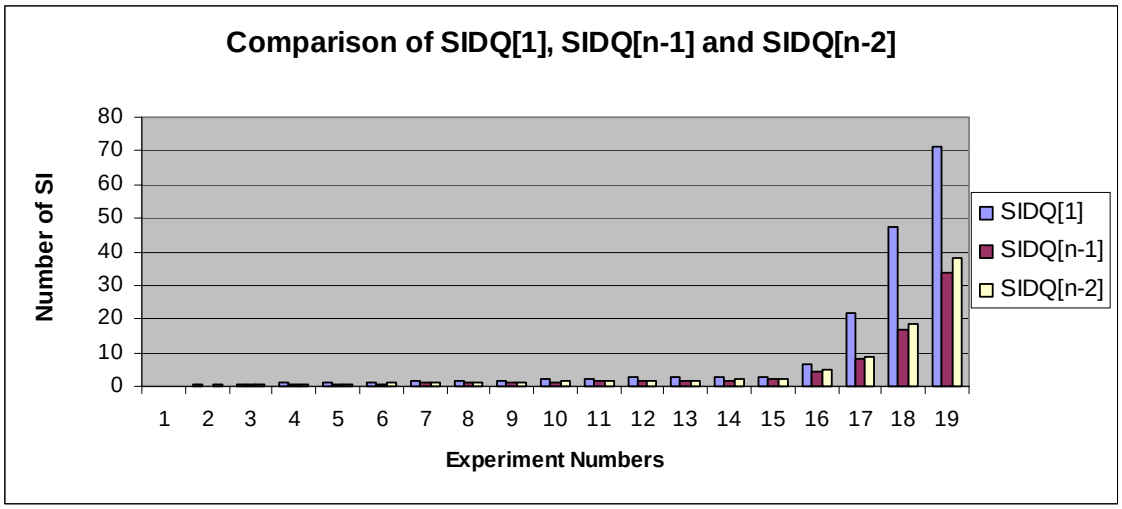

Figure 5.9 Comparison of SIDQ[1], SIDQ[n-1] and SIDQ[n-2]

SIDQ[1], SIDQ[n-1] and SIDQ[n-2] follow the new definitions of significant intervals but they use the same merging criteria and logic as explained in [3]. So, SIDQ[1] should be slower compared to SIDQ[n-1] and SIDQ[n-2] as expansion of intervals take place slowly while SIDQ[n-1] should be the fastest as expansion of intervals take place quickly in this case.

Figure 5.9 shows comparison of SIDQ[1], SIDQ[n-1] and SIDQ[n-2] in terms of performance and it is observed that SIDQ[1] takes a long time as compared to SIDQ[n-1] and SIDQ[n-2]. Not much difference was observed between SIDQ[n-1] and SIDQ[n-2] but SIDQ[n-1] is almost half as fast as SIDQ[1].

CHAPTER 6

CONCLUSION AND FUTURE WORK

In many application domains, intervals provide more information on the characteristics of the data as compared to time points. This thesis presents improvements in discovery of intervals for prediction. Several implementations of SID approaches are compared with each other to show how SQL-based approaches are scalable as compared to main memory approaches.

The current implementation of the algorithm works for Day and Week periodicity. However, time can be represented as a concept hierarchy, such as: Day->Week->Quarter->Financial Year. The ability of the algorithm to zoom in and out with different periods along the time hierarchy will be useful for the efficient analysis of data for pattern and trend detection. So, it would be useful to generalize the algorithm to work for different periodicity with the ability to drill-down and roll-up over the periodicity hierarchy. Further, SID algorithms currently identify significant intervals for minutes and seconds granularity. Depending upon the application domain and user requirement, the user may want to find significant intervals for other granularities such as hours or days. It would be beneficial to make this an interactive process. The idea in interactive mining is that an end user be allowed to query the database for SI at differing values of interval-length and confidence. The goal would be to allow such interaction without excessive I/O or computation. The authors in [13] maintain pre-processed

summaries that can quickly respond to online queries. Possibility of generating pre-processed summaries for interactive mining can be explored.

The current algorithm generates a lot of significant intervals that are overlapping. The time point may appear in a number of significant intervals. Although it indicates that the time point is significant, it makes it difficult for a user to associate an interval for that time point. Some metrics need to be formulated so that it will be possible to distinguish between different overlapping intervals that include the same time point. Also, it would be useful to order the significant interval using some user (or system) defined measure in order to better understand them.

APPENDIX A

INTRODUCTION TO SQL

Introduction to SQL

The relational model from which SQL draws much of its conceptual core was formally defined in 1970 by Dr. E. F. Codd in [12]. System/R project began in 1974 and developed SEQUEL or Structured English Query Language, which later revised as SEQUEL/2 and renamed as "SQL" with the inclusion of multi-table and multi-user features. SQL is used to create, maintain & query relational databases. A fundamental difference between SQL and standard programming languages is that SQL is non-procedural.

SQL was first standardized in 1986 by the American National Standards Institute (ANSI). Since then, it has been formally adopted as an International Standard by the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC). Although SQL is both an ANSI and an ISO standard, many database products support SQL with proprietary extensions to the standard language.

After 1986, a revised standard known commonly as SQL-89 or SQL1 was published in 1989. Due to partially conflicting interests from commercial vendors, much of the SQL-89 was intentionally left incomplete, and many features were labeled implementer-defined. SQL-92 significantly increases the size of the original 1986 standard to include a schema manipulation language for modifying or altering schemas, schema information tables to make schema definitions accessible to users, new facilities for dynamic creation of SQL statements, and new data types and domains. Other new SQL-92 features include outer join, cascade update and delete referential actions, set

algebra on tables, transaction consistency levels, scrolled cursors, deferred constraint checking, and greatly expanded exception reporting. SQL-92 also removes a number of restrictions in order to make the language more flexible and orthogonal. The major features that are introduced in SQL-99 are regular expression matching, recursive queries, triggers, non-scalar types and some object-oriented features. Major features in SQL-2003 are: XML-related features, *window functions*, standardized sequences and columns with auto-generated values (including identity-columns).

The three standards that matter today are:

1. SQL-92
2. SQL-99
3. SQL2003

Reasons for Using SQL-92:

1. **Compatibility:** This is the direct fallout of the previous point regarding standards. Since well-defined and established standards exist, and if the databases adhere to those, then portability from one SQL database to another is a trivial matter. Further, an SQL database conforming to set standards can be easily accessed by third party softwares and application tools. This will facilitate the development of quality applications and solutions around SQL databases.
2. **No coding required:** By using standard SQL it should be easier to move applications between different database systems without the need to rewrite a substantial amount of code.

3. **Adhoc mining:** SQL-aware data mining systems have the ability to support ad-hoc mining, i.e., allow mining arbitrary query results from multiple abstract layers of database systems or Data Warehouses.

APPENDIX B

CONFIGURATION FILE

```
# Different Appraoches can be used for SID
# 1: SID[1]
# 2: SID[n-1]
# 3: SID[n-2]
Approach_Number=3
#
# Req_Confidence is mandatory
Req_Confidence=.57
#
# Interval_Length: It specifies the intevals of length that the user is interested in
# for daily basis, the maximum length that can be specified with second's granularity is 24*60*60
# for weekly basis, the maximum length that can be specified with minute's granularity is 7*24*60*60
Interval_Length=40
#
# Time Granularity specifies the granularity of time for which the information is collected.
# it can be seconds, minutes or hours.
granularity=min
#for minutes
#granularity=sec
#for seconds
#granularity=hr
#for hours
# Period: This specifies the time period to be used for folding the time-series data to a particular period.
#period=weekly
period=daily
#
# Number of days is the total number of days for which data is collected.
Number_of_Days=92
```

REFERENCES

[1] 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. Montreal, Canada: p. 210--215, 1995.

[2] Cook, D.J., et al., MavHome: An Agent-Based Smart Home. In Proc of the Conference on Pervasive Computing, 2003.

[3] Srinivasan, A., Significant Interval and Episode Discovery in Time-Series Data, MS Thesis. CSE department, http://www.cse.uta.edu/Research/Publications/Downloads/CSE-2003-39.pdf, The University of Texas at Arlington. 2003.

[4] Bohlen, M. H., R. Busatto, and C. S. Jensen., Point- Versus Interval-based Temporal Data Models. In IEEE Data Engineering, 1998.

[5] Villafane, R., K. A. Hua, D. Tran and B. Maulik., Mining Interval Time Series. In Int'l Conference on Data Warehousing and Knowledge Discovery, 1999.

[6] Srinivasan, A. and S. Chakravarthy, Discovery of Interesting Episodes in Sequence Data. In PAKDD Workshops, May 2004, Sydney, Australia.

[7] Sarawagi, S., S. Thomas and R. Agrawal, Integrating Association Rule Mining with Relational Database Systems: Alternatives and Implications. Data Mining and Knowledge Discovery, 4, 89-125, 2000.

[8] Rantzau, Ralf, Frequent Itemset Discovery with SQL Using Universal Quantification. In Proc of the International Workshop on Database Technologies for Data Mining (DTDM); Prague, Czech Republic, March 2002.

[9] Ordonez, Carlos, Programming the K-means clustering algorithm in SQL. In Data Mining and Knowledge Discovery, 2004, 823-828

[10] Xuequn, S., Sattler Kai-Uwe and Geist Ingolf, SQL-based Frequent Pattern Mining with FP-growth. In Proc. 15th Int. Conference on Applications of Declarative Programming and Knowledge Management, Berlin, Germany, 2004.

[11] John F. Roddick and S. Myra, A Survey of Temporal Knowledge Discovery Paradigms and Methods. In IEEE Transactions on knowledge and data engineering, Vol 14. No. 14, July/August 2002.

[12] Codd, E. F., "A Relational Model of Data for Large Shared Data Banks", *Communications of the ACM,* Vol. 13, No. 6, June 1970, pp. 377-387.

[13] Srinivasan Parthasarathy, Mohammed J. Zaki, Mitsunori Ogihara, Sandhya Dwarkadas, "Incremental and Interactive Sequence Mining", 8th International Conference on Information and Knowledge Management , pp 251-258, Kansas City, MO, November 1999

[14] Thomas, S. and Chakravarthy, S. 1999. Performance Evaluation and Optimization of Join Queries for Association Rule Mining. In Proceedings of the First international Conference on Data Warehousing and Knowledge Discovery M. K. Mohania and A. M. Tjoa, Eds. Lecture Notes In Computer Science, vol. 1676. Springer-Verlag, London, 241-250.

[15] Thomas, S., 1998. Architectures and Optimizations for Integrating Data Mining Algorithms with Database Systems, PhD Dissertation. CSE department, The University of Florida, Gainesville

[16] Beera, R., Relational Database Algorithms and Their Optimizations for Graph Mining, MS Thesis. CSE department, http://itlab.uta.edu/ITLABWEB/Students/sharma/theses/Ramji.pdf The University of Texas at Arlington. 2003.

[17] Kona, H. V., Association Rule Mining over Multiple Databases: Partitioned and Incremental Approaches, MS Thesis. CSE department, http://itlab.uta.edu/ITLABWEB/Students/sharma/theses/Hima.pdf The University of Texas at Arlington. 2003.

[18] Mishra, P., Performance Evaluation and Analysis of SQL-based Approaches for Association Rule Mining, MS Thesis. CSE department, http://itlab.uta.edu/ITLABWEB/Students/sharma/theses/Pratyush_ThesisReport.pdf, The University of Texas at Arlington. 2003.

[19] Balachandran, R., Relational Approach to Modelling and Implementing Subtle Aspects of Graph Mining, MS Thesis. CSE department, http://itlab.uta.edu/ITLABWEB/Students/sharma/theses/Pratyush_ThesisReport.pdf, The University of Texas at Arlington. 2003.

[20] Miller, R. J., Y. Yang, Association Rules over Interval Data. In Proc of the 1997 ACM SIGMOD international conference on Management of data, 452 – 461, 1997.

[21] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In Proc. of the 1996 ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD '96), Montreal, Canada, June 1996.

[22] Srikant, R. and R. Agrawal, Mining Sequential Patterns: Generalizations and Performance Improvements. In 5th Intl. Conf. Extending Database Technology (EDBT). Avignon, France, 1995.

BIOGRAPHICAL INFORMATION

Sunit Shrestha was born on November 28, 1975 in Kathmandu, Nepal. He received his Bachelor of Technology degree in Electronics and Communication Engineering from Regional Engineering College, Kakatiya University, Warangal, Andhra Pradesh, India in June 1999. He worked for World Distribution Nepal Pvt. Ltd from August 1999 to July 2002 as a software engineer. In the fall 2002, he started his graduate studies in Computer Science and Engineering at The University of Texas, Arlington. He received his Master of Science in Computer Science and Engineering from The University of Texas at Arlington in August 2005. His research interests include Data Stream Management System, Sensor Database, Data Mining and Business Intelligence.