AUT | COMPUTING + MATHEMATICAL SCIENCES

Novel applications of Association Rule Mining- Data Stream Mining

Omkar Vithal Kadam
(Student ID: 0787047)

This thesis is submitted as part of Degree of Masters of Computers and Information Sciences at the Auckland University of Technology

August 2009

Table of Contents

List of Abbreviations 6

List of Tables 7

List of Figures 8

Abstract 9

Chapter 1 Introduction 10

- 1.1 Motivation for the Research 10
- 1.2 Research Objective 12
- 1.3 Structure of the thesis 13

Chapter 2 Literature Review 15

- 2.1 Introduction 15
- 2.2 Exact approaches to Frequent Itemset Mining 15
- 2.3 Approximate approaches to Frequent Itemset Mining 18
- 2.4 Memory Management Techniques 20
- 2.5 Summary 24

Chapter 3 Research Methodology 25

- 3.1 Introduction 25
- 3.2 Research Approach 25
- 3.3 Research Opportunities and Hypotheses 26
- 3.4 Overall Architecture 26
- 3.5 Basic Building Blocks 29
  - 3.5.1 Data Structure 29
  - 3.5.2 Chernoff Bound 30
  - 3.5.3 1 Itemset Pruning 30
  - 3.5.4 Closed Itemset Mining 32
  - 3.5.5 Generation of Frequent Itemsets from Closed Frequent Itemsets 34
- 3.5.6 Overall Algorithm Structure 35
- 3.6 Summary 47

Chapter 4 Experimental Design 48

- 4.1 Introduction 48
- 4.2 Datasets 48
- 4.3 Performance Metrics 49
- 4.4 Support and Reliability 50
- 4.5 Experimental Plan 51

4.5.1 Experiment 1 51
4.5.2 Experiment 2 51
4.5.3 Experiment 3 53
4.6 Summary 54
Chapter 5 Research Findings 55
5.1 Introduction 55
5.2 Findings from Experiment 1 55
5.3 Findings from Experiment 2 61
5.4 Findings from Experiment 3 64
5.5 Summary 66
Chapter 6 Future Work 67
Chapter 7 Conclusion 69
References 70

Attestation of Authorship

“I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the qualification of any other degree of diploma of a university or other institution of higher learning, except where due acknowledgement is made in the acknowledgements.”

Yours sincerely,

(Omkar Kadam)

Acknowledgements

I sincerely thank Dr. Russel Pears and Dr. Yun Sing Koh for their endless support and encouragement without which this research work would not have been attainable.

I also thank Professor Krassie Petrova and Gordon Grimsey for their support and help.

List of Abbreviations

- CET: Cost Enumeration Tree,
- CFI: Closed Frequent Itemsets,
- DIUpdate: Direct Update tree,
- DSM: Data Stream Mining algorithm,
- FI: Frequent Itemsets,
- FPDM: Frequent Pattern Data Stream Mining algorithm,
- FP tree: Frequent Pattern tree,
- minSup: minimum support,
- Sup: support

List of Tables

Table 2-1: Node maintenance statistics 17

Table 3-1: Transactional dataset A 33

Table 3-2: Transactional dataset B 37

Table 3-3: Transactional dataset C 45

Table 4-1: Parameter values 49

Table 4-2: Performance metrics 49

Table 4-3: Key parameters used in experimentation 50

Table 5-1: Accuracy with fixed delta-Experiment 1-1 56

Table 5-2: Accuracy with fixed support-Experiment 1-2 56

Table 5-3: Accuracy with fixed delta-Experiment 1-3 59

Table 5-4: Accuracy with fixed support-Experiment 1-4 59

Table 5-5: Accuracy findings-Experiment 3 64

Table 5-6: Performance findings-Experiment 3 65

List of Figures

Figure 2-1: Relationship between frequent, closed frequent and maximal frequent itemsets 22
Figure 3-1: Overall Architecture 28
Figure 3-2: Item pruning 32
Figure 3-3: Itemset lattice 33
Figure 3-4: Node structure 36
Figure 3-5: Closed tree 37
Figure 3-6: Powerset generation 43
Figure 3-7: Hash table superimposed on tree 44
Figure 3-8: Closed tree construction 46
Figure 4-1: Recall and Precision formulae 50
Figure 5-1: Memory and time with fixed delta-Experiment 1-1 57
Figure 5-2: Memory and time with fixed support-Experiment 1-2 58
Figure 5-3: Memory and time with fixed delta-Experiment 1-3 60
Figure 5-4: Memory and time with fixed support-Experiment 1-4 60
Figure 5-5: Recall with fixed support-Experiment 2-1 61
Figure 5-6: Precision with fixed support-Experiment 2-2 62
Figure 5-7: Recall with fixed support-Experiment 2-3 63
Figure 5-8: Precision with fixed support-Experiment 2-4 63
Figure 5-9: Execution details-Experiment 3 66

Abstract

From the advent of association rule mining, it has become one of the most researched areas of data exploration schemes. In recent years, implementing association rule mining methods in extracting rules from a continuous flow of voluminous data, known as Data Stream has generated immense interest due to its emerging applications such as network-traffic analysis, sensor-network data analysis. For such typical kinds of application domains, the facility to process such enormous amount of stream data in a single pass is critical.

Nowadays, many organizations generate and utilize vast data streams (Huang, 2002). Employing data mining schemes on such massive data streams can unearth real-time trends and patterns which can be utilized for dynamic and timely decisions. Mining in such a high speed, enormous data streams significantly differs from traditional data mining in several ways. Firstly, the response time of the mining algorithm should be as small as possible due to the online nature of the data and limited resources dedicated to mining activities (Charikar, 2004). Second, the underlying data is highly volatile and subject to change over period of time (Chang, 2003). Moreover, since there is no time for preprocessing the data in order to remove noise, the streamed data can have noise inherent in it. Due to all aforementioned problems, data stream mining is receiving increasing attention and current research is now focused on the efficient resolution to the problem cited above.

Although, the field of data stream mining is being heavily investigated, there is still a lack of a holistic and generic approach for mining association rules from data streams. Thus, this research attempts to fill this gap by integrating ideas from previous work in data stream mining. This investigation focuses on the degree of effectiveness of using a probabilistic approach of sampling in the data stream together with an incremental approach to maintenance of frequent itemsets in a data stream environment. The following thesis describes the design and experimentation conducted with a novel association rule mining algorithm that can be deployed on a high speed data stream.

Chapter 1 Introduction

1.1 Motivation for the Research

Association rules, as its name suggests, expresses relationships between items in (generally large) datasets. This powerful data mining technique has a wide range of applications including Market-Basket analysis, text mining, web mining and pattern recognition between genes in a dataset. Such rules enable users to uncover hidden relationships and patterns in large datasets. For example, customer's buying patterns or relationships between different genes in organisms. The seminal work of (Agrawal, 1994) and the proposed Apriori algorithm spurred a plethora of research in the area of association rule mining.

In a wide range of emerging applications, data is in the form of an enormous, continuous stream where the speed at which the data is produced outstrips the rate at which it can be mined (Charikar, 2004). This is in direct contrast to traditional static databases; thus data stream mining therefore is substantially deviant from conventional data mining in numerous aspects. Firstly, the absolute volume of data embedded in a data stream over its lifespan can be overwhelmingly huge (Gaber, 2005). Secondly, due to resource bottlenecks, generating timely responses by keeping response time to queries on such data streams is necessary (Jiang, 2006).

Because of the issues stated above, data stream mining has become the subject of intense research and the problem of obtaining timely and accurate association rules is a contemporary research topic. There is a critical need to switch from traditional data mining schemes to those methods that are able to operate on an open-ended, high speed stream of data (Manku, 2002).

Due to the inherent nature of a data stream, any mining scheme faces the following challenges (Gaber, 2005). Firstly, due to the continuous nature of stream data, the traditional approach of scanning the database multiple times for model creation is no longer feasible. Moreover, the time required to rescan the database can severely degrade performance and response time. In the most fundamental sense, storage of the large amount of data produced will pose a major constraint on storage resources. Hence, an effective and compact memory structure along with an appropriate single pass algorithm becomes indispensable for data stream mining (Jiang, 2006).

Secondly, the algorithm should be able to adjust itself to the changing nature of the stream data which gives rise to the notion of *Concept Drift* (Wang, 2003). All data streams are susceptible to change in the content of data with the passage of time. Such changes in data content usually give rise to changes in patterns, thus giving rise to the phenomenon referred to as *Concept Drift*. Concept drift tends to further complicate the process of mining as patterns which were generated on a past segment of data may not hold in the future, thus giving need to a continuous monitory mechanism that checks and updates models created as and when necessary.

Thirdly, due to the fact that data in a stream may arrive at high speed, the data has to be processed quickly lest we lose important data. This constrains the mining speed to be faster than the incoming data rate (Lin, 2005), otherwise, the algorithm will have to employ some form of approximation technique such as load-shedding, sampling, or sketching (Gaber, 2005) which has the effect of degrading accuracy due to the probabilistic approach employed. Since storage of data in secondary storage is not an option, the success of any data stream mining algorithm depends on its ability to build a synopsis of the data in main memory and maintain such a synopsis in the face of continuous changes that take place in the underlying data stream. To this end, research into closed itemset mining becomes relevant as closed itemsets are a compact representation of the set of all itemsets without loss of information (Chi, 2004). Given a set of frequent closed itemsets it is possible to deduce the subset of frequent itemsets, along with their support values. Thus closed frequent itemset mining is one of the fundamental building blocks of this research.

The three factors above, taken together, dictate that an incremental approach to mining needs to be adopted in a data stream environment (Huang, 2002). The major factor affecting the efficacy of such a mining algorithm is: computing resources such as CPU time and memory; hence a resource-aware algorithm capable of adjusting itself as per resource availability is crucial for upholding the performance of the mining algorithm. Thus, it is clear that in the field of data stream mining, a novel approach is required that for exploits the generation of frequent closed itemsets along with sampling and memory optimization techniques is crucial.

1.2 Research Objective

Many application domains require investigating the associations inherently present in the databases, such domains include text mining, pattern analysis, web data analysis, bio informatics (Pang, 2006). Association rule mining, as the name suggests, an activity of discovering hidden relationships and associations in the large databases, therefore becomes a highly valuable and vital data mining technique in such domains. Armed with information and relationships provided by association rule mining techniques , the data miner can have the ability to determine buying patterns of customers in market-basked analysis application or to unearth the genetic correlation in a bio-informatics study or to discover credit card defaulter behavioral patterns in banking domain.

Association rule mining basically can be seen as a two step activity: first, determining the frequent itemsets that fulfill the minimum support threshold, second, deriving rules from the frequent itemsets that were discovered in first step. Since, the efficiency and effectiveness of the generated rules is highly affected by the frequent itemset mining phase (Chi, 2004), the frequent itemset mining phase has received much significant attention so far. The works such as (Chi, 2004), (Charikar, 2004), (Yu, 2004), (Jiang, 2006) are few examples where the focus of the research has been mainly on frequent itemset mining rather than rule generation phase. Another reason for researchers to focus on frequent itemset mining is because the rule generation can be executed exponentially quicker than first phase (Bodon, 2003). Therefore, this research also is focused on the problem of the frequent itemset mining instead of the rule generation.

The seminal work of (Aggarwal, 1993) first addresses the problem definition of frequent itemset mining, which can be modified to adjust finding frequent itemsets over continuous data stream and be expressed as following, Let ‘I’= {i_1, i_2, i_3... i_m} be set of items. A data stream of transactions ‘T’ is a sequence of incoming transactions where each transaction contains itemset ‘X’ which is a subset of items ‘I’. Support of ‘X’, denoted as ‘sup(X)’ is the number of transactions that contained X. The itemset X is a frequent itemset is $\text{sup}(X) \geq$ ‘s’, where ‘s’ is minimum support threshold. The algorithm Data-Stream-Mining (DSM) is to find an approximate collection of frequent itemsets (relative to the minimum support threshold provided by the miner). The approximation is controlled (similar to (Yu, 2004)) by parameters epsilon ‘ε’ and delta ‘δ’ to regulate the degree of error and reliability respectively.

1.3 Structure of the thesis

This chapter has discussed the challenges associated with rule generation in a data stream environment. Technical issues such as the need for efficient memory management and the need for an incremental approach for model building were shown to be indispensable for mining rules in a data stream environment. Furthermore, the problem of concept drift was identified and it was noted that a monitoring mechanism that continuously tracks changes in the data stream needs to be implemented.

In the next chapter we will survey past work done in the area of association rule mining, with particular emphasis to algorithms proposed for mining in a data stream environment. We will also note and briefly comment on research into closed frequent itemset mining as it forms one of the cornerstones of our research.

In Chapter 3 we will present our methodology which will include an overall architecture and the design of the algorithm that we utilized for experimentation. The three different strands of our research, namely the partitioning of the data stream into blocks, the use of closed itemset mining and the incremental approach will be fully described.

A plan for the empirical study will be presented in Chapter 4. We will detail the various experiments that we ran and compare results with previous work, where appropriate. Our experimentation will track performance measures such run time, memory consumption and rule accuracy which is measured in terms of Precision and Recall.

Chapter 5 presents the experimental results that show that our proposed algorithm outperformed the current state of the art data stream association rule miner on all metrics that we tracked.

Chapter 6 discusses several different directions in which future work can be undertaken to further improve the performance of our proposed. Chapter 7 concludes the thesis with a discussion of the key achievements of the research.

Chapter 2 Literature Review

2.1 Introduction

The problem of mining association rules from a data stream has been addressed by many authors but there are several issues (as highlighted in previous sections) that remain to be addressed. In the following section we will discuss existing literature based on the problems in data stream mining that they address.

The research in this domain can be effectively classified into three different domains namely, *Exact methods for Frequent Itemset Mining*, *Approximate Methods* and *Memory Management* techniques adopted for data stream mining.

2.2 Exact approaches to Frequent Itemset Mining

There are two main approaches to mining association rules in a data stream and these can be categorized into either Exact or Probabilistic approaches (Gaber, 2005). The Exact approaches offer the highest level of accuracy but compromises on execution speed, particularly in the case of voluminous, high speed streams. The algorithms that adopt exact approaches include (Yang, 2004), (Chi, 2004).

(Yang, 2004) proposed a method for determining short frequent itemsets in one scan of data and then generating association rules from the discovered frequent itemsets. It used a basic primitive data structure, an array, to store the support information of the frequent itemsets for a predefined length of the itemset. The elements in the array are arranged in lexicographical order just as in a dictionary. The array at any moment of time contains only those itemsets who has length less than ‘k’ where ‘k’ is predefined maximum length of frequent itemsets in the database. Since the array is set up in lexicographical order, adding a new itemset to the array becomes very efficient and quick with the help of a merge sort operation.

Whenever a transaction is received, for all the itemsets that fulfill the itemset length condition (less than k), a method 'rank' is invoked to return the index of the itemset in the array structure. Once the index position is obtained, the itemset at that location gets its support count incremented. If the itemset does not exist then an entry for the new itemset is created with the help of the merge sort operation. Whenever the user requests mined results, all the itemsets fulfilling minimum support threshold are determined by traversing the array and the association rules are generated and presented to the user.

The approach proposed by (Yang, 2004) suffered from a major limitation in that it was capable of only handling short ($k \leq 3$, where 'k' is length of itemset) itemsets and it failed in the case of large itemsets. It is based on the assumption that most applications would be interested in only short frequent itemsets. This assumption is questionable in a data stream environment.

Chi (2004) proposed a new algorithm, Moment, to mine and store all closed-frequent-itemsets using a sliding window which holds the most recent samples in a data-stream. The recent samples are stored using memory-structure called the Closed-Enumeration-Trees (CET) The CET is constructed using a depth-first process which maintains every itemset in lexicographical order and if the itemset is found frequent then it is added to the tree as a node. Non-frequent itemsets are also stored as they may become frequent in the future. This approach distinguishes between 4 types of nodes, namely, Infrequent gateway node (infrequent nodes whose parents and/ siblings are frequent), Unpromising gateway node (infrequent node who has a superset having same support as itself), Intermediate node (frequent node who has a superset having same support as itself), and Closed node (node which has no children with greater or equal support as itself).

During the sliding window timeframe, whenever a transaction arrives, Moment classifies the nodes present in the transaction into one of the 4 categories mentioned previously. Unlike prefix trees, Moment's CET maintains only closed itemsets and nodes that mark the boundary between closed itemsets and the rest of the itemsets. A hash table keeps track of all the nodes seen so far which is updated to maintain support information of all the nodes seen so far. After the arrival of a new transaction the oldest transaction is effectively deleted from the time window by traversing through the nodes that were affected by this transaction. At the same time the new transaction is added by noting the node type for each itemset contained in the transaction and updating the relevant nodes in the CET as needed. Whenever the user requests for mined results the CET (which observes only closed nodes and boundary nodes) is traversed to mine the top ‘n’ closed-frequent itemsets.

(Chi, 2004) assumed that all the interesting changes occur at the boundary of closed-frequent itemsets and rest of the itemsets which turns out to be true most of the time. Nevertheless, the Moment algorithm suffers greatly due to the CET data structure (cost-enumeration-trees) whose operations such as tree creation and maintenance is very time-consuming (since it scans the constructed tree in a depth-first manner). Although the algorithm suggested by (Chi, 2004) outperformed the then state-of-art approach which was CHARM (M. Zaki, Gouda, K., 2001), it maintained a large number of nodes in the tree structure which hindered its performance significantly.

$minsup$ (in %)	closed itemset #	CET node #	CET node # per closed	changed node #	new node #
1.0	4097	148450	36.2	0.14	6.28
0.9	5341	168834	31.6	0.06	0.92
0.8	6581	187076	28.4	0.13	0.64
0.7	8220	212774	25.9	0.09	0.35
0.6	10270	249549	24.3	0.08	1.30
0.5	12655	309575	24.5	0.10	2.58
0.4	16683	433595	26.0	0.18	4.20
0.3	24907	722645	29.0	0.26	9.52
0.2	45353	1614726	35.6	0.67	27.23
0.1	172396	5955425	34.5	3.41	88.69
0.05	722261	19691999	27.3	14.75	286.34
0.03	1704558	45246906	26.5	41.07	646.84

Table 2-1: Node maintenance Statistics (Chi, 2004)

It is evident from Table 2.1 that Moment spends significant amount of resources on creating and maintaining a huge number of unnecessary nodes. Column 4 shows that the total number of nodes it maintains is at least 26.5 times the number of closed frequent nodes. Given that all information on rule generation can be obtained through the use of closed frequent itemsets only, it is evident that the storage and computational overheads consumed by Moment are very high.

Thus, it can be surmised that, although algorithms based on exact methods guarantee better accuracy they suffer greatly in terms of computation time and memory requirements. Therefore, several approximation based methods were initiated to provide satisfactory performance by keeping computation costs under control. Such approximation based approaches include (Yu, 2004), (Wang, 2003), (Manku, 2002) and (Charikar, 2004).

2.3 Approximate approaches to Frequent Itemset Mining

We now describe approximate methods that prioritize on performance, at the possible expense of accuracy. Some of them offer worst-case error guarantees to provide a level of confidence in the results generated (for example, (Charikar, 2004)). The algorithms that do not provide such error guarantees include (Yu, 2004). Since these were approximated results, the algorithms following these methods suffered from either *False-Positives* (certain itemsets which are not frequent are wrongly identified as frequent itemsets) or *False-Negatives* (certain frequent itemsets are not identified by the algorithm as frequent itemsets) (Yu, 2004)(Jiang, 2006). Many of these algorithms employed approximation techniques such as sampling, load-shedding, clustering and sliding window mechanisms (Gaber, 2005).

Yu (2004) proposed algorithms that can mine frequent items as well as frequent itemsets from data-streams with the use of a memory-consumption constraint. The suggested algorithm, Frequent-Data-stream-Pattern-Matching, (FDPM) produces a set of frequent itemsets that has Recall values that are reported to be in the high 90 percentile mark, meaning that the results it produces are a very good approximation to that produced by exact methods such as Moment. The approximation is governed by two parameters that are referred to as the error-control and reliability-control parameters that set the level of error and the degree of statistical significance attributable to the result, respectively. The algorithm is false-negative oriented meaning that it may not pick every frequent itemset but it guarantees that every itemset picked is frequent.

The algorithm uses the Chernoff bound (Ravikumar, 2004) to conceptually partition the data stream into frames, each of which represents a chunk of data on which inferences about the frequency status of itemsets can be made with a pre- specified probabilistic guarantee. FPDM2 uses two data structures, namely, ‘P’ and ‘F’. The structure P maintains all the frequent itemsets in the current frame whereas F maintains frequent itemsets seen so far. Whenever a transaction arrives FPDM2 updates the support information of itemsets that exist in P. At the end of a frame it determines the frequent itemsets in P that must be transferred to F. Once the current frequent itemsets are transferred to F, the data structure P is truncated in order to make room for frequent itemsets from the next data frame. Whenever the user requests the mined results, the algorithm outputs elements from F that fulfill the minimum support threshold. As mentioned before the approach is false negative in nature and was compared against (and outperformed) false positive approaches such as Lossy-counting and Sticky sampling (Manku, 2002). The Chernoff bound enabled the FPDM2 algorithm to prune infrequent itemsets from the data structure F safely without loss of information as statistical guarantees could be obtained regarding the frequency status of such itemsets. Thus, the memory resources consumed by FDPM could be kept within very tight bounds unlike exact methods such as Moment.

Charikar also employed an approximate approach (Charikar, 2004). With the help of a compact data-structure, their Count-sketch algorithm mined the data stream in a single pass.

The thrust of (Charikar, 2004) lies in the creation and maintenance of its data structure, Count Sketch. This data structure consists of a collection of hash functions along with an array of counters that simulate an array of hash tables (referred to as Heaps). Count sketch has two crucial methods: to append the newly found itemset, and to estimate its support count (which is done with the help of median values obtained from hash functions). Once the data structure is constructed, whenever a new transaction comes, the algorithm computes the estimated support count for itemsets present in the transaction and adds them into the data structure if the estimate for these itemsets is greater than the estimated value for that heap. If the itemset already exists then its count information is updated. Whenever the user requests for mined results, it reports the top ‘k’ elements as frequent itemsets.

This approach mainly focused on obtaining the top ‘n’ frequent itemsets with the help of an efficient data-structure. The algorithm is simple in design and is compared with a traditional sampling algorithm. The Count-sketch algorithm outperformed sampling in the context of storage-space since Count-sketch keeps only the top ‘n’ objects from stream whereas sampling keeps substantially a large number of objects. The proposed algorithm can be extended to a 2-pass algorithm for a scenario of estimating items with large frequency-change between two inter-connected data-streams; this issue was not addressed so far by any other work. The algorithm is based on hash functions which are inherently fast, thus yielding increased speed for the Count-sketch algorithm. The data structure count-sketch stores only top ‘k’ elements at any moment of time. In spite of this, the authors did not address the cost of actually storing the elements from the data-stream since different encoding techniques (such as Zipfian method) will yield different space usage.

It can be deduced that both exact and probabilistic approaches each have their own benefits and drawbacks. Therefore, selection of an association rule mining approach should be done as per the domain of investigation since some of the application domains may require a higher degree of accuracy (such as the Medical or Financial sectors) and certain domains like Marketing require faster rather than the most accurate results.

2.4 Memory Management Techniques

With respect to memory management, researchers have emphasized on the use of compact data structures for incrementally maintaining itemsets in contrast to traditional static database approaches (Jiang, 2006). This is primarily because the traditional approaches are not applicable for data stream mining for several reasons. First is the problem of insufficient memory. The stream data is vast in volume and storing such voluminous data is impractical. Second, the support information of the transactions is susceptible to frequent updates and therefore, scanning and updating such a huge volume of data is a very costly process. Therefore, it is essential to keep minimal, yet sufficient enough for mining the association rules from stored data. As an answer to this problem, research by (Yang, 2004), (Charikar, 2004), (Chang, 2003) keeps only frequent itemsets in main memory (Jiang, 2006). Thus, their research concentrates on the use of compact and efficient memory structures to hold information pertaining to only frequent itemsets.

(Charikar, 2004) used the Count-sketch data structure that keeps the estimated count support of high frequency itemsets. The main problem with this approach is that it supports the generation of only top 'N' itemsets (as ranked by frequency of occurrence in the data) and does not consider the notion of concept drift. Moreover, it suffers from the accuracy-space tradeoff as with many other approximation based approaches. Yu (2004) showed that small increases in accuracy were accompanied by very exponential increases in memory requirements.

Another similar approach is that of (Huang, 2002) which followed the use of the FP (frequent-pattern) tree structure, a compact data structure (Lin, 2005) for intermediate storage of frequent itemsets. This method is modified and followed by many other research works such as (Lin, 2005) and (Chi, 2004). Yet, creation, maintenance, and traversal remain primary concern and potential bottleneck for FP tree (Huang, 2002).

In general, the literature has employed the tree structure for maintaining frequent itemsets and this research will be following the same approach of using tree structure for intermediate storage of frequent itemsets.

As mentioned so far, mining association rules from stream data demands significant amount of computation resources such as CPU time and memory. This is mainly due to the fact that the algorithm has only one scan of the data in data stream mining. Thus, the decision as to when to update the model (whether at mining time or as and when updates come) is crucial and influences the performance of the underlying mining algorithm (Jiang, 2006).

The most commonly followed approach is the use of an incremental approach for maintaining and updating association rules. The first such effort was that of (Cheung, 1996). Although, this approach was not fully applicable for stream data, it provided the necessary framework for all other approaches that follow an incremental approach for updating and maintaining association rules. Zheng in (Zheng, 2003) employed a metric for determining the difference between sequential patterns. This metric was used to indicate when to update the support information of the sequential patterns. Nevertheless, this approach was suitable only where the problem of concept drift was minimal. Certain other approaches try to determine a trigger event which can be used as a cue to update all the transactions. Such approaches include (Ben-David, 2004) which used statistical means such as chi-square methods to find out the difference between past frame data and current frame data. Depending on the variation in the data the decision of updating the rule base or to defer the update to a later stage was taken. Nevertheless, it assumed that the underlying data distribution would remain the same for the lifespan of the data stream which is seldom the case in most real-world data streams.

Besides an efficient memory construct to hold valuable information, memory management also involves the use of the closed frequent itemsets. In practice, transactional data leads to the production of an exponentially vast number of frequent patterns, therefore it becomes incumbent to discover a small fraction of patterns that are representative and from which other frequent itemsets can be derived (Pang, 2006). There are two such representations, namely, Maximal frequent itemsets, and, Closed frequent itemsets (Celgar, 2006). The proportion of maximal and closed frequent itemsets against frequent itemsets is as shown in following figure.

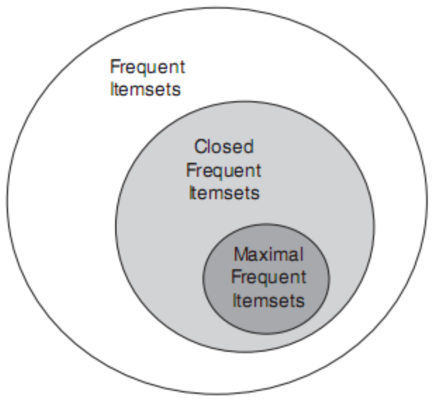


Fig. 2-1 Relationships between Frequent, Closed frequent and Maximal frequent itemsets (Pang, 2006)

A maximal frequent itemset can be defined as a frequent itemset for which none of its direct children (supersets) are frequent itemsets, whereas, a closed frequent itemset ‘X’ can be defined as a frequent itemset for which none of its direct successors (supersets) have same support as that of ‘X’. It is clear from the definition and the figure above that the number of maximal itemsets in a dataset is significantly lesser than that of closed frequent and frequent itemsets. In spite of providing a concise representation, Maximal frequent itemsets do not provide lossless representation since they do not contain support details of their ancestors (subsets). Closed frequent itemsets, on the other hand do provide a compact and lossless representation and therefore much research (Jiang, 2006), (Wen, 2004), (Chi, 2004), (Zaki, 2002) has focused on exploiting the power of closed frequent itemsets in association rule mining.

Jiang (2006) exploited closed frequent itemsets for association mining in a data stream environment. The proposed algorithm, called CFI-stream computes closed itemsets incrementally, on the fly without any support information (with the help of closure function). It uses the closure function to determine whether an itemset is closed or not. The data structure referred to as DI Update (Direct Update) tree stores only closed itemsets in a lexicographical order.

CFI-stream is based on sliding window framework and therefore has to account for new transactions arriving in the window as well as decaying the effect of transactions that were dropped off. Whenever a transaction is dropped off as result of its departure from the sliding window, a routine traverses through every itemset in that transaction and then updates the related closed itemsets to reflect deletion of the transaction. Addition of a new transaction is also dealt with similarly by updating only associated closed itemsets. This checking of closed nodes to see if they are associated with itemsets from current transaction saves computational time in case of introduction of a new transaction, but it can severely degrade performance while diminishing the effect of past transactions since it has to traverse through the tree for all the nodes and update them.

Although CFI-stream outperformed Moment (Chi, 2004) in most of the experiments, it is highly efficient only for the datasets where there exists a higher degree of correlation between the transactions. Moreover, the housekeeping functions to simulate sliding window framework, such as traversing the DI Update tree and updating all the associated closed nodes for introduction of new transactions and deletion of old transactions can very well be resource consuming.

It is observed that closed frequent itemset mining has been significantly researched for last few years and it continues to be so because of advent of different application domains such as high speed data streams.

2.5 Summary

In this chapter, we have provided a brief outline of association rule mining algorithms in data stream environment with the help of basic introduction, variety of approaches, their pros and cons. We have also underlined the current major issues in mining association rules in data streams. In the following chapters we will carry out a detailed analysis of the research methodologies, research issues and design a novel algorithm followed by an analysis of experimental results in Chapter 5.

Chapter 3 Research Methodology

3.1 Introduction

This chapter presents the research methodology and paradigm that fundamentally governs the methodology selected to fulfill the planned objectives of this research. It offers a brief overview of research approach, the methodology chosen to carry out this research and the research methods that were employed.

The field of data mining is essentially concerned with pragmatic and efficient solutions. Data mining is mainly employed in order to optimize the value of existing data and extract useful and valuable information from large datasets. Therefore, such investigations are more closely aligned with a Positivist research paradigm or theoretical perspective since we assume that the application of data mining will result in the generation of useful knowledge (Dash, 2005).

The French philosopher Comte suggested that a positivist approach utilizes observation and experiment to understand the domain under consideration (Dash, 2005) which is the case in data mining as well. Hence, this research will be heavily reliant on the positivist (Collis, 2003) research paradigm.

3.2 Research Approach

As opposed to the Positivist research approach, Interpretivist research tends to be subjective in nature where interpretation of an individual (or researcher) about a phenomenon or event is important (Burrel, 1979). However, the key features of the Positivist research approach such as the notion of objectivity, well-planned experiments and research, the seeking of exact measurements and evaluations, etc. tends to align with this research work more closely than the Interpretivist approach. Therefore, this research will utilize the Positivist approach.

3.3 Research Opportunities and Hypotheses

The Introduction section underlined some of the crucial issues that need to be addressed in data stream mining. Despite the fact that there has been substantial research carried out in the area of data stream mining , it still lacks a holistic and generic approach to mine association rules from data streams. Thus, this research will attempt to fill this gap by integrating ideas from previous work in data stream mining. This investigation will focus on the degree of effectiveness of using a probabilistic approach of sampling in the data stream together with an incremental approach to maintaining frequent itemsets in a data stream environment.

3.4 Overall Architecture

This section highlights the interface of main building blocks in this algorithm (the building blocks are explained at length in next section). The basic building blocks this algorithm is based on are as follows,

- Tree Data Structure,
- Chernoff Bounds,
- 1 Itemset pruning,
- Closed Itemsets,
- Generation of Frequent Itemsets (FIs) from Closed Frequent Itemsets (CFIs)

The aforementioned blocks mainly make the foundation of the proposed algorithm, Data Stream Mining. Our approach is based on closed frequent itemset mining, and therefore there has been a strong necessity to have a tree data structure which is capable of holding parent-child relationship; essential in closed itemset determination.

Chernoff bounds is a statistical guarantee which allows determining a number of samples (in our case, transactions) that would allow us to infer with acceptable confidence. Chernoff bounds allows us to establish the number of transactions that make up the individual frames (Yu, 2004).

The powerset generation, an imperative task in order to determine all possible supersets of singleton items turns out to be a resource intensive process since the average number of singleton items in a transaction from data stream can very well be in few hundreds, moreover, many of such generated powersets yield no rewards in closed itemset mining, therefore, we have incorporated a method of 1 itemset pruning which strips the incoming transaction from any infrequent items found in previous passes of processing, allowing us to reduce the number of singleton items that generate interesting supersets without losing any information. In this way, we use only the previously found frequent items to generate powerset since according to Apriori principle; no infrequent subset can lead to frequent superset (Aggarwal, 1994).

Closed itemset mining is a very promising alternative to frequent itemset mining mainly because closed frequent itemsets provide a concise and lossless representation (Pang, 2006) allowing us not only to infer the support information of its supersets but also generation of all the frequent itemsets. It means that just by maintaining closed frequent itemsets we can determine all the frequent itemsets along with their support information. Since the closed frequent itemsets are subset of frequent itemsets (Pang, 2006), maintaining a smaller number of items offers us computational gains.

Our algorithm is mainly closed frequent itemset mining, but in order to generate interesting patterns, we need the frequent itemsets. (Pang, 2006) presented a method to identify all frequent itemsets along with their support information from closed frequent itemsets, we shall be implementing this method in our algorithm so as to list out frequent itemsets as mined results.

Following diagram highlights the overall architecture of our algorithm underlining how the building blocks mentioned previously fit together in our algorithm,

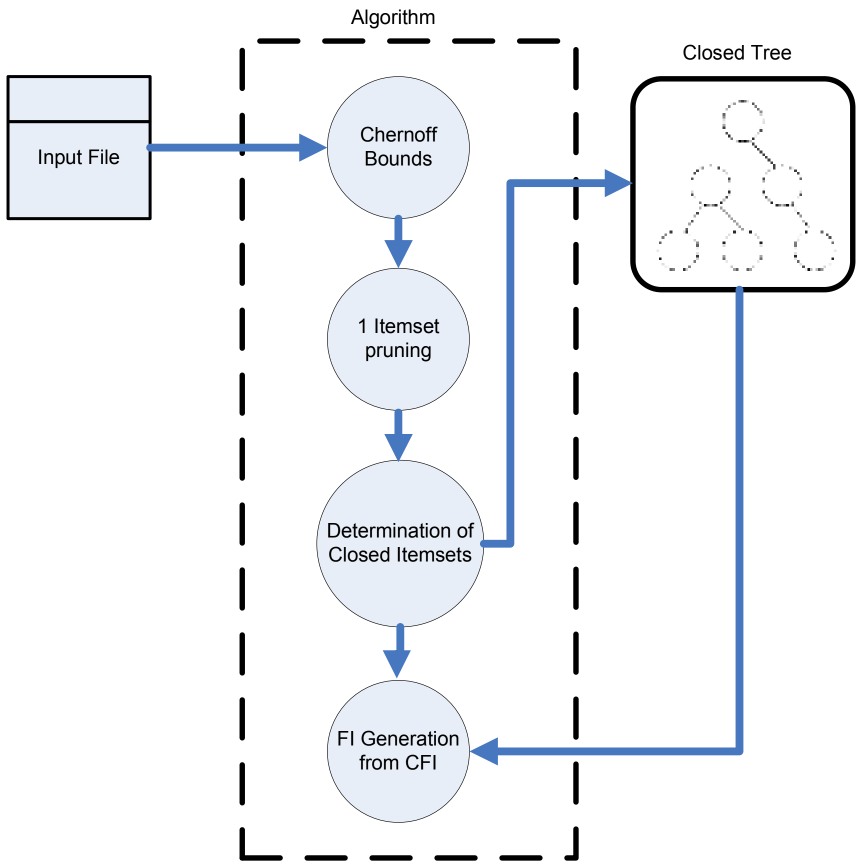


Fig. 3-1 Overall Architecture

As shown in figure 3-1, the algorithm reads the input transaction file (in a data stream environment, it shall be continuous stream of transactions), the Chernoff bounds breaks the input transaction stream into frames, depending on previously identified frequent singletons, 1 itemset pruning removes infrequent singletons from the transaction. Once, the transaction is striped off from the infrequent items, a block of code generates powerset for every pruned transaction. Once the powerset is generated, with the definition of closed itemset, the algorithm updates the closed tree that holds the intermediate information about all the closed itemsets. At the end of every frame, we store the closed itemsets found in that frame into the closed tree, whenever the user requests for the mined results, the algorithm traverses through the closed tree and determines the closed frequent itemsets, once all the closed frequent itemsets are identified, the routine to generate FIs from CFIs is called, giving us all the frequent itemsets along with their support information.

3.5 Basic Building Blocks

This section presents the major building blocks that constitute the proposed for mining association rules in a Data Stream environment.

3.5.1 Data Structure

In the domain of high speed data streams where the volume of incoming data is enormous, an effective and concise data structure is required to store, update and retrieve essential information. This is mainly because of memory constraints and the huge amount of incoming data in case of data streams (Jiang, 2006). In the absence of such a compact data structure buffering of data on secondary storage may be required, leading to significant increases in processing time due to I/O operations.

An efficient data structure is therefore a vital part of an effective data stream mining we thus seek a compact memory structure capable of supporting efficient insertion, deletion, modification and retrieval.

The existing literature has favored several variations of a tree data structure such as the Frequent Pattern (FP) Tree (Huang, 2002) and the Closed Enumeration Tree CET (Chi, 2004) for the following reasons,

- A tree data structure aids in fast counting of itemsets (Huang, 2002).
- With trees, we can employ inter and intra-node pruning techniques which will help us in reducing search space (Moonesinghe, 2006).
- It facilitates easy insertion, deletion, retrieval and updation of information (Huang, 2002).

Because of aforementioned reasons, the use of a tree data structure to store itemset information is common and therefore this research will also make use of the tree structure for retaining support information about the discovered frequent closed itemsets.

3.5.2 Chernoff Bound

The Chernoff bound which is a special case of the Chernoff inequality and gives a lower bound value for 'n' independent, yet equally likely incidents (Ravikumar, 2004). Algorithms such as CHARM (M. Zaki, Li, W., Ogihara, M., 1997) and Moment (Yu, 2004) used the Chernoff bound for determining a sample size because of Chernoff bound's statistical guarantees on the degree of error in the estimation of itemset support.

Considering 'X' as a real valued variable with range R, the Chernoff bound states that (Ravikumar, 2004), with a probability of (1-δ) the actual mean of the variable ($\ddot{X}$) under consideration is within 'Є' of the mean irrespective of the underlying stochastic distribution of the variable X, where Є is given by:

$$\epsilon = \sqrt{\frac{R^2 \ln(2/\delta)}{2n}}$$

Equation 3.1

In other words, the above statement means that if we can determine $\ddot{X}$ to be within Є of its actual value with a probability of 'δ', we need to accumulate only 'n' samples of variable 'X' to make inferences. The value n is given by:

$$n = \tfrac{1}{2} (R/\epsilon)^2 \ln(2/\delta)$$

Equation 3.2

If we associate $\ddot{X}$ with the running support value of an itemset then the Chernoff bound enables us to determine the size of that block of data from which statistically sound inferences can be made about the frequency status of itemsets.

3.5.3 1 Itemset Pruning

The necessary step for computing itemsets from the stream of items in a transaction is the computation of a powerset that would take into account all the possible combinations of itemsets of size 1 (that is, individual items appearing in the transaction without combination with other items) seen in a transaction. The problem such an approach is that in a data stream where the incoming transaction can very well hold hundreds of items, the CPU time would be prohibitive as the time complexity for the powerset computation is O(2^n) where n is the items in the transaction.

As a solution to this problem, our approach undertakes 1 itemset pruning. In this approach, the data stream is segmented into a number of equal-sized blocks which we call frames. The size of a frame is determined by the Chernoff and is given by,

$$n_0 = \frac{2 + 2\ln(2/\delta)}{s}$$

Equation 3.3

(For derivation of above formula from generic Chernoff bounds formula, please refer (Yu, 2004)). As per above formula, when, the minimum support threshold 's'=0.001 and delta 'δ'= 0.1, the memory bound n_0 = 7991, that is every frame is marked by 7991 transactions.

We use the first frame of data to determine a list of 1-frequent itemsets (singletons). We then make use of the Apriori property (Agrawal, 1994) which states that a subset of a frequent itemset must also be a frequent itemset which implies that no superset that is infrequent can ever yield a frequent subset. It is therefore safe and appropriate to focus resources on computing the powerset from only the frequent singletons. Once the first frame is processed, we are in a position to prune infrequent singletons and compute the powerset only from the singletons who survive the pruning step. In order to avoid missing frequent singletons in subsequent frames, we will monitor singletons in every frame and compile a list of singletons that were frequent up to the last frame that was fully processed. At the end of every frame, the algorithm will identify the set of frequent singletons found in the current frame and compare it with the cumulative list found across all previous frames. The union of these sets will then be used to prune the transactions in the current frame. The procedure is outlined in Fig 3-2 below.

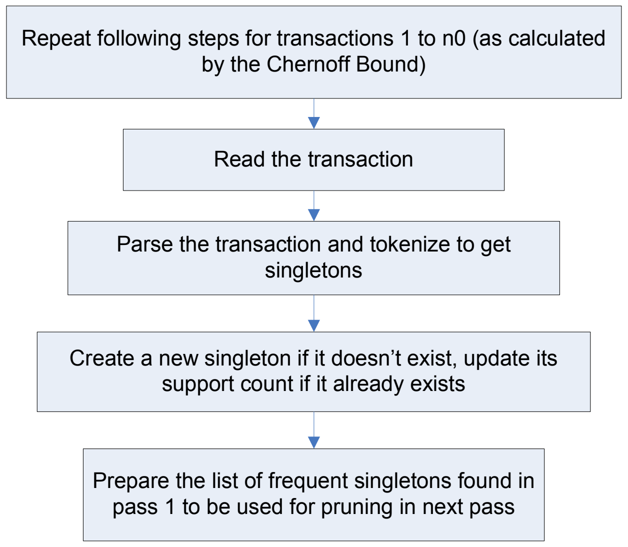

Repeat following steps for transactions 1 to n0 (as calculated by the Chernoff Bound)
Read the transaction
Parse the transaction and tokenize to get singletons
Create a new singleton if it doesn't exist, update its support count if it already exists
Prepare the list of frequent singletons found in pass 1 to be used for pruning in next pass

Fig. 3-2 Item Pruning

3.5.4 Closed Itemset Mining

Frequent itemset mining has a crucial role to play in association rule mining (Agrawal, 1994). Nevertheless, the task of mining association rules can result in a huge number of candidate itemsets that need to be assessed against their frequency status, thus impacting on efficiency. The classical Apriori (Agrawal, 1994) approach generates candidate k-itemsets for every pair of (k-1) frequent itemsets. When the minimum support threshold is et a low enough value, this can result in a combinational explosion the number of candidates to be processed (Chi, 2004).

Closed Frequent itemset mining presents itself as an attractive alternative to classical frequent itemset mining, particularly in a data stream environment. A closed itemset I is any itemset that does not contain a superset with the same frequency as itself. A closed frequent itemset is one that has support above the minimum support threshold. Extensive research has shown that the set of closed frequent itemsets is a small subset of the set of all frequent itemsets (Chi, 2004), (Gaber, 2005), (Pang, 2006), (Jiang, 2006). At the same time, closed frequent itemsets can be used to both derive the identities and the support values of all frequent itemsets (Pang, 2006). Pang et al describe an algorithm by which the support of all frequent itemsets can be extracted efficiently from the set of closed frequent itemsets.

We now illustrate the compact nature of closed itemsets with a simple example. Consider the following example of a frequent itemset lattice corresponding to a set of 4 transactions.

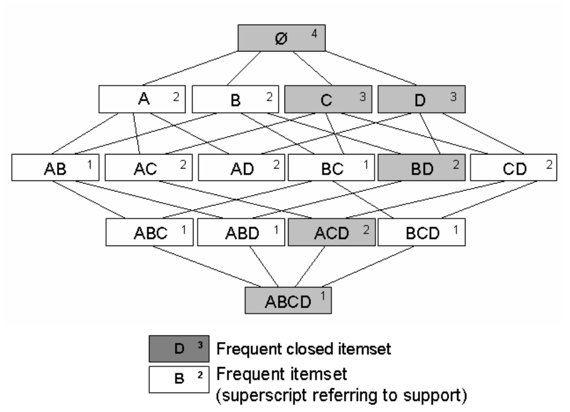


Fig. 3-3 Itemset Lattice

Figure 3-3 highlights a frequent itemset lattice with a minimum support threshold of 1. The transactional dataset that produces this lattice is shown below in Table 3-1.

Transaction ID	Items			
1	C			
2	B	D		
3	A	C	D	
4	A	B	C	D

Table 3-1 Transactional dataset A

As it is clear from this simple example, the number of closed frequent itemsets is significantly lower than that of frequent itemsets Given the compact nature and the lossless representation of closed itemsets it is a logical choice for mining data streams which require models to be stored entirely in memory and it for these reasons that we adopted this approach in our research. Closed itemsets offer the possibility of mining at low support thresholds that tend to overwhelm the memory due to the often exponential increase in the number of candidate itemsets that need to be tracked with classical frequent itemset mining. Closed itemset mining, on the other hand effectively enables the miner to operate at lower support levels than would be possible with traditional methods.

3.5.5 Generation of Frequent Itemsets from Closed Frequent Itemsets

As explained before, maintaining the closed itemset serves multiple purposes, viz. compact representation and reduced overhead. Another such advantage of concentrating on closed frequent itemsets is that the Closed Frequent Itemsets (CFIs) provide loss-less information, meaning that by just keeping track of CFIs, we can find out all other frequent itemsets as well. Once the closed frequent itemsets (CFIs) are identified, these CFIs can be used to determine the support of all the non closed frequent itemsets (Pang, 2006). The basic principle needed to find the support of non-closed frequent itemsets is to exploit the definition of a closed frequent itemset, that is, if a node is not closed, its support must be the same as one of its supersets. According to the Apriori principle, any transaction that has a superset of item X must also contain item X itself, but any transaction that contains X does not necessarily include the superset of X; therefore, the support of a non-closed frequent itemset must be equal to the maximum support among its supersets (Pang, 2006). The following routine infers the support of non-closed frequent itemsets from the set of closed frequent itemsets,

<u>Routine: calculateSupport ()</u>

<u>Parameters:</u>

- Let ‘C’ be a list of closed frequent itemsets,
- Let ‘F’ be a list of all non-closed frequent itemsets,
- Let ‘maxLength’ be length of the largest frequent itemset

<u>Method:</u>

```
1. Find the maxLength in order to facilitate bottom-up traversal (that is
   largest itemset first)
2. For k= maxLength to 1
      a. Find list of all the frequent itemsets ‘$F_k$’ of size ‘k’,
      b. For all ‘f’ Є $F_k$, repeat following,
            i. If ‘C’ does not contain ‘f’, then support of ‘f’ is maximum of
               its descendants’ supports
      c. End for
3. End for
```

Description:

The routine traverses the tree in a bottom up fashion. For each node scanned its closure status is first determined; if the node is not closed then, as mentioned previously, its support must be the maximum of its supersets' support. On the other hand if the node is closed then no further action is taken other than to scan the next node in the bottom up traversal order.

In this way, the compact and lossless closed itemset representation allows us to find all possible frequent itemsets without actually storing them.

3.5.6 Overall Algorithm Structure

As discussed in previous sections, the most suitable data structure for frequent closed itemset mining is a tree structure. Accordingly this research uses a tree structure for storing itemset information that would facilitate the mining phase later on.

Each node in the tree represents an itemset 'i'. Unlike prefix trees (Chang, 2003) and FP trees (Giannella, 2003), this tree structure maintains only closed itemsets which includes both frequent and infrequent itemsets. However, the algorithm will undertake periodic pruning to ensure that tree maintains closed itemsets that are either currently frequent or have the potential to become frequent over a period of time into the future. The pruning routine that we use is as follows,

Routine: pruneTree ()

Parameters:

- Let 'H' be a list of closed itemsets in the form of a Hash table,
- Let 'minSup' be a minimum support threshold,

Method:

4. For all the elements in Hash table, do the following
    a. Retrieve the node 'N' from the tree with the help of reference from Hash table,

```
        b. If Support(N) < minSup, then
            i. Obtain the list 'L' containing all the children of N,
            ii. Remove the nodes present in 'L' from tree and hash table
        c. End If
    5. End for
```

<u>Description:</u>

The pruneTree routine, traverses through Hash table that keeps track of all the frequent nodes created so far, and investigates infrequent closed nodes present in the hash table. Once, we have obtained the infrequent nodes, we retrieve their reference from hash table in order to find corresponding nodes in the tree (which has parent-child information of the nodes). After we get a list of infrequent nodes, we traverse through their children and remove them from the tree as well as the hash table. It makes more sense to remove the children of infrequent nodes and purge that branch because the children of infrequent nodes can only become frequent if one or more of their parents become frequent, therefore we give chance to infrequent nodes to become frequent in future but purge their children.

The node structure is as follows and shown in Figure 3-4:

- Itemset identification, 'i',
- Support 's',
- Links to its immediate ancestors and descendants.

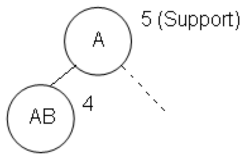


Fig. 3-4 Node Structure

The literature has adopted several variations of tree structures such as the FP tree (Huang, 2002), CET tree (Chi, 2004) and prefix tree (Chang, 2003). This algorithm however is based on the lattice tree structure similar to the work of (Chang, 2003). One of the tasks for future work would be comparing and contrasting other tree structures available and verifying the use of lattice for storing intermediate information about itemset supports.

This research concentrates on discovering closed itemsets over a data stream. On arrival of a new transaction, the algorithm will inspect every Itemset in the transaction and update the support counts of related closed itemsets. All the current closed itemsets are maintained in the lattice tree structure. This tree structure always remains resident in main memory so as to reduce memory latency. Consider following set of transactions shown in Table 3-2:

Transaction ID	Items			
1	A	B		
2	A	B	C	
3	A	C		
4	B	C		
5	A	C	D	

Table 3-2: Transactional dataset B

The above set of transactions will result in the closed itemset tree given in Figure 3-5. As mentioned previously we observe that the number of closed nodes (at 8) is significantly less than the number of nodes that a simple prefix tree would have generated (at 15) for the same set of transactions.

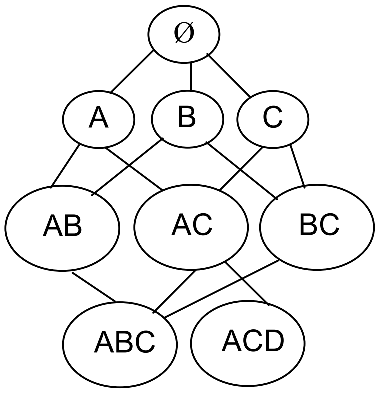


Fig. 3-5 Closed tree

Although most of the algorithms output the closed itemsets as soon as the user requests for the mined results, this algorithm however postpones the output of results until the end of the current frame as determined by the Chernoff bound. This is in order to achieve processing efficiency in terms of computing time and speed. The algorithm could output the mined results whenever user asks for it but this would give rise to a less accurate picture; therefore, as a solution to this issue, the algorithm postpones the output until the expiry of the frame.

However, in the context of a high speed data stream with tens of thousands of transactions arriving per second the time postponement will be negligible in size. Consider a test scenario that we executed in experimentation, where minimum support threshold was 5% and delta threshold was 10%, the average execution time for a frame was 75 milliseconds which is much smaller than the human reaction time which is generally 120 to 160 milliseconds (Kosinski, 2009), suggesting that there will be no practical need in a real time scenario to output instantaneous mined results. Moreover, the higher human response time indicates that the effect of postponing of the mining will be insignificant.

Our approach produces an incremental algorithm where it inspects closed itemsets and updates their support information on the basis of past mined results. This is therefore a more effective approach than those algorithms which scan and generate closed itemsets multiple times. Following routine presents the algorithm used to process the first frame.

Routine: firstPass ()

Parameters:

- Let 'minSup' be the minimum support threshold,
- Let 'frameSize' be number of transactions per frame determined by the Chernoff bound,
- Let 'frequentSingletons' be the list of frequent 1 itemsets found in this frame
- Let H be a hash table contains the identity of frequent itemsets,

Method:

```
1. For index=0 to frameSize
    a. Read the transaction from the data file,
    b. Parse and tokenize the transaction to obtain the singleton items
    c. For all tokens, repeat
        i. If the token is already seen previously then update its
           support information,
        ii. Else, create a node entry in the hash table H for the newly
            obtained token
    d. End for
```

```
2. End For
3. Populate the list 'frequentSingletons' with frequent 1 level items with the help of minSup threshold
```

Description:
The pass through the first frame is primarily to determine frequent singletons which will be used to prune transactions in subsequent frames. Whenever a transaction arrives, it is parsed and tokenized to obtain singleton itemsets. Once, all the singletons in a transaction are identified, they are looked up against a hash table that maintains singletons observed so far. If a particular singleton exists in the hash table then its support information is updated or else a new entry corresponding to the previously unregistered singleton is created. Once all the transactions from pass 1 are processed, a temporary list is populated with the frequent singletons found in the pass; this list will be used for transaction pruning in the next pass.

Routine: secondPass ()
Parameters:

- Let 'minSup' be the minimum support threshold,
- Let 'frameSize' be number of transactions per frame as determined by the Chernoff bound,
- Let LA be the list of frequent singletons found across all previous passes
- Let LC be the list of frequent singletons found in the current pass
- Let LT be the list of singletons of interest in the current transaction
- Let H be a hash table contains the identity of frequent itemsets,

Method:

```
    LC =Φ
1. For index=0 to frameSize
    a. Read the transaction from the data file,
    b. LT= Φ
    c. Parse and tokenize the read transaction to obtain singletons,
    d. For all the singletons, repeat
        i. If the singleton is already seen then update its support information,
        ii. Else, create a node entry in the hash table H for the newly
            obtained singleton
    e. End for
    f. // now get singletons belonging to the current transaction that
       have been //found to be frequent in pass 1
    g. LT= {set of singletons in current transaction} ∩ LA
    h. Generate the powerset of LT
    i. For every element of the powerset, repeat
        i. If the element is already seen then update its support
            information,
        ii. Else, create a node entry in hash table H for the new
            element,
    j. LC = LC U LT
    k. End for
    l. // ensure that items that become frequent in the current pass are
       picked up in // the next pass
    m. LA = LA U LC
```

Description:

This pass is used to find out frequent itemsets to be used for Apriori-like pruning. Steps 1-a to 1-e are similar to that of routine firstPass () which deal with singleton maintenance. Once the first pass is done, all subsequent passes through the data use the LA list to purge infrequent singletons from the current transaction; in this way the pruned transaction contains only frequent singletons seen so far. We do not lose any potential frequent items because even if an item is flagged as being infrequent in the LA list it still has an opportunity to be inserted into the LA list at the end of the pass if it does become frequent in the current pass. This will ensure that such an item is picked up in the next pass.

We now describe a generic routine that is used to mine the stream from the 3rd pass onwards.

Routine: mineData ()

Parameters:

- Let ‘minSup’ be minimum support threshold,

- Let ‘frameSize’ be number of transactions per frame determined by Chernoff bounds,
- Let LC be list of frequent 1 level items found in this pass
- Let LA be list of current frequent singletons found across all frames so far
- Let H be a hash table contains the identity of frequent itemsets,

<u>Method:</u>

```
1. Repeat while data file is not empty,
    a. For index=0 to frameSize,
        i. Read and parse the transaction to generate singletons,
        ii. For every singleton,
            1. If the singleton has already been seen then update
               its support information,
            2. Else, create a node entry in hash table H for the
               newly obtained singleton
        iii. End for
        iv. $LT = LT \cap LA$
    b. Generate the powerset of LT
       // The logic here is first identify frequent itemsets found in previous
       pass, if an itemset is not frequent then it is not viable to generate
       its superset since due to Apriori principle that future superset will
       also be infrequent. //
        i. For every element of powerset, repeat
            1. Check if the element was marked as frequent in
               previous pass,
                a. If frequent then add it to a temporary list T,
                b. Else, continue to the next element in
                   powerset
        ii. End for,
        iii. For all the elements in T, repeat
             // In this step, we find out the common prefixed itemsets.
             Such itemsets, when merged together will be prime
             candidates for being frequent – itemsets AB, AC which
             have already been identified as being frequent when
            merged gives ABC which is a prime candidate for being
            frequent //
                1. Retain only those elements that have common
                   prefixes
                2. Add them to list “powersetElements”
        iv. Goto step(iii) to generate supersets,
        v. For all the elements, in powersetElements list, repeat
            // This step deals with the node creation and maintenance
            for the closed tree.//
                1. If the element exists as node in the Tree, then
                   retrieve the node and update its support information
                2. Else,
                    a. Check whether this node qualifies to be
                       added to the createNode list (refer to the
                       description below for explanation),
                    b. If node qualifies to be added, then check if
                       there exists superset ‘e’ of current node ‘f’. If
                       so, then add node ‘f’ with support =
                       support(e)+1,
            LC = LC U LT
        vi. End for
        vii. Now, the createNode list contains only closed nodes to be
             created with their support information, create entries in tree
             for all the elements from the createNode list,
        viii. This marks the end of the frame,
    c. Prune the tree by deleting children of infrequent nodes, keep
       infrequent nodes so as to give them chance to become frequent in
       future frames,
    d. LA = LA U LC
2. Traverse through the tree and determine Closed frequent itemsets (all
   the nodes that fulfill the minSup threshold will qualify as CFIs),
3. Generate frequent itemsets from CFIs (as explained in section 3.4.5)
4. Output the list of frequent itemsets.
```

Description:

This routine is the central focus of the proposed data stream mining approach. The initial steps 1-a-i to 1-a-iii are similar to that of the routines given earlier for passes 1 and 2. Similar to pass 2, after stripping off the infrequent items from the transaction we generate a powerset (containing itemsets, rather than items), so there is a major difference in powerset generation when compared to passes 1 and 2. In this pass, as in all passes, we make use of the Apriori pruning principle. We use the frequent itemsets found in pass 2 to aid us in this step and it works as follows. After obtaining frequent singletons, we generate itemsets of size 2 from singletons. We then check for existence of any frequent itemset in this size 2 itemset list; if there are any frequent itemsets found then we check for common prefixes (there is no need to generate supersets of two or more itemsets if they do not have any common prefixes). If we get any frequent itemsets composed from common prefixes then all we have to do is to find higher level itemsets from these common prefixed frequent itemsets. This is a recursive procedure which is used to generate all higher level itemsets. Figure 3-6 further explains the use of the Apriori principle to optimize powerset generation.

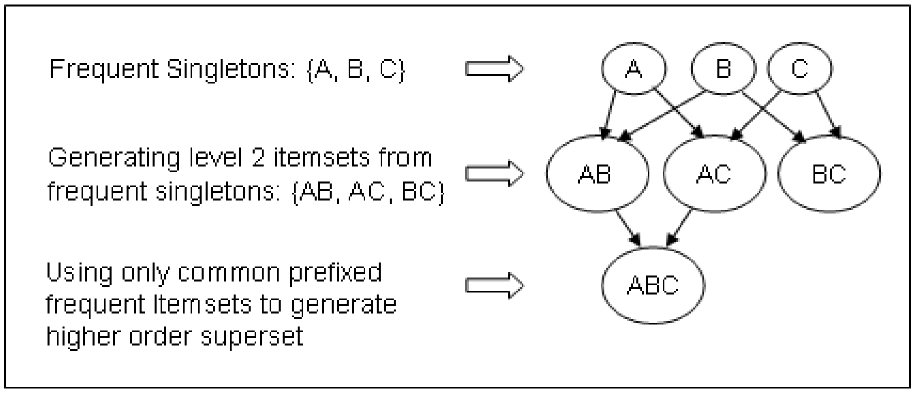

Frequent Singletons: {A, B, C}
Generating level 2 itemsets from frequent singletons: {AB, AC, BC}
Using only common prefixed frequent Itemsets to generate higher order superset
A
B
C
AB
AC
BC
ABC

Fig. 3-6 Powerset Generation

Consider the example shown above where we have A, B, C as frequent singletons; our powerset generation approach first generates their immediate supersets AB, AC, and BC. Once they are generated, we determine the itemsets to be used for higher level powerset generation (common prefixed frequent itemsets). In the above example, assuming that AB and AC are frequent itemsets (and given the fact that they have a common prefix A), they will be used to generate the level 3 itemset that is ABC. In this way we avoid creation of unnecessary and potentially infrequent powerset elements.

Figure 3-7 shows that after the powerset is generated, we check every element of the powerset against the hash table that is superimposed on prefix tree structure.

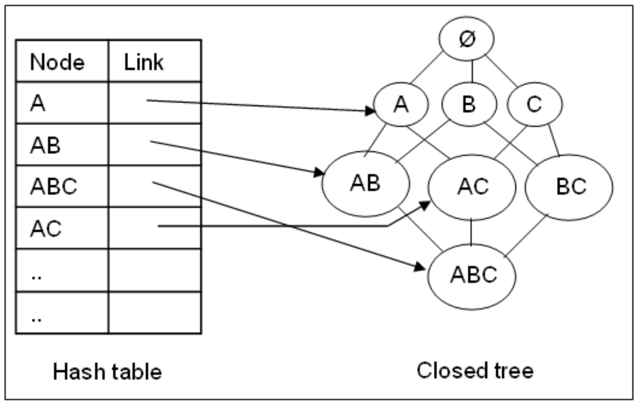


Fig. 3-7 Hash table superimposed on Tree

If an itemset is already seen, then we update its support count or else we add it to a temporary list which is responsible for batch creation of nodes at the end of transaction. While adding an itemset to this list we perform one basic check: we add an itemset to this list only if there does not exist, a superset of this node in the temporary list having the same support. This test is to ensure that we do not create nodes that are not closed. Once the node qualifies to be created, we then check the starting support value that it needs to have; for this we look up the hash table to find a superset, if we find such superset then the starting support of this new node must be support of superset+1 (as explained in algorithm). At the end of the transaction, the temporary list “createNode” has all the nodes that must be created as a consequence of the current processed transaction. Now we traverse through the createNode list and generate nodes for all the entries present in that list. The tests that we just explained ensure that at any moment of time our hash table which is superimposed on a prefix tree contains only closed nodes (frequent as well as infrequent) at any moment of time. These steps are repeated for all the transactions constituting frame (marked by Chernoff bounds value). After the frame expiry, we update the lists and hash tables to reflect latest frequent singletons and itemsets information and prune the children of infrequent closed nodes from the tree; this keeps tree nodes bound and ensures faster traversal through the tree.

Because the Apriori principle is used in generating the powerset, we avoid creation of children of infrequent itemsets similar to Moment (Chi, 2004). At the end of the frame we also undertake pruning. In this phase we traverse through the tree and identify the latest infrequent nodes. Once we have obtained these, we purge their children since for these children nodes to be frequent or even potentially frequent, their parents must become frequent first, therefore it is safe to delete them. Due to these two mechanisms, namely, using Apriori principle for powerset generation and pruning at the frame end, we keep the node creation and maintenance operations under good control.

Since there is no definite end to data stream the above steps are carried out until such time that the user requests for mined results. Whenever the user requests for the output, we traverse through the tree and identify all the closed frequent itemsets.

Consider the following worked example that illustrates the working of the algorithm. The transactions T1, T2, T3, T4 and T5 are considered to arrive in consecutive frames F1, F2, F3, F4 and F5. For the sake of simplicity we use a frame size of 1, although in practice the frame size will be much higher than this. Table 3-3 gives details of the five transactions.

Transaction ID	Items
T1	A B D
T2	A B
T3	A
T4	B C
T5	A B C

Table 3-3: Transactional dataset C

The expansion of the tree occurs as shown in Figure 3-8.

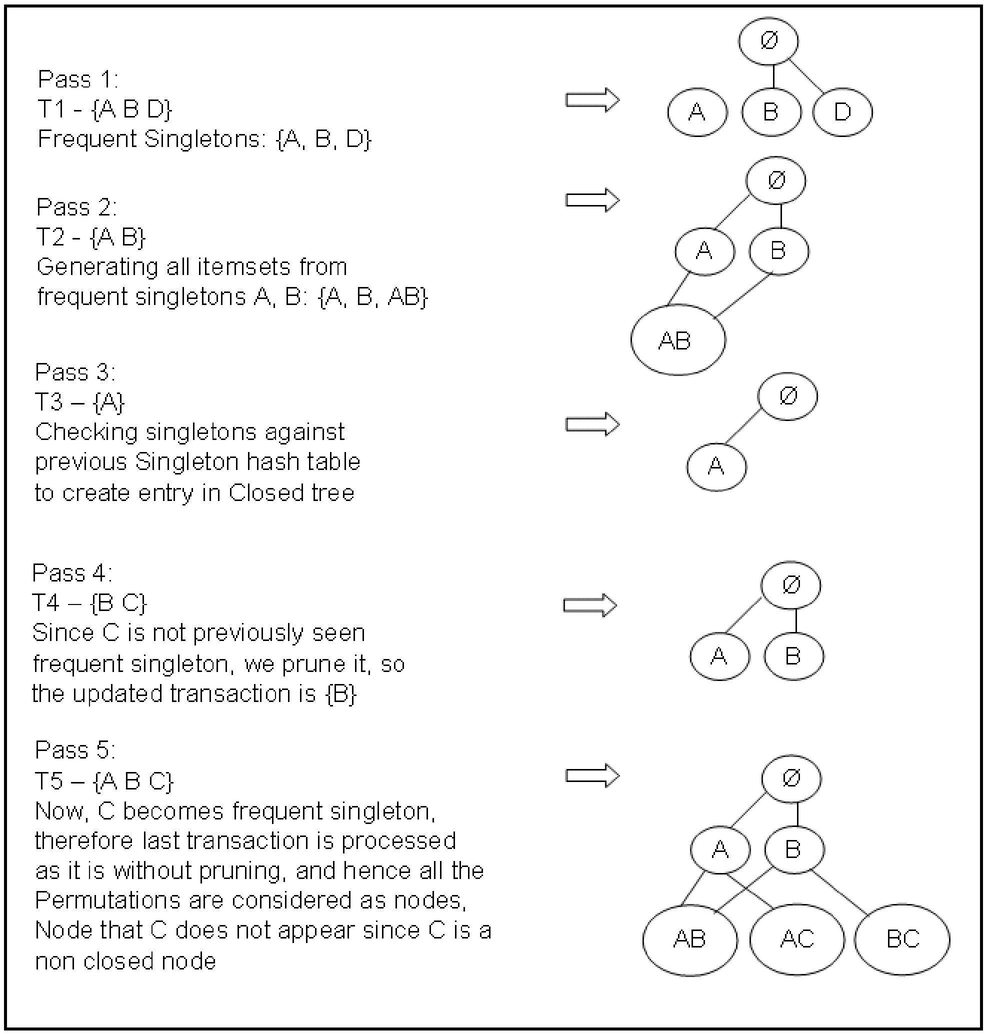

Pass 1:
T1 - {A B D}
Frequent Singletons: {A, B, D}
Ø
A
B
D
Pass 2:
T2 - {A B}
Generating all itemsets from frequent singletons A, B: {A, B, AB}
Ø
A
B
AB
Pass 3:
T3 – {A}
Checking singletons against previous Singleton hash table to create entry in Closed tree
Ø
A
Pass 4:
T4 – {B C}
Since C is not previously seen frequent singleton, we prune it, so the updated transaction is {B}
Ø
A
B
Pass 5:
T5 – {A B C}
Now, C becomes frequent singleton, therefore last transaction is processed as it is without pruning, and hence all the Permutations are considered as nodes, Node that C does not appear since C is a non closed node
Ø
A
B
AB
AC
BC

Fig. 3-8 Closed Tree construction

As explained previously, Pass 1 and 2 do not result in the creation of closed tree, as their role is to determine frequent singletons and frequent itemsets to be used for transaction pruning and powerset reduction. In pass 1, the transaction T1 is treated purely as a collection of singletons and corresponding entries in the hash table that maintains singletons are made. In pass 2, transaction T2 is not needed to be pruned since it does not contain any infrequent singleton, the algorithm then generates all the powerset elements (as shown in the figure above), and these powerset elements are used in next pass to restrict powerset generation. During the pass 3, the actual creation of the closed tree commences. Transaction T3 that contains only A is responsible for the creation of a closed node A. The next transaction T4 is purged to remove item C since it has not been seen previously and therefore results only in creation of another node B in the closed tree. The final transaction T5 contains all the singletons seen previously; therefore no transaction pruning is performed. In the generation of the powerset, items A and B gives rise to higher level elements such as AB, AC, and BC since A, B and C are nodes in the closed tree. One point to note here is that, this will not lead to the creation of powerset element ABC since none of its parents are yet frequent. In this way we limit and optimize powerset generation. The output of pass 5 is mined output with a minimum support threshold of 40%.

3.6 Summary

In this chapter, we have outlined and discussed research approaches and objectives, along with the basic building blocks that constitute our proposed approach for association rule mining in a data stream environment. We provided an overall framework for our research approach as well as detailed algorithms that describe our proposed data stream miner for generating association rules. The following chapters will cover the empirical studies and research findings.

Chapter 4 Experimental Design

4.1 Introduction

This chapter will focus on describing the describing the experiments designed to evaluate the performance of our proposed Data Stream Mining (DSM) algorithm. DSM is compared against a contemporary frequent itemset mining algorithm called the Frequent Pattern Data Miner or FPDM2 (Yu, 2004). Although DSM was designed to mine closed frequent itemsets efficiently, we have extended its capabilities to mine all frequent itemsets by incorporating a routine that determines the frequent itemsets from the closed frequent itemsets (as explained previously in Chapter 3).

The following sections will explain the experimental design along with metrics that we used to compare the performance of DSM against that of FPDM2.

4.2 Datasets

The experimentation is carried out with the help of synthetic datasets that are generated through the use of a dataset generator that is publically available (Agrawal, 1994). Acquiring a real life dataset is quite difficult due to the fact that many organizations refuse to part with their data because of the sensitivity and confidentiality of the data. Therefore, artificial synthetic data generators such as IBM are very commonly used by researchers to evaluate and benchmark their algorithms' performances.

At the same time synthetic dataset generators have the added advantage that various different data characteristics can be captured which may not manifest with certain real word datasets. Such characteristics are the degree of sparsity of the data, variation in the length of the frequent itemsets, the number of items in a transaction and finally the number of transactions itself. Variations in these key data characteristics enable us to test the sensitivity of the algorithms on each of these different conditions.

We have generated three synthetic datasets with the help of IBM generator by varying parameters such as average transaction length, average size of a potential frequent itemset, number of unique itemsets in the generated dataset and the number of transactions in the dataset. Table 4-1 describes the aforementioned values for generated datasets.

Dataset	Average Transaction Size	Average Frequent Itemset size	Unique Items	Number of Transactions
T5I4	5	4	10K	500K
T10I4	10	4	10K	100K
T15I6	15	6	10K	100K

Table 4-1: Parameter Values

The above configurations for datasets are the most commonly used by Data Stream mining researchers such as (Yu, 2004), (Chi, 2004) for benchmarking their proposed algorithms.

4.3 Performance Metrics

The DSM and FPDM2 algorithms are evaluated against certain commonly used performance metrics such as Accuracy (in terms of Recall and Precision), Computational performance (in terms of time taken to process the dataset), and Memory consumption in terms of number of nodes maintained. Recall and precision can be defined as shown in Table 4-2 below (Provost, 1998).

Metrics	Definition
Recall	Recall indicates how many correct frequent itemsets were found.
Precision	Precision indicates how many of frequent itemsets found are correct

Table 4-2: Performance Metrics

The above metrics are further elaborated as below,

$$\text{Recall} = \frac{\{\text{Frequent Itemsets in data}\} \cap \{\text{Retrieved frequent itemsets}\}}{\text{Frequent Itemsets in data}}$$

$$\text{Precision} = \frac{\{\text{Frequent Itemsets in data}\} \cap \{\text{Retrieved frequent itemsets}\}}{\text{Retrieved frequent itemsets}}$$

Fig. 4-1: Recall and Precision formulae

where “Frequent Itemsets in data” corresponds to all the frequent itemsets that are actually present in the dataset (identified using Apriori implementation), and “Retrieved frequent itemsets” are the itemsets reported by algorithms (DSM and FPDM2) as frequent itemsets.

4.4 Support and Reliability

Minimum support is the threshold that determines whether an itemset is of any interest to the end user. The reliability, on the other hand is the probability that the estimated support of an itemset as measured over the frame structure imposed by the Chernoff bound is within a given error margin, $\in$. These two parameters are the key factors that affect performance, given a dataset. Table 4-3 shows the support and reliability values used for the experimentation:

Parameter	Values used
Support	0.1
	0.05
	0.01
Reliability	0.1
	0.05
	0.01

Table 4-3: Key Parameters used in Experimentation

The support range of [.01, 0.1] is typically used by many researchers in association rule mining. Typically, low support values stress association mining algorithms as they give rise to many candidate itemsets, thus requiring large CPU and memory resources.

The reliability measure expresses a probability that the estimate of support produced by the Chernoff bound is in error. A value of r means that the true support value is within an interval of [-∈,∈] with (1-r)*100% probability. The three settings of the reliability parameter allow us to test our results at the 90%, 95% and 99% probability levels. Again, these settings have typically been used in prior research before (Yu, 2004), (Cormode, 2007), (Chuang, 2009).

4.5 Experimental Plan

The experiments were conducted on a Windows XP PC equipped with a 1.7 GHz Intel Pentium IV processor, 1 GB of RAM memory and 40 GB of hard disk space. The algorithms were implemented in the Java programming language. The following sections describe how the different experiments were designed.

4.5.1 Experiment 1

This experiment was mainly designed to compare DSM and FPDM2 with respect to the previously explained performance metrics. For this experiment, we have used the T10I4 and T15I6 datasets which are relatively dense when compared to T5I4 used in the previous experiment. The following steps were executed in this experiment:

a) For the T10I4 dataset, vary the minimum support parameter while keeping delta constant,
b) For theT10I4 dataset, vary the delta parameter while keeping minimum support constant,
c) Repeat steps a and b for the T15I4 dataset
d) Measure the accuracy, computational performance and memory consumption for each of the steps a), b) and c) above.

4.5.2 Experiment 2

This experiment was sketched to determine the effect of concept drift (Wang, 2003) on the performance of DSM and to compare it against FPDM2. Here, we developed a routine that simulated the effect of concept drifts with the help of a

Java construct called Random Generator. This random generator (on the basis of an input parameter that governed the degree of concept drift) randomly selects a subset of previously found frequent singletons whose effect will be diminished by removing them from transactions to exhibit the effect of concept drift. In this experiment, we have considered two scenarios, first, where the data stream suffers from concept drift from the beginning, and second, where the concept drift occurs after several frames of transactions are processed. The simulation of concept drift can be understood from the following routine,

<u>Routine: conceptDrift ()</u>

<u>Parameters:</u>

- Let 'cDrift' be a flag indicating that user has requested to simulate effect of concept drift,
- Let 'cThreshold' be the degree of concept drift

<u>Method:</u>

```
1. Read user input corresponding to concept drift and assign it to cDrift,
2. Check if cDrift= True, then
    a. For all the transactions, do following,
        i. Find out all the singletons present in this transaction,
            1. For all the singletons, do following,
            2. Call a random generator to obtain a random number
               'randNum' corresponding to the current singleton,
            3. If randNum ≥ cThreshold then do not remove this
               singleton from the consideration,
            4. Else, eliminate this singleton from the transaction to
               be processed (to routine mineData()),
            5. End If
            6. End For
    b. End for
3. Else, continue processing of all the transactions as explained in
   mineData()
6. End If
```

Description:

This routine is an ongoing check for all the transactions. When the user initiates the algorithm, an input is received from the user indicating whether or not the user intends to enable the concept drift simulation. If the user has requested to activate the concept drift, then a small block of code is called before processing every singleton of a transaction. This block of code generates a random number that is checked against the concept drift threshold cThreshold. If the random number generated qualifies (that is greater or equal to cThreshold), then this singleton of the transaction can be processed, else, the current singleton is removed from the transaction.

This block of code works in juxtaposition with the function that prunes the transaction from the infrequent singletons. While purging an incoming transaction of infrequent singletons, this routine ensures that only selected frequent singletons are advanced for further processing. The selection process targets frequent singletons at random.

We used the T10I4 dataset for this experiment and the following steps were undertaken:

a) For DSM, we varied the degree of concept drift from 0% to 60% in steps of 20% while keeping the support and delta parameters constant,
b) Recorded recall and Precision for DSM,
c) Repeated steps a) and b) for FPDM2.

4.5.3 Experiment 3

The underlying intention of this experiment is to understand the effects of sparsity on DSM performance. For this experiment, we have used the T5 I4 dataset which is sparse in nature (average size of transaction being 5 items per transaction). Since the dataset is sparse we needed a significant number of transactions to ensure the reliability of the mined results. The experiment carried out the following:

a) Varied the minimum support threshold given as input parameter while keeping reliability (delta) constant
b) Measured the performance in terms of accuracy, computational performance and memory consumption.

4.6 Summary

This chapter presented the reasoning and the structure of the empirical study that we have used to evaluate the performance of the suggested algorithm. It also presented a description of the datasets and the performance metrics with which the assessment is to be carried out.

Chapter 5 Research Findings

5.1 Introduction

The previous chapters have described the fundamental background behind closed itemset mining, research objectives, overall architecture, and experimental design. This chapter will focus on the experimental findings.

Both DSM and FPDM2 were tested on synthetic datasets and compared against predefined performance metrics such as Accuracy, Computational Performance, and Memory consumption.

5.2 Findings from Experiment 1

This experiment was mainly designed for comparing DSM and FPDM2 with respect to performance. We first varied the minimum support threshold while keeping the delta parameter constant. We recorded the accuracy, performance and memory consumption for DSM and then repeated the procedure for FPDM2. For this experiment, we have used T10I4 and T15I6 dense datasets generated using the IBM data generator (IBM). The Recall and Precision were calculated by comparing DSM and FPDM2 results against the Apriori implementation (http://www.borgelt.net/apriori.html).

The Apriori implementation was run against data batched across a fixed number of frames and presented as a single unit fixed size dataset to the Apriori algorithm. It should be stressed that Apriori was only used for benchmarking purposes for the Precision and Recall values. As mentioned earlier in the thesis Apriori cannot be used in an actual data stream environment.

For the dataset T10I4, Table 5-1 gives the findings.

		DSM		FPDM2	
		Recall	Precision	Recall	Precision
Support	Delta				
0.1	0.1	100%	100%	100%	89%
0.05	0.1	100%	100%	100%	93%
0.01	0.1	100%	100%	100%	94%

Table 5-1: Accuracy with fixed Delta-Experiment 1-1

DSM and FPDM2 performed equally well on Recall for this experiment; however, in terms of Precision, DSM outperformed FPDM2 (as shown in above table). FPDM2 reported an increased false positive rate, reporting more frequent nodes than what exists in reality. This is basically because FPDM2 periodically prunes its primary and secondary pool, removing all the support information of the pruned itemsets. Thus when these itemsets are seen again, their support information is incomplete and inaccurate, resulting in marking infrequent nodes as frequent and vice versa (as seen in next experiment results).

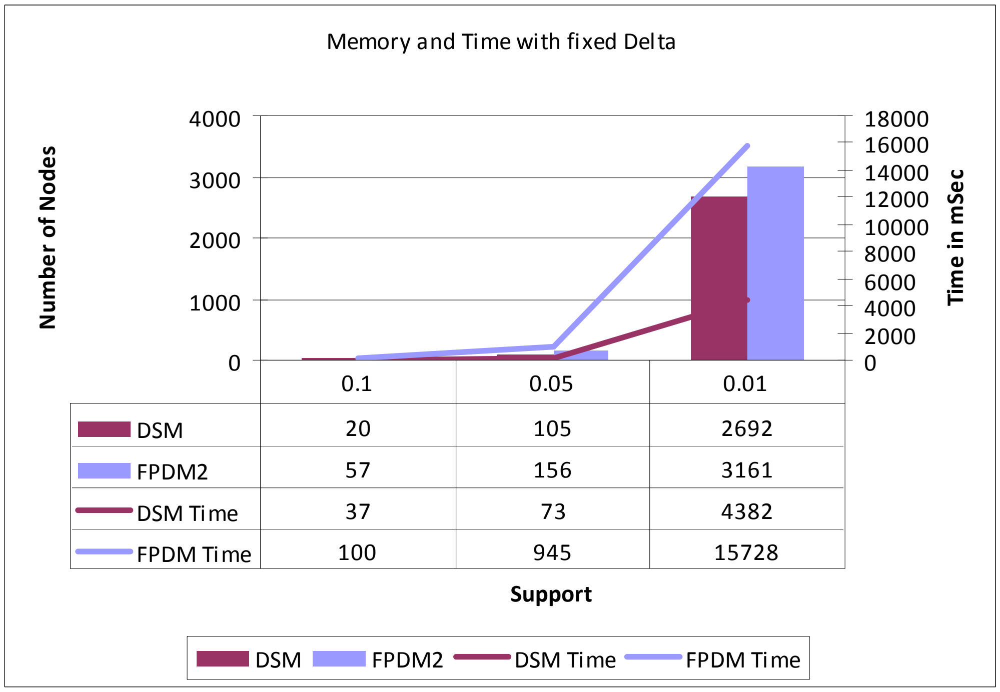
We then repeated the same procedure with varying delta and keeping the minimum support threshold constant at 0.05. Here, DSM outperformed FPDM2 in terms of Recall as well as Precision,

		DSM		FPDM2	
		Recall	Precision	Recall	Precision
Delta	Support				
0.1	0.05	100%	100%	100%	93%
0.05	0.05	93%	100%	80%	100%
0.01	0.05	100%	100%	80%	100%

Table 5-2: Accuracy with fixed Support- Experiment 1-2

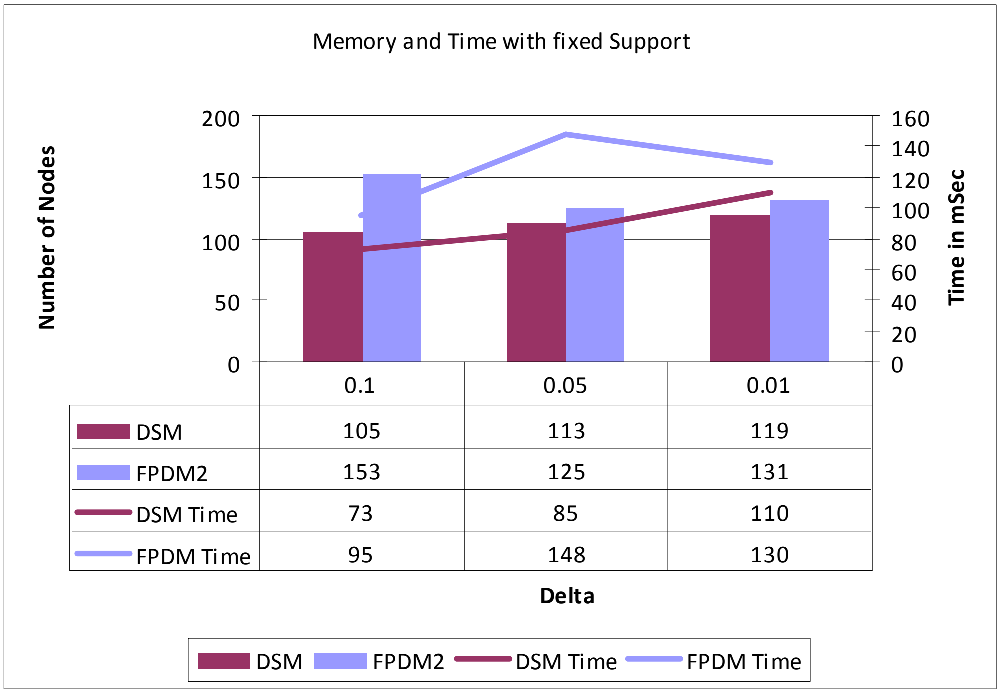
As the delta threshold was lowered, FPDM2's Recall suffered whereas DSM continued to outperform FPDM2. In terms of Precision, initially FPDM2 had poor Precision for the higher value of delta, but as the delta threshold was lowered, it matched DSM's Precision. The frame size is inversely proportional to the delta value (as shown in section 3.4.2). As delta decreases, the frame size increases and the Chernoff estimate of support becomes more reliable, causing a lesser number of frequent itemsets to be incorrectly deleted from FDPM2's secondary pool which keeps track of frequent itemsets over a period of time (as opposed to the primary pool that keeps track of the frequent itemsets found in current frame). This gives rise to an improvement in FDPM2's Precision.

Next, we evaluated the memory and computational performances of DSM and FPDM2. DSM continued to outperform FPDM2 on both these metrics.



	0.1	0.05	0.01
DSM	20	105	2692
FPDM2	57	156	3161
DSM Time	37	73	4382
FPDM Time	100	945	15728

Fig. 5-1 Memory and time with fixed delta-Experiment 1-1



	0.1	0.05	0.01
DSM	105	113	119
FPDM2	153	125	131
DSM Time	73	85	110
FPDM Time	95	148	130

Fig. 5-2 Memory and time with fixed support-Experiment 1-2

DSM computes the frequent itemsets only when the user requests for mined results; until such time DSM continues to process and maintain closed frequent itemsets. Since the number of closed frequent nodes is significantly lesser than frequent nodes, maintaining and processing takes a lesser amount of time. Therefore FPDM2 consumes more space and execution time for maintaining and reporting frequent itemsets as compared to DSM which is based on closed frequent itemset mining.

We next ran experiments on the T15I6 dataset, and the results were significantly different. For constant delta and varying support, both DSM and FPDM2 performed equally well on Recall and Precision; whereas for constant support and varying delta, DSM outperformed FPDM2 in terms of Recall but recorded similar Precision to FPDM2.

Tables 5-3 and 5-4 highlight the findings of this experimental setting,

		DSM		FPDM2	
		Recall	Precision	Recall	Precision
Support	Delta				
0.1	0.1	100%	100%	100%	100%
0.05	0.1	100%	100%	100%	100%
0.01	0.1	100%	100%	100%	100%

Table 5-3: Accuracy with fixed Delta-Experiment 1-3

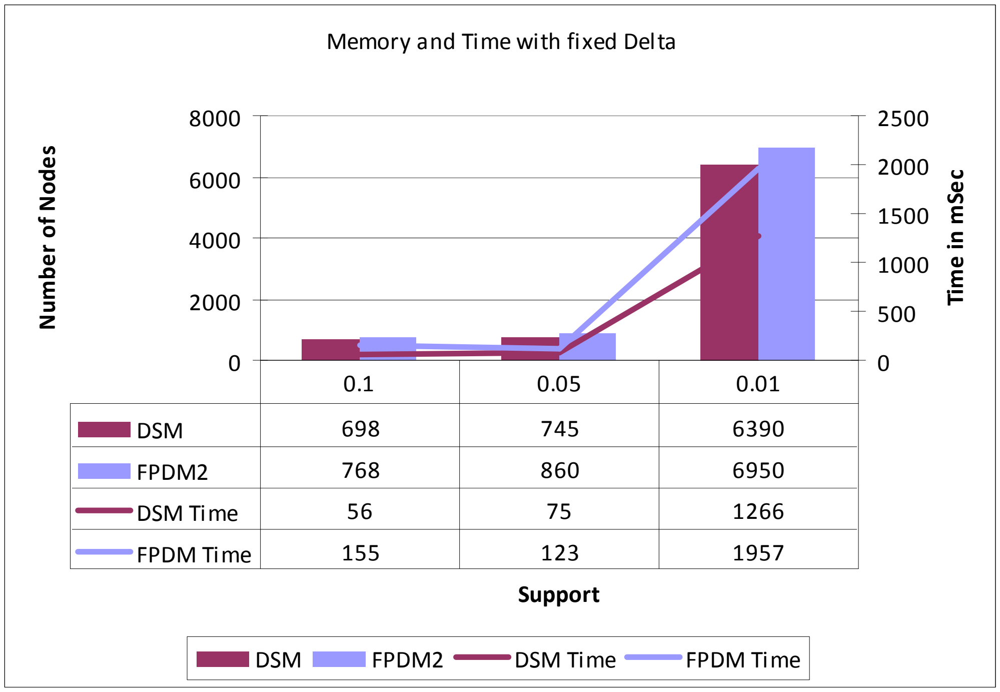
		DSM		FPDM2	
		Recall	Precision	Recall	Precision
Delta	Support				
0.1	0.05	100%	100%	100%	100%
0.05	0.05	93%	100%	89%	100%
0.01	0.05	100%	100%	81%	100%

Table 5-4: Accuracy with fixed Support-Experiment 1-4

The T15I6 dataset was significantly denser than T10I4, and therefore the loss of support information that FPDM2 suffered in a less dense dataset was not as influential, resulting in better Precision and Recall. In the case of fixed support, the density of the dataset helped FPDM2 to result better recall than its Recall in similar settings with T10I4 dataset.

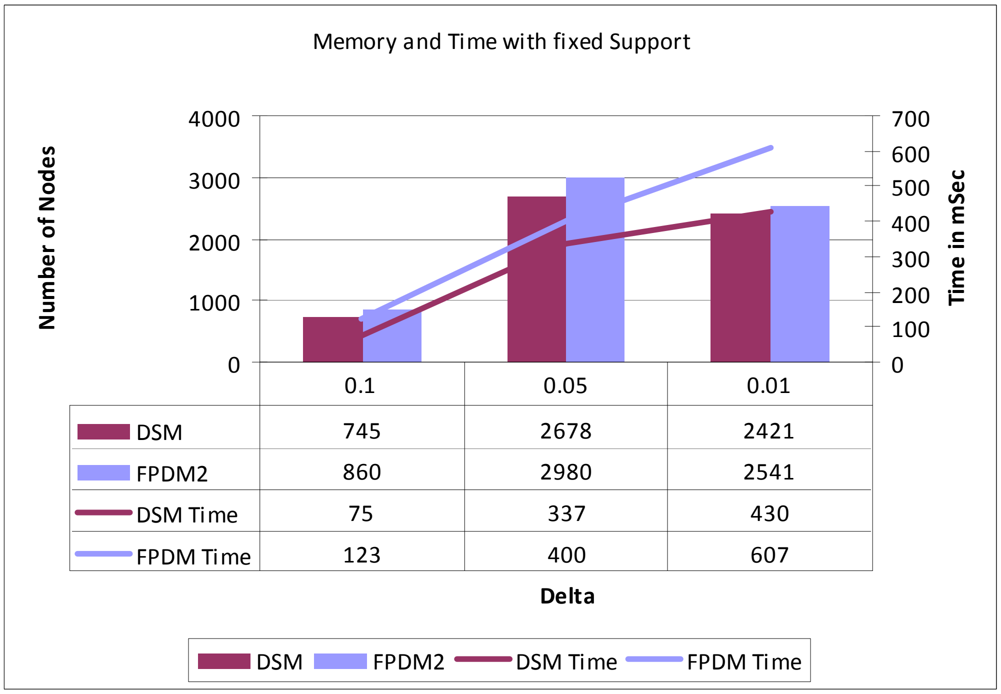
DSM and FPDM2 were again tested against computational performance and memory consumption for this dataset and results were similar to the results obtained for the T10I4 dataset. DSM was marked as better in terms of number of nodes created as well as average time taken to process a frame of transactions as determined by the Chernoff Bound.

Figures 5-3 and 5-4 below report the execution time and number of nodes created for this experiment,



	0.1	0.05	0.01
DSM	698	745	6390
FPDM2	768	860	6950
DSM Time	56	75	1266
FPDM Time	155	123	1957

Fig. 5-3 Memory and time with fixed delta-Experiment 1-3



	0.1	0.05	0.01
DSM	745	2678	2421
FPDM2	860	2980	2541
DSM Time	75	337	430
FPDM Time	123	400	607

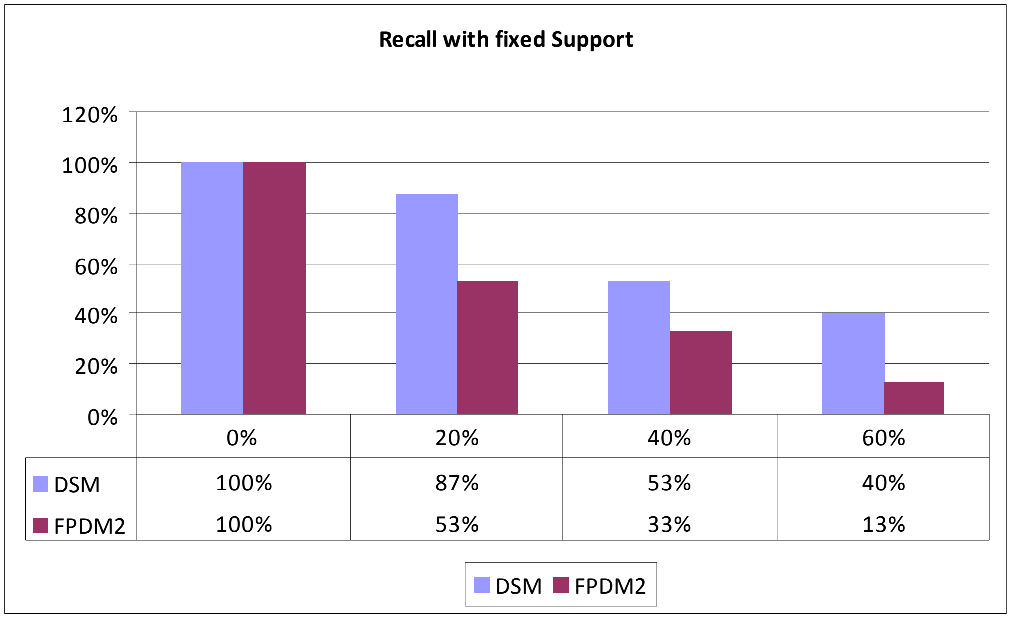
Fig. 5-4 Memory and time with fixed support-Experiment 1-4

As it is clear from the graph, DSM generates lesser nodes than FPDM2 and finishes its frames significantly earlier than FPDM2. DSM has continued to outperform FPDM2 in terms of computational time and space is mainly because, DSM is based on the closed frequent itemset based approach, maintaining closed frequent nodes has proven to be much more efficient than maintaining a huge collection of frequent nodes. Once DSM has mined closed frequent itemsets, generation of frequent itemsets from CFIs (as explained before) is a significantly easy, quick task and is carried out only once while presenting output to the user. This saves a significant amount of memory and CPU time. FPDM2 does not have this liberty and therefore suffered as compared to DSM.

5.3 Findings from Experiment 2

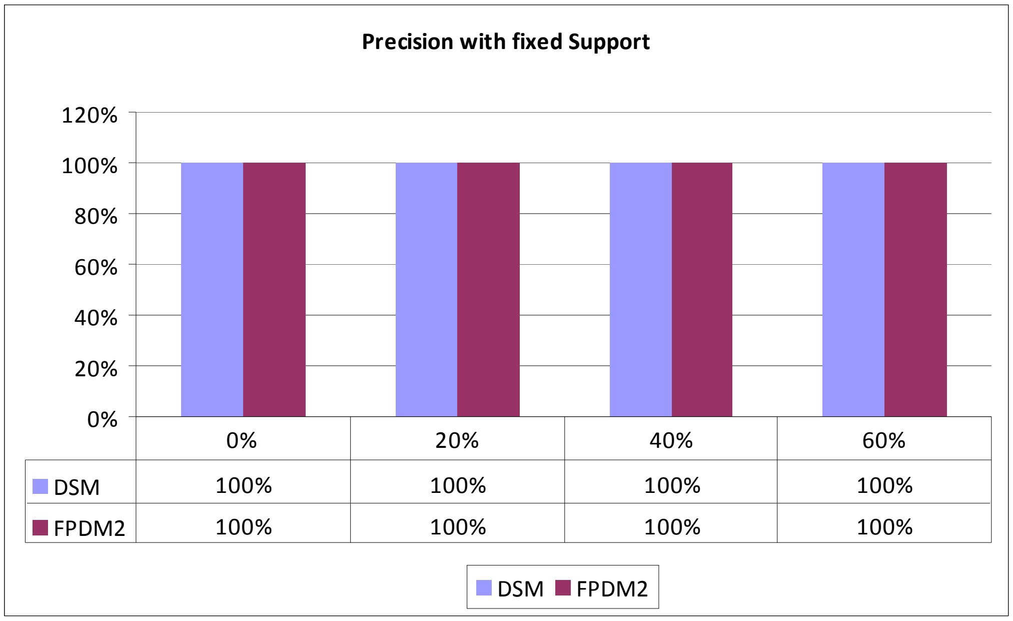
Experiment 3 was carried out to test the effects of concept drift on accuracy of DSM. To carry out this experiment, a small routine was developed which with the help of Random generator (a Java construct). This routine randomly selects whether to process a singleton from the transaction or not (as explained before in experimental design). In this experiment, we have considered two settings, one, where the data stream exhibits the effects of the concept drift from the beginning (that is, we call the routine that deals with concept drift simulation from pass 3 onwards), and second, where the concept drift begins to demonstrate after a larger number (for this dataset, the drift was triggered after 50th frame) of passes of transactions are processed.

The findings of the first setting are as shown in Figure 5-5 and 5-6.



	0%	20%	40%	60%
DSM	100%	87%	53%	40%
FPDM2	100%	53%	33%	13%

Fig. 5-5 Recall with fixed support-Experiment 2-1



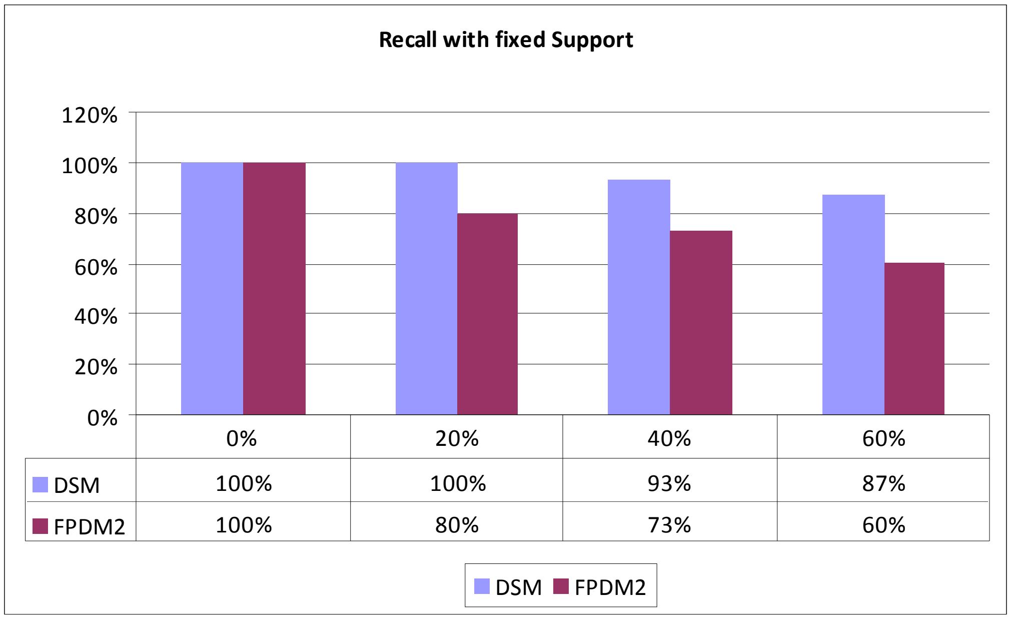
	0%	20%	40%	60%
DSM	100%	100%	100%	100%
FPDM2	100%	100%	100%	100%

Fig. 5-6 Precision with fixed support-Experiment 2-2

As it is clear from above figures, FPDM2 severely underperformed DSM in terms of Recall. At 20% of the concept drift, FPDM2 recorded a recall of 53% whereas DSM registered 87% Recall. As we continued to increase the degree of concept drift, FPDM2 continued to degrade its Recall steeply while DSM's Recall reduced more gradually. In terms of Precision, however, both DSM and FPDM2 performed equally well, recording 100% precision on all levels of concept drift indicating that all the presented itemsets were undoubtedly frequent itemsets. The FPDM2 reported significantly lesser Recall because, due to concept drift, a lesser number of the nodes obtained a chance to prove themselves frequent and more of the nodes were pruned (since their support was not large enough to qualify as being frequent) in FPDM2, resulting in a sharp drop in Recall. The lowest recorded Recall was 13% for FPDM2 whereas DSM delivered 40% Recall at 60% of concept drift. DSM's performance was not as affected as FPDM2 because DSM, instead of deleting the infrequent nodes, only purges their descendants' branch in the closed tree, thus allowing a potentially frequent node to prove itself in future frames.

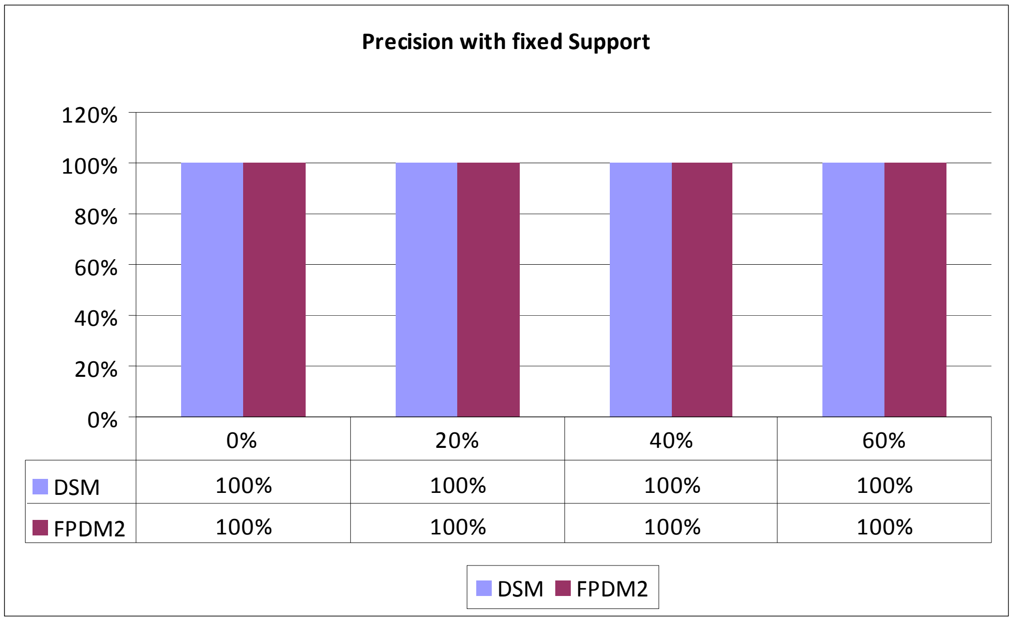
Both algorithms reported 100% Precision because the itemsets that were reported as frequent itemsets were having high enough support to be unaffected by the degree of concept drift.

The outputs of second setting were better than those of first setting in the case of Recall, Figures 5-7 and 5-8 report the outcomes of the second setting.



	0%	20%	40%	60%
DSM	100%	100%	93%	87%
FPDM2	100%	80%	73%	60%

Fig. 5-7 Recall with fixed support-Experiment 2-3



	0%	20%	40%	60%
DSM	100%	100%	100%	100%
FPDM2	100%	100%	100%	100%

Fig. 5-8 Precision with fixed support-Experiment 2-4

In the second setting, the effects of concept drift were exhibited after few frames have been processed by both the algorithm. The dataset used was T10I4 having 100K transactions and the concept drift routine was called after processing half of the transaction base, that is, after 50K transactions were processed. This setting recorded improved results in DSM primarily because, since the concept drift began exhibiting its effect (that is when the routine started randomly selecting set of previously seen frequent singletons and removing them from the transaction), the algorithm had seen enough of the data to build its baseline and keep on processing incrementally. At 20% of concept drift DMS recorded 100% recall, the lowest recorded value for DSM in this setting was 87% whereas in previous setting this value was 40%. FPDM2 as well performed better than it had performed in the previous setting, since FPDM2 also got a chance to observe the data stream for several frames, the effect of future concept drift was not heavily detrimental, the highest and lowest recall FPDM2 reported were 80% and 60% respectively (significantly better than previous values of 53% and 13% respectively).

5.4 Findings from Experiment 3

The previous two experiments clearly established the superiority of DSM over FDPM2. We were thus interested in assessing how DSM would perform on a sparse dataset. The output of DSM was cross-checked against an Apriori implementation provided at http://www.borgelt.net/apriori.html as in the previous two experiments. Table 5-5 displays the result for the T5I4, a sparse dataset.

Support	Delta	Recall	Precision
0.007	0.1	86%	100%
0.006	0.1	94%	100%
0.005	0.1	94%	98%
0.004	0.1	96%	99%
0.003	0.1	97%	100%

Table 5-5: Accuracy findings-Experiment 3

We t varied the support threshold values in a tight range from 0.007 to 0.003, as shown in the Table 5-5. Support values above the upper threshold of 0.007 produced very few frequent itemsets. As the minimum support threshold was lowered, Precision and Recall tended to improve. As transaction pruning is done on the basis of historical information, higher support levels give rise to more aggressive pruning of transactions, resulting in some frequent itemsets being pruned. This results in lower Recall values at the high end of the support range.

Next, we traced DSM in terms of Memory consumption and Computational performance; the findings are as shown in Table 5-6.

Support	Delta	Nodes	Time (mSec)
0.007	0.1	479	25
0.006	0.1	651	61
0.005	0.1	1572	152
0.004	0.1	4372	747
0.003	0.1	16757	7587

Table 5-6: Performance findings-Experiment 3

The findings in terms of memory consumption and execution time were in line with past research; as we lower the minimum support threshold, the number of nodes created in the closed tree increased and so did the execution time for mining. As in (Yu, 2004) and the experiment described above, the reduction in the support threshold widened the perimeter of nodes that are considered in future frames and resulted in lesser degree of pruning, thus giving rise to an increase in the number of nodes. The increase in the number of nodes also gave rise to an increase in execution time since more number of nodes had to be maintained.

Following figure 5-9 details the rise in the number of nodes created and corresponding increase in execution time,

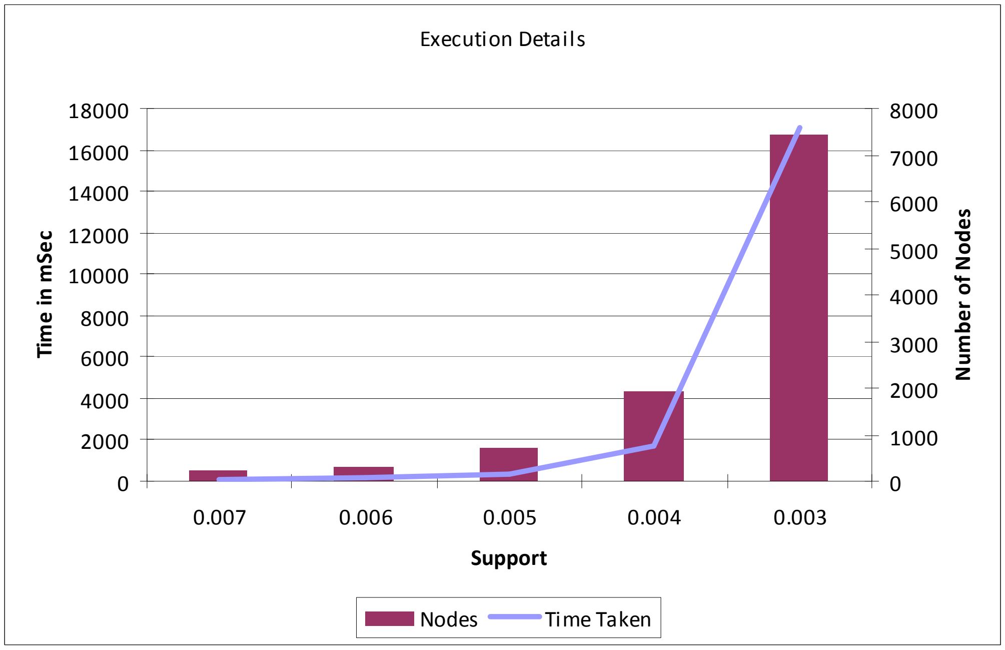


Fig. 5-9 Execution details- Experiment 3

5.5 Summary

This chapter has presented the research findings and analysis of the experimental outputs. We have used graphs and tables whenever appropriate to explain the outputs in more detail.

All the experiments have proved that our proposed algorithm DSM has outperformed FPDM2 (Yu, 2004) in terms of all the performance metrics during all the experiments. DSM has also proved itself to be promising in dealing with the phenomenon of concept drift which is quite often observed in real-world data streams.

Chapter 6 Future Work

Although this research has proposed a generic algorithm for mining closed frequent itemsets from a high speed data stream, there is a great deal of scope for further development of this algorithm.

One possible extension of this research would be an investigation of the benefits of deferred updates. Deferring the updates means postponing the updates to the support information until the expiry of the current frame. At the expiry of the frame all updates are batched together and then applied as one single operation, rather than updates being applied for each transaction. This mechanism offers the opportunity of reducing CPU overhead by reducing the number of update operations on the closed tree, which is the current performance bottleneck in the DSM algorithm.

However, this deferred update mechanism poses a few challenges as well. One such challenge is to avoid revisiting nodes to avoid redundant (and in some cases spurious) updates. Another problem is to keep track of the deferred support value that must be added to every node's support count. These issues must be handled as part of implementing the deferred update strategy. As a part of future work, we will be addressing these issues and analyzing the effects of deferring updates on the accuracy of the mined results.

Another area of the future research would be to estimate, rather than to measure the support information of the itemsets (Laur, 2005). Estimating the support rather than updating every itemset on its occurrence can be significantly rewarding. Along with support estimation, a closure-property-check, similar to (Jiang, 2006) (which uses Galois operator, a commonly used mathematical function as a closure operator) to determine whether an itemset is likely to be closed itemset or not can further increase gains. The future research would therefore be scrutinizing this combination of the support estimation and a closure-properly-check.

Future work will also explore possibilities to further optimize the powerset generation that the algorithm has to perform while processing the input transactions. Despite the heuristics used to trim transactions and reduce the size of the powerset generated, there will always be a low enough support threshold that causes the number of frequent items in a transaction to be large. This in turn will give rise to correspondingly larger powersets. It will thus be profitable to look for more efficient ways to handle powerset creation by exploiting previously seen information.

Yet another direction for future research is to exploit the concept of maximal frequent itemset mining. Maximal itemset are an even more compact representation of frequent itemsets then closed frequent itemsets. There has been a great deal of research in Maximal frequent itemset mining (Gouda, 2001), (Yang, 2004), (Ao, 2007). Our future research will also try to look into the opportunities and challenges that Maximal frequent itemset mining holds in a data stream environment.

Chapter 7 Conclusion

Data stream mining is one of the most intensely investigated and challenging research domains in contemporary research in the data mining discipline as a whole. The peculiarities of data streams render conventional mining schemes inappropriate.

In this research, we presented an overview of a novel approach for mining the closed itemsets from a data stream. We have implemented an efficient closed prefix tree to store the intermediate support information of frequent itemsets. Moreover, we have employed the Apriori principle to reduce unnecessary powerset creation along with transaction pruning to further enhance transaction processing. We developed an incremental closed itemset mining algorithm based on the probabilistic guarantees of Chernoff bounds.

The Chernoff bound helps in purging unnecessary itemsets from the data stream and keeps the memory requirements within reasonable bounds. We compared our approach with the FPDM2 algorithm proposed by Yu (2004). Although FPDM2 is a frequent itemset mining algorithm and DSM a closed frequent itemset mining algorithm, a basic routine computing frequent itemsets from closed frequent itemsets enabled us to assess the performance of both the approaches on the uniform grounds.

Our experimentation showed that DSM not only outperformed FPDM2 in all the experimental settings but also excelled during the test for the effects of concept drift. The next step in the research would be advancing the investigation into the future research areas highlighted in the previous chapter.

References

Agrawal, R., Srikant, R. (1994). *Fast algorithms for mining association rules.* Paper presented at the 20th International Conference on Very Large Data Bases (VLDB'94), Santiago, Chile.

Ao, F., Yan, Y., Huang, J., Huang, K. (2007). Mining maximal frequent itemsets in data streams based on FP trees. *Springer Verlag Berlin Heidelberg*, 479-489.

Ben-David, S., Gehrke, J., Kifer, D. (2004). *Detecting change in data streams.* Paper presented at the 30th VLDB Conference, Toronto, Canada.

Burrel, G., Morgan, G (1979). *Sociological paradigms and organizational analysis*. London: Heinemann.

Celgar, A., Roddick, J. (2006). Association mining. *ACM Computing Surveys, 38*(2), 1-42.

Chang, J., Lee, W. (2003). *Finding recent frequent itemsets adaptively over online data streams.* Paper presented at the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA.

Charikar, M., Chen, K., Colton, M. (2004). Finding frequent items in data streams. *Theoretical Computer Science*, 1-11.

Cheung, D., Han, J., Vincent, T., Wong, C. (1996). *Maintenance of discovered association rules in large database: An incremental updating technique.* Paper presented at the IEEE International Conference on Data Mining, New York, USA.

Chi, Y., Wang, H., Yu, P., Muntz, R. (2004). *Moment: Maintaining closed frequent itemsets over a stream sliding window.* Paper presented at the IEEE International Conference on Data Mining, Brighton, UK.

Chuang, K., Chen, H., Chen, M. (2009). Feature-preserved sampling over streaming data. *ACM Transactions on Knowledge Discovery from data, 2*(4), 15-60.

Collis, J., Hussey, R. (2003). *Business Research*. Basingstoke, UK: Palgrave Macmillan.

Cormode, G., Garofalakis, M. (2007). *Sketching probabilistic data streams*. Paper presented at the SIGMOD'07.

Dash, N. (2005). Selection of the Research Paradigm and Methodology. *Online Research Methods Resource*. Retrieved from http://www.celt.mmu.ac.uk/researchmethods/Modules/Selection_of_methodology/index.php

Gaber, M., Zaslavsky, A., Krishnaswamy, S. (2005). Mining data streams: A review. *ACM SIGMOD Record, 34*(2), 18-26.

Giannella, C., Han, J., Pei, J., Yan, X., Yu, P. (2003). Mining frequent patterns in data stream at multiple time granularities. In *Next Generation Data Mining* (pp. 105-124).

Gouda, K., Zaki, M. (2001). *Efficiently mining maximal frequent itemsets*. Paper presented at the 2001 IEEE International Conference on Data Mining.

Huang, H., Wu, X., Relue, R. (2002). *Association analysis with one scan of databases.* Paper presented at the IEEE International Conference on Data Mining, Maebashi City, Japan.

IBM. IBM data generator, from http://www.almaden.ibm.com/cs/projects/iis/hdb/Projects/data_mining/datasets/syndata.html

Jiang, N., Gruenwald, L. (2006). *CFI-stream: Mining closed frequent itemsets in data streams.* Paper presented at the KDD, Philadelphia, USA.

Jiang, N., Gruenwald, L. (2006). Research issues in data stream association rule mining. *ACM SIGMOD Record, 35*(1), 14-19.

Laur, P., Nock, R., Symphor, J., Poncelet, P. (2005). *On the estimation of frequent itemsets for data streams: Theory and experiments*. Paper presented at the CIKM.

Lee, S., Cheung, D., Shan, M. (1997). Maintenance of discovered association rules: When to update. *Research Issues on Data Mining and Knowledge Discovery*, 1-8.

Li, L. (2003). *A graph-based algorithm for frequent closed itemset mining.* Paper presented at the 2003 IEEE Systems and Information Engineering Design Symposium, Charlottesville, VA.

Lin, C., Chiu, D., Wu, Y., Chen, A. (2005). *Mining frequent itemsets from data streams with a time-sensitive sliding window.* Paper presented at the SIAM International Conference on Data Mining, California, USA.

Manku, G., Motvani, R. (2002). *Approximate frequency counts over data streams.* Paper presented at the 28th International Conference on Very Large Data Bases, Hong Kong, China.

Moonesinghe, H., Fodeh, S., Tan, P. (2006). *Frequent closed itemset mining using prefix graphs with an efficient flow-based pruning strategy.* Paper presented at the 6th International Conference on Data Mining (ICDM'06), HongKong, China.

Pang, T., Steinbach, M., Kumar, V. (2006). *Introduction to Data Mining*: Addison-Wesley.

Provost, F., Fawcett, T., Kohavi, R. (1998). *The case against accuracy estimation for comparing induction algorithm.* Paper presented at the Fifteenth International Conference on Machine Learning, Madison, Wisconsin USA.

Ravikumar, P., Lafferty, J. (2004). *Variational chernoff bounds for graphical models.* Paper presented at the 20th conference on Uncertainty in artificial intelligence AUAI '04 Edinburgh, Scotland.

Uno, T., Kiyomi, M., Arimura, H. (2004). *Efficient mining algorithms for frequent/closed/maximal itemsets*. Paper presented at the IEEE ICDM Workshop on Frequent Itemset Mining Implementations.

Wang, H., Wei, F., Yu, P., Han, J. (2003). *Mining concept-drifting data streams using ensemble classifiers.* Paper presented at the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA.

Wen, L. (2004). An efficient algorithm for mining frequent closed itemset. *IEEE*, 4296-4299.

Yang, G. (2004). *The complexity of mining maximal frequent itemsets and maximal frequent patterns*. Paper presented at the KDD.

Yang, L., Sanver, M. (2004). *Mining short association rules with one database scan.* Paper presented at the International Conference on Information and Knowledge Engineering, Las Vegas, Nevada.

Yu, J., Chong, Z., Lu, H., Zhou, A. (2004). *False positive or false negative: Mining frequent itemsets from high speed transactional data streams.* Paper presented at the 30th VLDB Conference, Toronto, Canada.

Zaki, M., Hsiao, C. (2002). *Charm: An efficient algorithm for closed itemset mining.* Paper presented at the 2nd International Conference on Data Mining, Arlington, USA.

Zaki, M., Gouda, K. (2001). *Fast vertical mining using diffsets*: Rensselaer Polytechnic Institute

Zaki, M., Li, W., Ogihara, M. (1997). Evaluation of sampling for data mining of association rules. *IEEE*, 42-50.

Zheng, Q., Xu, K., Ma, S. (2003). *When to update the sequential patterns of stream data.* Paper presented at the Pacific-Asia Conference on Knowledge Discovery and Data Mining Seoul, Korea.