Near Closed Frequent itemsets to accelerate the generation of association rules in a data stream environment

Nathaphong Viriyarattanaporn
(Nat)

A dissertation submitted to
Auckland University of Technology
In partial fulfilment of the requirements for the degree of
Master of Computer and Information Sciences (MCIS)

2010

School of Computer and Mathematical Sciences

Supervisor: Dr Russel Pears

Table of Contents

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 or diploma of another university or institution of higher learning, except where due acknowledgements are made.”

Yours Sincerely,

(Nathaphong Viriyarattanaporn)

Acknowledgements

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

I also would like to thank Omkar Kadam for his help and support on his previous research.

Lists of Abbreviations and Terms

- DSM: Data Stream Mining
- NCN: Near Closed Node
- FI: Frequent Itemsets
- CFI: Closed Frequent Itemsets
- Datasets: or database contained multiple transaction use in data mining.
- Closed Itemset/Node: Refers to itemset/Node; None of its supersets have exactly the same support as itself.
- Maximal Itemsets/Nodes: Refers to itemset/Node; None of its supersets are frequent.
- Subset: Term use to make reference to all the parent nodes of tree node.
- Superset: Term use to make reference to all the children nodes of tree node.
- Powerset: Term use to make reference to all possible node combinations of a particular tree node.
- Precision: The accuracy proportion of generated frequent itemsets
- Recall: The proportion of generated frequent itemsets against the actual number of frequent itemsets
- minSup: Minimum support is minimum frequency of occurrence for an item or itemset that is required for it to be identified as frequent.
- Reliability (Delta): Amount of error tolerance, used in combination with minSup used to determine frame size.

List of Tables

List of Figures

Abstract

The subject of this research is mining data stream. It is one of the most challenging and widely researched areas in Knowledge Discovery and Data Mining (KDD). A data stream is a continuous, voluminous, and unpredictable flow of data which occurs in many application domains. In a previous study (Kadam, 2009), Data Stream Mining (DSM) algorithm was proposed to overcome these problems on association rules mining. It was built using various techniques such as closed frequent itemsets, tree data structures, itemsets pruning, and statistical sampling. We have developed Near Closed Nodes algorithms, which can be applied to algorithms for mining association rules that utilised closed itemsets structure.

In this study, we look into the characteristics of closed frequent itemsets and propose a novel concept called Near Closed Nodes (NCN). This concept was thoroughly explored and later developed in conjunction with an existing DSM algorithm. By incorporating NCN into the DSM algorithm, we were able to increase the performance of both speed and memory usage. A comprehensive experimental study was performed to compare the performance of DSM and DSM-NCN using both simulated and real world datasets. Based on the results from the experimental study, we concluded that DSM-NCN outperformed DSM in most circumstances, especially when the datasets were dense.

Chapter 1 Introduction

Data stream mining arises with the ubiquity and affordability of computing as well as growth of the Internet. Knowledge Discovery and Data Mining (KDD) on data streams is essential in many application domains. These domains include but are not limited to computer networks, finance, and web usage.

KDD is the term given to the processes and technologies used to extract useful information and knowledge from data. Data mining can be classified into four main categories; classification, clustering, prediction and association rule mining. These data mining categories can be utilised with a wide variety of data including; Offline (static database), and Online (stream data). Offline mining is characterised by data that is processed in bulk form; data is static and all data can be processed at once. On the other hand, online data mining is characterised as mining on data that arrives continuously and is dynamic in nature. Online mining is essential for many applications where data is continuously arriving at high speed.

Data stream mining encompasses many mining techniques including Association Rules mining which is a technique that finds relationships between items within a given dataset. There are many applications that benefit from this type of data mining on a data stream including network security (Lee & Stolfo, 2000), and web usage (Bamshad, Honghua, Tao, & Miki, 2001). However traditional method such as the Apriori algorithm(Rakesh Agrawal, Imieliński, & Swami, 1993) alone is inadequate in a data stream mining environment due to a variety of technical issues (Jiang & Gruenwald, 2006b).

1.1 Research Motivation

Data streams pose unique and challenging characteristics that differentiate them from traditional databases. These characterise by behaviours including continuous arrival of data, unbounded length, high rate of arrival, and ever changing data distribution (Jiang & Gruenwald, 2006b). These characteristics pose great challenges to researchers to find algorithms that can extract patterns from such an unbounded stream of data.

High speed data streams pose problems to algorithms as the rate of processing needs to keep up with the speed with which data arrives in order to ensure that valuable data is not lost as new data arrive. Therefore it is crucial that algorithms utilise techniques such as incremental model building and statistical sampling to accelerate the mining process (Gaber, Zaslavsky, & Krishnaswamy, 2005).

Continuous arrival of unbounded data presents many problems to association rules mining algorithms as accumulative data can be limitless. This poses major challenges for researchers to design algorithms that are capable of exploiting compact storage methods and ways to efficiently access and update models that accurately capture patterns inherent in the data stream.

Together with the continuous arrival of data there is also the possibility of changes in the underlying stochastic data distribution. This phenomenon is referred to as concept drift (Haixun, Wei, Philip, & Jiawei, 2003), a term which is used in the data mining literature to describe changes in the underlying data distribution.

These three characteristics of data streams have created great opportunities and incentives for researchers to develop efficient algorithms that mitigate the effects from these issues. This research will focus on making improvement on mining efficiency to the existing Data stream mining implementation(Kadam, 2009).

1.2 Research Objectives

The broad objectives of the research are as follows.

- Investigate methods for reducing processing time for generating association rules in a data stream environment.
- Investigate methods for improving memory utilisation in the context of association rule mining in a data stream environment.
- Perform a comparative performance study with contemporary association rules data stream miner.

From these three broad objectives, we are aiming to develop improved methods which can be applied to an already developed implementation and to make comparison between the improved and original versions. We selected a developed implementation called Data Stream Mining or DSM (Kadam, 2009). This implementation was selected as it outperformed the existing state-of-the-art algorithm in association rule data stream mining.

1.3 Dissertation Structure

This chapter introduced the concept of data stream mining while discussing major challenges with generating association rules on data streams. These challenges arise from the continuous, unbound, and unpredictable nature of data streams. Chapter 2 will review

existing techniques and approaches to association rules mining on a data stream environment.

In chapter 3, we will present the underlying research methodology and discuss methods that we propose to overcome various issues imposed by the data stream environment.

Chapter 4 will present an experimental plan for a comparative performance study with an existing algorithm (DSM). It will outline various aspects of the experiments including performance metrics tested, chosen datasets, and the variables used in the experimentation.

Chapter 5 will present the experimental results and discuss the impact made by the methods that were introduced to improve the performance.

In the final chapter we will provide a discussion and summarise our research. It will conclude with directions for future work and further improvement.

Chapter 2 Literature Review

2.1 Introduction

The Apriori algorithm and its variants (R. Agrawal, Mannila, Srikant, Toivonen, & Verkamo, 1996; R. Agrawal & Srikant, 1994) were the catalyst for widespread research in the area of association rules mining. The Apriori algorithms are far superior to the brute-force approach at finding frequent itemsets (Tan, Steinbach, & Kumar, 2005). However, as stated in the previous chapter algorithms based on Apriori alone are not efficient and suitable to mine a data stream.

Association rules analysis from data stream environment poses various issues. These issues arise by the very nature of a data stream and were briefly discussed in the previous chapter. In this chapter we will discuss these issues in more depth and review past research that addressed them.

The past research that we are reviewed in this chapter can be classified into three distinct research domains. The three research domains according to Jiang and Gruenwald (Jiang & Gruenwald, 2006b) are as follows; data extraction, memory management, and mining approaches. This will now be discussed in turn.

2.2 Data Extraction Methods

Data streams pose key concerns over the ways in which data is processed. As streams are continuous, unbound, and unpredictable they require fundamental changes in the way which data is to be processed and extracted. Previous research has proposed three different processing models for data streams (Zhu & Shasha, 2002), which are Landmark, Damped and Sliding Window.

The Landmark model mines all frequent itemsets over the entire data stream from a specific data point called the landmark to the present point in time.

The Damped model, streams are weighted according to their age in the stream. Transactions in the stream have weights assigned to them with older transactions weighted less than more recent transactions.

The Sliding Window model finds and maintains frequent itemsets within specific sliding windows. Processing and storing of frequent itemsets occurs within sliding windows thus

enabling the mining to capture the most recent patterns. Figure 2.1 illustrates the differences in processing between the models.

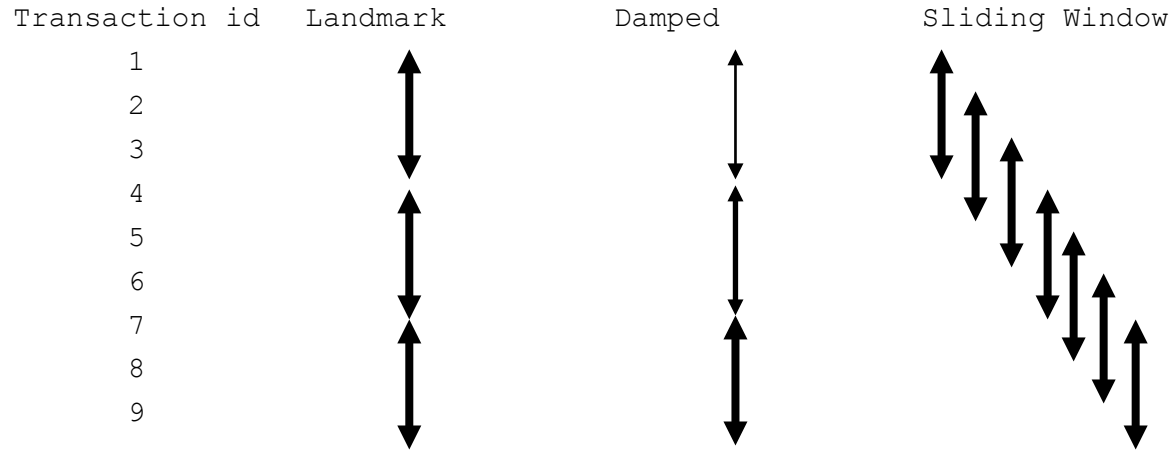


Figure 2.1: Transaction processing in the Landmark, Damped and Sliding windows models.

Choosing which models data stream mining depend on variety of reasons: application needs, expected input rate, expected outcome, and available resources. These mining models can be tailored to suit their needs and limitations.

2.3 Memory management and Data structures

The volume of arrival data from a data stream poses a major challenge as data streams are continuous and unbounded. Therefore it is crucial for algorithms designed to work with data streams to employ memory management techniques that store information effectively while enabling efficient retrieval and update of information.

There have been a number of attempts to reduce memory usage. These techniques are essential to effectively handle the large number of candidate sets that arise, unlike with finite static datasets. Techniques using data structures such as Frequent-pattern tree (FP tree) provide great improvement over Apriori methods in both speed and memory usage. This is achieved by optimized node sharing, reduced candidate generation, and reduction in the search cost by using divide-and-conquer-method (Han, Pei, Yin, & Mao, 2004).

To further improve mining efficiency many researchers have adopted estimation methods by maintaining estimated or reduced information. The Count-sketch data structure only keeps the most frequent itemsets (Charikar, Chen, & Farach-Colton, 2004) while discarding information about infrequent itemsets. The algorithm proposed by Yang keep support information on frequent short itemsets only (itemsets size < 3) thus keeping memory usage low (Yang & Sanver, 2004).

Mining association rules in a data stream environment can lead to generating and maintaining a vast number of itemsets. Thus researchers have resorted to more compact forms of representation such as Closed and Maximal itemsets (Tan et al., 2005), which require a smaller memory footprint to maintain.

Maximal frequent itemsets are frequent itemsets for which none of its immediate supersets are frequent. This representation significantly reduces the number of itemsets that need to be maintained since all frequent itemsets can be derived from the set of maximal itemsets. However, maximal itemsets have one significant drawback: while it is possible to derive all frequent itemsets, it is not possible to derive their support (Tan et al., 2005). Many researchers such as (Mao, Wu, Zhu, Chen, & Liu, 2007) and (Gouda & Zaki, 2001) have exploited the compact nature of maximal frequent itemsets in their research.

Alternatively, closed frequent itemsets also provide a compact and lossless representation of frequent itemsets. A closed frequent itemset is defined as a frequent itemset for which none of its immediate supersets have the same support. Unlike maximal itemset, closed itemset do not suffer from loss of support information. Researchers, including (Chi, Wang, Yu, & Muntz, 2004), (Zaki & Hsiao, 2002), (Pei, Han, & Mao, 2000) have adopted closed frequent itemsets properties in their mining algorithms.

Closed frequent itemsets offer a lossless compact representation of frequent itemsets, although they are not quite as compact as frequent maximal itemsets counterparts as Figure 2.2 illustrates. As such in this research we will exploit closed frequent itemsets due to its compact representation as well as its lossless property.

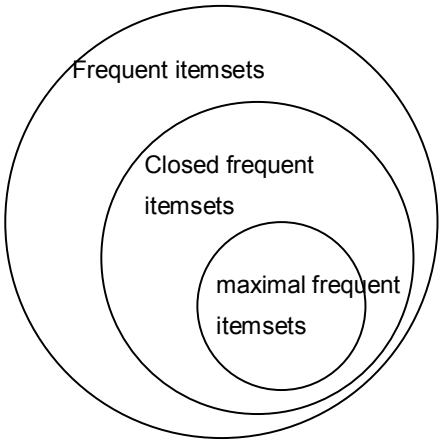


Figure 2.2: Relationship between frequent, closed frequent, and maximal frequent itemsets (Tan et al., 2005).

2.4 Mining approaches

Association rules mining research falls into either approximate or exact approaches (Gaber et al., 2005). The approximate approaches trade off the accuracy of the rules generated with mining speed, while exact approaches do not compromise accuracy.

The approximate approach generally provides association rules which contain either false-positives or false-negatives (Yu, Chong, Lu, & Zhou, 2004). Some approaches generate false-positive which mean the algorithms generate non frequent itemsets in the final result as well as frequent itemsets. On the other hand false-negative misses some frequent itemsets. Approximate approaches offer opportunities for researchers to develop a variety of novels techniques to increase mining speed and/or reduce memory usage.

2.5 Statistical sampling techniques

Statistical techniques, particularly sampling methods offer an attractive mechanism for reducing the volume of data to be processed in a data stream. However prior assumptions regarding the statistical distribution of data in a data stream are best avoided as each stream is unique in character and subject to change continuously over time. In circumstance data stream subjected to countless type of influences over a long period of time (Hulten, Spencer, & Domingos, 2001).

Common ways of dealing with an ever changing distribution of data in a data stream is to either discard old information or apply less weight to the older information (Kifer, Ben-David, & Gehrke, 2004). However these methods alone are not sufficient as they cannot predict the level of error from either discarding and/or discounting older information. Thus many researchers have chosen Hoeffding bound (Domingos & Hulten, 2001) or Chernoff bound(Chi et al., 2004; Zaki & Hsiao, 2002) to determine sampling size with a guarantee on the level of error.

2.6 Notable Research

There has been significant research in association rules mining in a data stream context. This research used various types and combination of techniques to improve memory utilisation and speed of association rules mining. Four of the most notable research works are reviewed in detail as they form the basis of the research discussed in this dissertation.

2.6.1 FPDM

Yu proposed the Frequent-Data-Stream-Pattern-Matching (FDPM) algorithm that can mine frequent itemsets from a data stream (Yu et al., 2004). FDPM is a false-negative

approximate one-pass algorithm. According to the literature FDPM has good accuracy and speed compared to previous approximate algorithms. FDPM allows control over accuracy using a combination of error and reliability parameters that operate on a version of probability theory known as Chernoff bound (Ravikumar & Lafferty, 2004) which dictates the partitioning of a data stream into segments. Yu proposed two versions of FPDM known as FPDM-1 and FPDM-2.

The FPDM-1 algorithm finds frequent items (1-itemsets), while FPDM-2 algorithm generates frequent itemsets of size $n \geq 2$. FPDM-2 uses two data structures to maintain frequent itemsets. The Chernoff bound is used to break up the data stream into a number of identically sized segments. Each segment represents that minimum number of data instances that are required to make statistically valid decisions. Structure P maintains the current batch of frequent itemsets within the current segment, while F stores potential frequent itemsets that have been discovered in the data stream up to the current point in time. When FPDM-2 processes the transactions it first updates the support information of itemsets that exist in P. Then at the end of the segment it determines frequent itemsets from P and transfers these to F. FPDM-2 controls its memory usage by pruning infrequent itemsets in F.

2.6.2 Moment

In contrast to FDPM, Moment is an exact approach that takes advantage of closed frequent itemsets (Chi et al., 2004). Moment uses the sliding windows scheme to incrementally maintain current closed frequent itemsets and the boundary. These itemsets are maintained in closed enumeration tree (CET) memory structure. CET stores four different categories of itemsets (nodes) including infrequent gateway nodes (infrequent itemsets which have frequent parents or parent's sibling nodes), unpromising gateway nodes (infrequent itemsets that have the same support as its superset), intermediate nodes (frequent itemsets having the same support as itself), and closed nodes (itemsets with no supersets that have greater or equal support). These nodes represent closed nodes and nodes that may change their status in the near future.

Whenever a transaction arrives, Moment traverses all itemsets composed from the transactions that belong in CET that is related to that transactions. It updates support information and, if necessary, classifies itemsets into one of four categories of closed or boundary nodes and adds them to the CET. It also deletes the oldest transactions from the window and updates the CET accordingly.

Moment uses less memory and CPU to mine closed nodes compared to the CHARM algorithms (Zaki & Hsiao, 2002), which is another exact associated rules mining algorithm using frequent closed itemsets. According to the research it performs an order of magnitude faster than CHARM. This is due to the incremental update model used by Moment. Their experimentation revealed that the majority of updates during incremental update result in a small proportion of CPU-intensive operations which correspond to category change or growth in CET.

2.6.3 CFI-stream Algorithm

CFI-stream was another algorithm proposed by Jiang. CFI-stream incrementally mines frequent closed itemsets in a data stream environment (Jiang & Gruenwald, 2006a). CFI-stream incrementally computes closed itemsets in one pass. It maintains only closed itemsets and their support information in a data structure referred to as Direct Update (DIU) tree. It generates closed nodes using a closure function to determine itemsets status.

Whenever a new transaction arrives, it performs an update only on associated closed itemesets which are determined by the closure function that it employs. CFI-stream use sliding windows mechanism; therefore the oldest transaction is dropped when new transaction arrives and a deletion operation is performed on DIU tree. Frequent closed itemsets can be generated per user's request by traversing the DIU tree.

In the research literature, CFI-stream outperformed Moment (Chi et al., 2004) in both CPU and memory usage at lower minimum support threshold, and dense datasets.

2.6.4 The Data Stream Mining Algorithm (DSM)

Data Stream Mining (DSM) is chosen to be the basis of this research for both techniques and actual implementation, a recently developed algorithm proposed by Omkar Kadam (Kadam, 2009). DSM uses an incremental approach to mine frequent itemsets using closed frequent itemsets. It uses a hash tree structure to maintain closed and related itemsets. It creates and uses a hash function to provide access to retrieve nodes and traverse a tree structure. The design details of this algorithm will be described in the next chapter.

Experimentation showed that DSM proved to have at least the same or better performance in term of accuracy than FPDM2 (Yu et al., 2004). In term of CPU and memory utilisation, DSM outperformed FPDM in all the experiments that were conducted.

2.7 Summary

In this chapter, we discussed various research opportunities to improve performance of association rules mining in data stream environment. This chapter concentrated on literature and algorithms that inspired this and previous research. In the next chapter we will choose appropriate research methodologies in order to design and implement an algorithm that will take advantage of these researches.

Chapter 3 Research Paradigm and Methods

3.1 Introduction

This chapter presents the methodology and framework that was used to conduct the research described in this dissertation. It offers an overview of various methods that were used in this research and explains the reasoning behind the chosen approaches.

In the field of KDD it is fundamentally important that the knowledge that results from the discovery process is useful while maintaining efficiency within the discovery process. The patterns generated must be validated for accuracy through a rigorous quantitative process. This suggests that the research be based on the positivist methodology (Popper, 1959).

3.2 Research Paradigms

In Information Systems research there are three fundamental research methodologies: Positivist, Interpretivist, and Critical. These different methodologies provide researchers with approaches and guidelines on which to base their research (Orlikowski & Baroudi, 1991).

The Positivist research methodology is based on the premise of discovering theory which can be investigated with structured and rigorous methods (Straub, Gefen, & Boudreau, 2004). The theory produced by Positivist research must provide formal propositions, quantifiable measures of variables and methods of hypothesis testing in order to make sound interferences about phenomenon (Orlikowski & Baroudi, 1991).

Orlikowski and Baroudi described Interpretivist research as a contrast to Positivist research as it based on the assumption that phenomena can be observed and understood through social constructs that participants interact with during the conduct of the research. In the Critical research methodology the goal is to critique the conventions or “status quos” in order to find novel ways of solving problems.

3.3 Research Framework

To successfully conduct Positivist research in order to achieve the specific objectives set out for this research we need to ensure that the methods used must be aligned to the chosen paradigm and provide the way in which ‘valid’ knowledge can be discovered.

Design science, unlike Natural science, attempts to create artefacts that are of practical use and provide improvement on the status quo (Hevner, March, Park, & Ram, 2004). The artefacts built can be evaluated using rigorous methods and thus the process aligns with the

Positivist methodology. Accordingly, we have decided to use the Design science framework for this research. Design science research, as proposed by Hevner et al consists of the two basic activities: build and evaluate which are used over a number of cycles to produce the desired artefacts, as presented in figure 3.1.

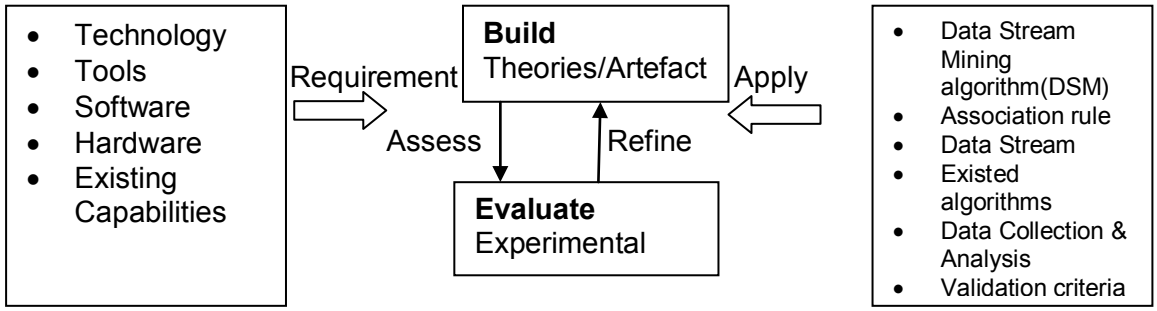


Figure 3.1: Research Framework for this dissertation, diagram adapted from (Hevner et al., 2004)

3.4 Research Methods

According to the Design science framework previously stated, the main activity consists of a series of Build and Evaluate cycles. These activities will play a major part in this research. Continuing in this chapter we will look into theory and implementation of DSM-NCN which were used in the build stage of our research, while chapter 4 will discuss further on evaluation activities (experimental plan).

Over the course of this research the build stage will play a critical role. This section presents the major building blocks that we used to build a new method of mining association rules over a data stream environment.

The starting point for the build phase is the existing DSM implementation (Kadam, 2009) that we have introduced in Chapter 2. The following sections present concepts and techniques that were used for in the current DSM implementation as well as novel methods that we have adopted to improve the current implementation.

3.4.1 Chernoff Bound

Association rules mining over data streams impose challenges on how transactions can be processed due mainly to the open-ended nature of streaming data. As a consequence streams must be processed differently to traditional static datasets. As opposed to mining on a traditional static dataset, one method that has been used to cope with open-ended streams of data is statistical sampling. In this approach the data stream is split into an infinite number of discrete partitions. Each partition represents the minimum number of transactions required

to make statistically reliable inferences about the average support of items taken across the entire data stream.

The Chernoff bound (Ravikumar & Lafferty, 2004; Yu et al., 2004) is used for this estimation. The bound specifies that the estimated average support of an item obtained via sampling is within an error margin (Є) for range of given variables (R), degree of reliability (δ) and (n) is the number of instances seen so far in the data stream.

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

Given known error margin we can determine the partition size n which produces result that statistically lies within the user specified error margin.

$$n = ½ (R/ Є)^2 \ln (2 / \delta)$$

3.4.2 Data Structure

In data stream mining the type of data structure that is used to store information on the frequent items is crucial in determining memory utilization (Jiang & Gruenwald, 2006b). Again, this is dictated by the open-ended nature of data streams.

Much research in the field of association rule mining in data streams has focused on variations of tree-like structures. Following are two examples of tree -like structures that have been proposed in various research papers. The Frequent Pattern tree (FP-tree) provides a compact and efficient data structure to maintain frequent itemsets (Han et al., 2004). The Itemset-tidset search tree reduces memory usage and computation that is required to stored closed itemsets and its lattices (Ching-Jui, 2005). There is general agreement in the association rule mining community that tree-like structures are optimum with respect to both storage and computational efficiency. The fact that DSM (Kadam, 2009) also features this structure was an added bonus given that this research seeks to extend the capabilities of DSM.

3.4.3 Frequent Closed Itemsets Mining

Frequent itemset mining is an essential part of association rule mining. However mining association rules based on frequent itemsets alone is inefficient as discovering a single frequent pattern size x alone requires the generation of at least 2x candidates (Han et al., 2004). When dealing with dense datasets or mining at very low support thresholds (for

example > 0.1%) can result in a large amount of frequent patterns. Such exponential growth has a direct performance impact on both storage and computational overhead involved in the mining process.

Closed frequent itemsets have been shown to provide a much smaller search space when compared to frequent itemsets, thus resulting in reduced memory usage, while still preserving the ability to generate frequent itemsets without any information loss unlike maximal frequent itemsets discussed in previous chapter (Pasquier, Bastide, Taouil, & Lakhal, 1999).

3.4.4 Itemsets Pruning

In order to mine frequent Itemsets (in this case frequent closed itemsets) from a data stream it is crucial to generate a powerset from the items that comprise in the transaction. The following example illustrates the generation of a transaction's powerset.

Transaction (X) = (A, B, D, E).
Powerset (X) = (A, B, D, E, AB, AD, AE, BD, BE, DE, ABD, ABE, ADE, BDE, ABDE)

From the example above, a given transaction of size 4 will result in a powerset of size 15. Thus powerset generation can substantially increase computation time and memory usage as it generates 2^n -1 elements, where n is number of items in the transaction.

Given the explosion in computation time to generate powersets, the DSM implementation has employed the frequent singleton mining technique to reduce the volume of powerset generation (Kadam, 2009). The DSM implementation took advantage of the Apriori property (R. Agrawal & Srikant, 1994) which states that any subset of a itemset must first be frequent in order for that itemset to become frequent. The DSM implementation took advantage of this property by only generating powersets from singletons which are known to be frequent in the data stream, thus greatly reducing the size of the powersets generated. The example below shows the powerset X generated with the same example transaction given that the current frequent singleton is A and E.

Transaction (X) = (A, B, D, E) Powerset (X) = (A, E, AE).

Thus it can be seen that the size of the powerset has reduced from 15 to 3 as a result of this strategy. The Frequent singletons strategy was implemented by using a list of frequent singletons that is initially compiled at the end of the first frame and is continuously maintained and updated at subsequent frames.

3.4.5 Frequent itemsets generation from Closed Frequent Itemsets

As stated before, DSM uses Closed frequent itemsets. The DSM implementation generates frequent itemsets from frequent closed itemsets by using the principle of closed nodes (Tan et al., 2005) and the following principle:

> *"The support value of a non-closed frequent itemset must be equal to the maximum support among its supersets" (Kadam, 2009).*

From these principles DSM determines frequent itemsets from the tree of frequent closed nodes and non-closed nodes. As mentioned earlier the support of value of a given frequent closed node must be equal to the highest support value amongst its supersets.

Therefore frequent itemsets are contained within frequent closed nodes and all of its subsets except in the case when the supersets of the frequent closed nodes themselves happen to be frequent closed nodes. Based on this simple rule DSM can identify frequent itemsets from the set of closed frequent itemsets that were previously generated.

3.4.6 Tree Pruning

The DSM implementation maintains closed itemsets in a tree structure. As mining progresses the tree data structure that we maintain may contain closed nodes that are no longer frequent. The DSM implementation minimises the impact of this issue by pruning the closed tree at regular intervals to ensure that the tree contains only closed itemsets that are frequent or have the potential to be frequent.

3.4.7 Near Closed Nodes

Near Closed Node (NCN) is a new concept that was developed in this research to improve the efficiency of the closed frequent itemsets generation processes of DSM implementation. NCNs can be generated using closed nodes, and as the name suggests NCN itemsets are subsets of closed nodes which are on the verge of becoming closed. Figure 3.2 is a simplified transaction tree used to provide example of NCNs and explanation to the principles that were used in this research.

Transaction ID	Items
1	ABC
2	ABCD
3	ACD
4	BC

Table 3.1: showing transaction ID and items in that transaction

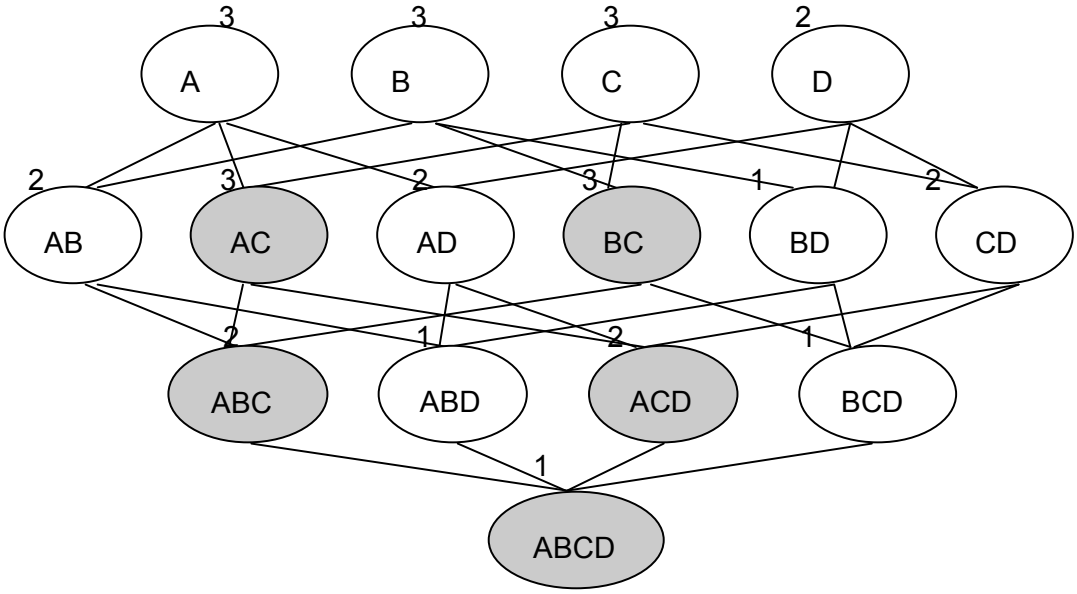


Figure 3.2: tree illustrates closed nodes (grey shading) and frequency for each itemset after the arrival of transactions 1, 2, and 3.

Observation 1: Nodes such as A, B and AB have the potential to become closed at the same time.

Observation 2: If A becomes closed it does not necessarily mean that AB or AD will become closed.

Observation 3: If transaction 5 is AB then, AB as well as its subsets (A and B) to become closed.

Observation 4: If transaction 5 is CD then, CD and C to become closed but not D.

Observation 5: If transaction 5 is AC then, it does not trigger any change in the closure status of A, C or AC.

From Observations 1 through 5 the following general principles can be derived.

Principle 1: Itemsets with closed node status never change to non-closed status, regardless of the arrival of any new transactions. This is due to the "landmark" model that we adopt in this research which captures the cumulative status of events that took place in the data stream from a given starting point in time. This is contrast to the sliding window model that tracks only the latest events. Principle 1 does not hold for the latter type of model.

Principle 2: Any non-closed subsets of closed nodes can become closed as long as an arrival transaction contains that subset but not the closed node itself.

Principle 3: If a subset of a closed node becomes closed (we will call it a 'new closed node') then any subset of the new closed node will become closed if frequency of that subset is greater than its supersets (ie. the new closed node).

Principle 4: (Stems from Principle 3) On the other hand if the frequency of any subsets of a new closed node is equal to the frequency of the new closed node itself, then no change in status results for the subset in question.

From Principles 1 through 4 we derive the following rules and example from previous scenario. These rules governing the generation of near closed nodes that we used extensively in our implementation.

Rule 1: Near closed nodes can be generated from non-closed subsets of a given closed node.

Example for Rule 1: near closed nodes from AC, BC, ABC, ACD, and ABCD are A, B, C, D, AB, AD, CD, ABD, and BCD (derived from Principle 4).

Rule 2: If a powerset of a new transaction contains all of the items within a closed node, then there will not be any changes in status of that closed node or any subset of that node. We simply update the frequency of all items in the transaction (derived from Principle 2).

Example for Rule 2: if a new transaction with items A and C arrives then no changes in result in the closure status of A, C or AC.

Rule 3: If the arrival transaction does not trigger Rule 2 then the following applies: If any powersets of arriving transaction contains a near closed node then that near closed node must become closed (derived from Principles 1, 2, and 4).

Example for Rule 3: if a new transaction with items A and B arrives then AB will become closed as it is part of the near-closed node list generated from Rule 1.

Rule 3A: If the requirement of Rule 3 was met then the following rule applies. If a subset of a near closed node has a greater frequency then that subset will become closed.

Example for Rule 3A, If AB become closed from new transaction then A and B will become closed since A and B has higher frequency than AB.

Rule 3B: If the requirements of rule 3 were met then the following rule applies. If a subset of a near closed node has a frequency equal to that of the near closed node then no change results to that subset.

Example for Rule 3B, If CD becomes closed due to a new transaction then C will also become closed according to Rule 3A but no change in status results with D as the frequency of CD is equal to that of C.

The above three main rules and two sub-rules can be used to generate near closed nodes which in turn may result in identifying new closed nodes. These rules can be applied to the original DSM implementation to reduce the processing time to generate new closed nodes. Given below is the pseudo code for near closed node generation.

Generate near closed nodes; genNearClosed()

Description

After each pass genNearClosed() will traverse recently generated closed nodes. For each newly created closed node it will perform a traverse bottom-up traversal through the lattice of nodes. It selects supersets of closed nodes with the highest support which are not currently closed and flags those selected as near closed nodes.

```
1. For all new Closed nodes (recently generated from previous
   pass) repeat
      a. While a closed node has more parent (subset) nodes
           i. While parent node has more subsets
                1. If a node has greater support than previous
                   nodes, not currently closed, then store that
                   node in maxNode
          ii. End While
         iii. Commit maxNode into hash table consisting of NCNs
      b. End While
2. End For
```

3.5 Overall Implementation

The previous section explains concepts and techniques used to implement NCN to the original DSM implementation. In this section we will discuss overall design through brief description of key procedures with simplified pseudo code explaining the DSM-NCN implementation. This implementation is a modified version of the original DSM implementation incorporating the near closed concept as a method of reducing time for generating closed nodes.

DSM-NCN, like its DSM counterpart, contains three main procedures; firstPass(), secondPass(), and mineData(). The first two procedures run at the beginning stage of data stream collects necessary information for mineData() to routinely generate closed nodes. The modules, firstPass() and secondPass() on both implementations employ identical code while changes in the mineData() module were used to incorporate the novel near closed nodes concept.

Global Parameters

minSup; is user configurable minimum support threshold.

frameSize; is transactions per frame determined by Chernoff bound.

LA; is list of frequent singletons found across all previous pass.

LC; is list of frequent singletons found in current pass.

LT; is list of frequent singletons of interest in current transaction.

H is hash table contains identify of frequent itemsets.

frequentSingletons; is list of frequent singletons (1 itemsets) found in this frame.

First Pass; firstPass()

Description: The purpose of this pass is to generate frequent singletons which are required for subsequent passes to generate pruned powersets. This pass generates frequent singletons by reading transaction and tokenised into 1-itemsets (singletons) which are constantly being updated as more transaction are read. At the end of this pass a list of frequent singletons is populated using minimum support threshold.

1. For index=0 to frameSize
    a. Read the transaction from data file
    b. Tokenised the transaction to obtain singleton items
    c. For all token repeat
        i. If the token have already seen update its support information
        ii. Else, create a node entry in hash table H
    d. End for
2. End for
3. Populate the list of frequentSingletons(LA) with minSup minimum support threshold.

Second Pass; secondPass()

Description: The purpose of this pass is to generate frequent duo-itemsets to facilitate subsequent mineData() passes. secondPass() incorporated steps from 1-a to 1-e to generate frequent singletons mining similar to firstPass(). Intersection operating between previously frequent singleton and arrival singleton to ensure the singleton was frequent. Once overall frequent singletons were identified, then powersets can be generated result in frequent duo-itemsets.

```
LC = null
   1. For index=0 to frameSize
         a. Read the transaction from the data file
         b. LT = null
         c. Tokenised the transaction to obtain singleton items
         d. For all tokens repeats
              i. If the token have already seen update its support
                 information
             ii. Else, create a node entry in hash table H for new
                 singletons
         e. End for
         f. LT = LT ∩ LA
         g. Generate powerset of LT
         h. For every elements of 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 new
                 element
         i. LC = LC ∪ LT
         j. End for
   2. End for
   3. LA = LA ∪ LC
```

Mine Data; mineData()

Description: The purpose of this module is to generate frequent closed itemsets and is called from the third pass onwards. The initial steps from 1-a-i to 1-a-vi mirror those taken in secondPass() that generated 2-frequent itemsets (ie. Itemsets of size 2). Then, after the infrequent 2-itemsets have been eliminated this list is used to generate higher level itemsets (i.e. n-itemsets, with n>2). In general, the generation of a frequent n- itemset requires the identification of two frequent (n-1)-itemsets which have common prefixes.

Several checks required after frequent itemsets have been generated. First, check whether frequent itemsets already existed in the “near closed node” list, if it does then the itemset will be transferred to the “createNode” and “removeNode” list, otherwise update itemset’s support and continue on with the next itemset. If neither condition is met then checked itemset for closed node eligibility which then adds to “createNode” list and itemsets’s support can be calculated.

At the end of each transaction the “createNode” list is incorporated into the closed node tree while the nodes in the “removeNode” is removed from the near closed node tree data structure.

At the end of each frame the list of singletons, the closed nodes tree as well the near closed node tree are pruned and refreshed. These processes remove redundant nodes and ensure faster runs for future passes.

As mentioned earlier the differences in implementation between DSM and DSM-NCN exist manifest mainly in the mineData() module. These differences are extensions that perform near closed node generation and update these are highlighted in grey colour.

1. Repeat while data file is not empty
    a. For index=0 to frameSize
        i. Read the transaction from the data file
        ii. Tokenised the transaction to obtain singleton items
        iii. For all tokens repeats
            1. If the token have already seen update its support information
            2. Else, create a node entry in hash table H for new singletons
        iv. End for
        v. LT = LT ∩ LA
        vi. Generate powerset of LT
        vii. For every elements of powerset, repeat
            1. If the element was mark as frequent in previous pass then add to temporary list T
            2. Else, continue to next element
    b. End for
    c. For all element in temporary list T, repeat
        i. Retain only those elements that have common prefixes
        ii. Add them to list "powersetElements"
    d. End for
    e. For all elements in powersetElements list, repeat
        i. If the element exists in the near closed node tree, then it is removed from the "createNode" list and added to "removeNode" list
        ii. Else If element exists in closed node tree, then retrieve the node and update its support information
        iii. Else if this node qualifies to be added to createNode list
            1. Check if there exists superset 'e' of current node 'f'. If so support (f) = support(e)+1
    f. End for
    g. Commit createNode list into frequent closed tree.
    h. Remove node from removeNode list in the near closed tree.
    i. End of frame
2. LA = LA ∪ LC
3. Prune closed node tree removing children of infrequent nodes, but keep infrequent nodes which may change in future.
4. Traverse closed node tree, generating near closed nodes from newly create frequent closed nodes.
5. End of data file
6. Traverse closed node tree and determine closed frequent itemsets (node that meet minSup threshold).
7. Generate frequent itemsets from closed frequent itemsets.

3.6 Summary

In this chapter, we discussed and outlined research paradigms and methods used to achieve our objectives, along with some of the principles governing our new approach of using near closed nodes. The next chapter will provide a discussion on evaluation processes that we utilized in this research.

Chapter 4 Experimental Design

4.1 Introduction

In this chapter we will be describing the experiments designed to evaluate various performance indicators of our implementation that was intended to be an improvement over the original proposed Data Stream Mining (DSM) algorithm. The following sections will further explain how the experiment will be carried out and the reasoning behind the chosen methods.

4.2 Datasets

The experimentations are carried out using both synthetic data and real world data. Synthetic data was generated using the Data synthesizer component of Java implementation from ARtool (Cristofor, 2006). Real world data was from Frequent Itemset Mining Implementations Repository (Goethals & Zaki, 2003).

Synthetic datasets provided advantage over real world data as key characteristics of the datasets can be altered. These characteristics include number of transactions, average size of each transaction, number of unique items, and number of items that appear to be frequent in dataset.

The difference in these characteristic will allow experiments to test for sensitivity of the algorithms and allow us to provide an explanation for the difference in performance.

On the other hand using real world data on the experiment is by itself not effective to provide an explanation for the result performance due to limited amount of known datasets characteristics. However real world data could provide confirmation about result validity on real world application better than any synthetic data(Zheng, Kohavi, & Mason, 2001).

4.2.1 Synthetic Datasets

The ARtool package is a collection of tools and algorithms for mining of association rules. It enables users to generate datasets and perform association rules analysis. The synthetic data generator within the ARtool is used to generate synthetic data with configurable characteristics. Following are seven characteristics that can be altered using synthetic data generator from ARtool.

1. Number of transaction (num_transactions); define number of transactions to generate for a given dataset.

2. Average transaction size (avg_transaction_size); defines average size of each transaction.

3. Number of large itemsets (num_large_itemsets); defines number of large itemsets in the generation of transactions.

4. Average size of large itemsets (avg_large_itemset_size); defines the average size of large itemsets.

5. Number of item in transaction (num_items); defines number of unique items to appear in the transactional database.

6. Correlation mean (correlation_mean); defines the amount of mean correlation between the large itemsets

7. Corruption mean (corruption_mean); define mean of the corruption coefficient which indicates how much a large itemset will be corrupted before being used. This parameter simulates noise that occurs in a real-world dataset. It introduces an element of non-determinism in an artificial dataset.

Given these configurable characteristics the following four parameters will be independently altered for the experimentation. Table 4.1 shows the parameter values with a set of fixed characteristics include; number of transactions (100000 transactions), correlation mean (0.5), and corruption mean (0.5).

Datasets name	Average size of large itemsets (Default 4)	Average transaction size (Default 6)	Number of item in transaction (Default 1000)	Number of large itemsets (Default 300)
D1A	3	Default	Default	Default
D1B	4			
D1C	5			
D1D	6			
D2A	Default	6		
D2B		8		
D2C		10		
D3A		Default	1000	
D3B			2000	
D3C			10000	
D4A			Default	200
D4B				300
D4C				400
D4D				500

Table 4.1: Configurations used to synthesize datasets.

4.2.2 Real World Datasets

For this research two real-world datasets was chosen. These datasets was available from the Frequent Itemset Mining Dataset Reposity (Goethals & Zaki, 2003). Most of the characteristics of these datasets are unknown. Table 4.2 exhibits known information and characteristics of these datasets.

Datasets name	Description	Number of transactions
kosarak_click_stream.txt	annonymized click-stream data of Hungarian online news portal provided by Ferenc Bodon (REF)	990,002
retail.txt	annonymized retail market basket data from Belgian retail store donated by Tom Brijis (REF)	88,162

Table 4.2: Real world datasets and their characteristics

4.2.3 Datasets from previous work

These datasets were generated using the IBM generator previously used with DSM (Kadam, 2009) and FPDM2 (Yu et al., 2004) implementations to provide accuracy performance comparison.

Datasets name	Average Transaction Size	Average Frequent Itemset Size	Unique Items	Number of Transactions
T10I4D100K	10	4	10,000	100,000
T5I4D500K	5	4	10,000	500,000

Table 4.3: Previous work dataset and their characteristics

4.3 Performance Metrics

The original DSM implementation and modified implementation are evaluated against major performance matrices including Accuracy (recall and precision) and Resource usage (in term of processing power and memory consumption). Table 4.4 defines the performance metrics captured utilised by the experimentation.

Metrics	Description	Definition
Recall	The proportion of generated frequent itemsets against the actual number of frequent itemsets (FIs)**	$\frac{\text{Actual frequented itemsets in data} \cap \text{Generated frequent itemsets}}{\text{Actual itemsets in data}}$
Precision	The accuracy proportion of generated frequent itemsets (FIs)	$\frac{\text{Actual frequented itemsets in data} \cap \text{Generated frequent itemsets}}{\text{Generated frequent itemsets}}$
Processing time	amount of time in milliseconds (msec) to perform an analysis	Elapsed time between time stamp at the end of last frame and the beginning of first frame in millisecond
Memory Consumption	amount of memory used to perform an analysis	Average number of closed nodes and near closed nodes (for DSM-NCN implementation) at the end of each pass

Table 4.4: Description of performance metrics used and measurement definition.

**Actual number of frequent itemsets can be calculate using Apriori based frequent itemsets mining software developed by Christian Borgelt (Borgelt & Kruse, 2004) and further explanation will be provided in the next chapter.

4.4 Hypotheses

The following three hypotheses will be investigated during experimental stage of this research.

Ho1: The use of near closed itemsets accelerates the mining process over original DSM implementation.

Ho2: The use of near closed itemsets increase memory utilisation. In the other word it will reduce the number of nodes created by the algorithms.

Ho3: The use of near closed itemsets does not affect performance accuracy over original DSM implementation.

4.5 Experiment Plan and Execution

The new NCN algorithms were implemented over original DSM implementation which had been implemented in Java programming language. We re-use DSM code to reduce implementation time while keeping results of the previous work relevant. We also ensure that results of the experiments come from the additional of NCN algorithms and not from how they been implemented.

We design our experimentation into three separate parts which are to be performing on three different PCs with similar software and hardware outfit. Following are experiments that were design to measure performance matrices of the implementation; Accuracy, and Resource usage. These performance matrices will then we use for analysis and provide conclusion to given hypotheses.

4.5.1 Experiment 1: Accuracy performance comparison

Near Closed Nodes implementation (DSM-NCN) was designed and implemented to generate the same set of association rules as the original DSM implementation when mining on identical set of data and settings. Therefore there is no need to run multiple experiments on performance accuracy as long as we can run on few datasets to ensure validity of our accuracy performance hypothesis. In these experiments we are using two datasets that were used from previous research and one real world dataset. These experiments will enable us to verify that DSM-NCN implementation does not reduce performance of the original implementation to generate association rules at a variety of minimum support and reliability settings (Delta). The following are steps taken for these experiments

1. Execute T10I4D100K with both DSM and DSM-NCN implementations
2. Run datasets with variable support threshold and a fixed reliability of 0.1.
3. Run datasets with variable reliability and fixed support of 0.05
4. Repeat step 1) to 3) on T5I4D500K and retail datasets
5. Collect frequent itemsets generated on both implementations for all the experiments.

This experiment was designed to measure only frequent itemsets produce by the algorithms, therefore specification of this experiment run of this PC is irrelevant. However entire experiment 1 will run on one machine.

4.5.2 Experiment 2: Resource performances Comparison

NCN was design and implement to reduce processing times for each pass by predicting and storing future closed nodes thus potentially increase memory usage.

Average processing time was calculated using total running time from multiple identical runs (5 runs). Total running time was calculated using time different between firstPass() started and last transaction was completed by the algorithms.

Memory usage was calculated and provide as relative estimation using average closed node. Average closed node is the average number of closed nodes at the end of each pass (for DSM-NCN implementation average near closed nodes is also includes as part of memory usage).

Following are the two main experiments that will allow us to compare on both processing time and memory consumption on synthetic and real world datasets.

4.5.3 Experiment 2A: Experiments on synthetic data

Experiments on synthetic datasets were design to measure the difference of resource performance matrices between two implementations. Synthesising data allow us to modify various characteristic of the datasets. This enables us to test and identify datasets characteristics that may have effects on performance of the implementations. There are 14 synthetic datasets use for these experiments shown in table 4.1. Following are steps taken for these experiments

1. Execute synthetic data with both DSM and DSM-NCN implementations using fixed support and reliability.
2. Repeat step 1) until the datasets have been executed for five runs.
3. Repeat step 1) and 2) until all synthetic datasets have been executed.
4. Collect all data on per pass processed time, per pass nodes, running time, and number of nodes created.

All the synthetic experiments were run on PC with 1.86 GHz Core 2 Duo CPU, 2 GB of memory. A minimum support threshold of 0.05 with Delta value of 0.1 was used in the experimentation

4.5.4 Experiment 2B; Experiments on real world data

Real world datasets experiments were designed to measure and compare the effect of various minimum support threshold values and reliability on performance of two implementations while confirming the validity of the implementation with real world situations. Following are the steps taken for experiments which measure the effect of minimum support threshold on two implementations.

1. Execute kosarak_click_stream and retail datasets with both DSM and DSM-NCN implementations.
2. Run both datasets with variable support threshold of 0.1, 0.05, and 0.01 with a fixed reliability of 0.1.
3. Repeat step 1) and 2) until both datasets has been executed for five times.
4. Collect all data on per pass processed time, per pass nodes, running time, and number of nodes created.

On the other hand the reliability value quantifies the extent to which the support estimate produced by the Chernoff bound is in error. The reliability values used were 90%, 95%, and 99%. Following are steps taken for the experiments which measure the effect of reliability parameter on the two implementations that we compare.

1. Execute kosarak_click_stream and retail datasets with both DSM and DSM-NCN implementations.
2. Run both datasets with variable reliability of 0.1, 0.05, and 0.01 with a fixed support of 0.05
3. Repeat step 1) and 2) until both datasets has been executed for five times
4. Collect all data on per pass processed time, per pass nodes, running time, and number of nodes created.

All experiments on datasets kosarak_click_stream.txt, and retail.txt were conducted on two identical PCs with identical specification of 2.13 GHz Core 2 Duo and 2 GB of memory to speed up the process. Each dataset was experimented on the same machine to ensure that results are comparable.

4.6 Summary

This chapter described in detailed the evaluation processes to be used to compare DSM with the DSM-NCN implementation. The evaluation of both implementations will incorporate both synthetic and real world dataset. The four metrics we presented were used as the basis to measure both accuracy and efficiency performance of the artefacts designed and implemented in this study.

Chapter 5 Research Finding

5.1 Introduction

The previous chapter presented the experimental design and the experimental configurations that will be used in the experimental study to be presented in this chapter. This chapter will describe findings, offer explanations and provide analysis for each finding.

Both the original DSM and DSM-NCN implementations were thoroughly tested on wide range of synthetics datasets and real world datasets in order to compare accuracy and resource consumption.

5.2 Accuracy performance comparison (Experiment 1)

These experiments were designed to test and compare accuracy across the DSM and DSM-NCN implementations on selected datasets.

We recorded frequent itemsets generated from both implementations to make a comparison. The C implementation of the Apriori based algorithm was written by Christian Borgelt (Borgelt & Kruse, 2004) is available from http://borgelt.net/apriori was used to benchmark accuracy. We note that the Apriori algorithm cannot be used in an actual data stream environment, and so for our experimental testbed the data streams were simulated by static transactional datasets.

Tables 5.1 to 5.3, exhibit the accuracy performance on each dataset. Accuracy is measured in terms of Recall and precision. In addition, number of frequent itemsets (FIs) generated were collected for comparison.

For the real world dataset retail.txt is used, which contains anonymous market basket data from a Belgian retail store (Goethals & Zaki, 2003), Table 5.1 summarises the findings.

Experiment	Support	Delta	Original DSM			DSM-NCN		
			recall	precision	no of FIs	recall	precision	no of FIs
Varied Support/ Fixed Delta	0.1	0.1	100%	100%	13	100%	100%	13
	0.05	0.1	100%	100%	17	100%	100%	17
Varied Delta/Fixed Support	0.05	0.1	100%	100%	17	100%	100%	17
	0.05	0.05	100%	100%	17	100%	100%	17
	0.05	0.01	100%	100%	17	100%	100%	17

Table 5.1: number of FIs generated and performance accuracy on retail datasets.

Mining on the retail dataset, the DSM and DSM-NCN implementations, when run on the same minimum support and reliability settings, both generated the same set of frequent itemsets. Regarding accuracy performance, both generated frequent itemsets with100% Recall and Precision on all support and reliability settings when compared to actual frequent itemsets from the Apriori implementation.

For the synthetic datasets T5I4D500K.txt, Figure 5.2 summarises the findings.

Experiment	Support	Delta	Original DSM			DSM-NCN		
			recall	precision	no of FIs	recall	precision	no of FIs
Varied Support/ Fixed Delta	0.05	0.1	100%	100%	0	100%	100%	0
	0.01	0.1	100%	100%	0	100%	100%	0
	0.005	0.1	100%	100%	95	100%	100%	95
Varied Delta /Fixed Support	0.005	0.1	100%	100%	95	100%	100%	95
	0.005	0.05	100%	100%	95	100%	100%	95
	0.005	0.01	100%	100%	98	100%	100%	98

Table 5.2: accuracy performance and number of FIs on T514D500K datasets.

On the T514D500K dataset, the minimum support settings were lower compared to other experiments due to the sparse nature of the dataset (there are no frequent itemsets with minimum support greater than 1%). Both implementations, when executed with the same setting, produced the same set of frequent itemsets, thus they performed equally well in

regard to generating frequent itemsets. In regard to accuracy performance both implementations scored 100% on Recall and Precision on settings when compared to actual frequent itemsets from the Apriori implementation.

For the synthetic dataset T10I4D100K.txt, Figure 5.3 summarises the findings.

Experiment	Support	Delta	Original DSM			DSM-NCN		
			Recall	precision	no of FIs	recall	precision	no of FIs
Varied Support/ Fixed Delta	0.1	0.1	100%	100%	0	100%	100%	0
	0.05	0.1	100%	100%	24	100%	100%	24
Varied Delta/Fixed Support	0.05	0.1	100%	100%	24	100%	100%	24
	0.05	0.05	100%	100%	24	100%	100%	24
	0.05	0.01	100%	100%	24	100%	100%	24

Table 5.3: Number of FIs generated and performance accuracy on T10I4D100K.

As with the previous two datasets, the original DSM and DSM-NCN implementations, when run on the same settings, they generated exactly the same set of frequent itemsets. Thus in regard to accuracy on T10I4D100K both implementations have identical performance. Based on actual frequent itemsets from the Apriori implementation, both DSM and DSM-NCN generated 100% Recall and Precision rate.

From all three experiments the results produced were identical, thus reinforcing our belief that DSM-NCN does not sacrifice accuracy compared to DSM.

5.3 Resource usage performance comparison (Experiment 2)

These experiments were designed to evaluate resource usage performance of both memory usage and processing time on DSM and DSM-NCN implementations. The experiments were once again conducted on both synthetic and real world datasets. Two experiments were designed to reveal the sensitivity of resource performance on a variety of key parameters that govern the association rules mining process. These performances metrics were discussed in chapter 4 in term of processing time (average processing time) and memory consumption (average nodes created)

5.3.1 Finding from experiments with synthetic data (Experiment 2A)

Experiments on synthetic datasets were carried out to test the effect of varied characteristics of the datasets. These experiments were carried out to provide performance comparison between the two implementations across four dataset's characteristics. Table 5.4 summarises performance on time and memory for the synthetic datasets. These four characteristics were average large itemset size, average transaction size, the number of items, and the number of large itemsets. For full detail of datasets characteristics please refer to table 4.1.

Experiment		Average Total Time in milliseconds		% diff	Average nodes Created		% diff
		DSM	DSM-NCN		DSM	DSM-NCN	
Average large itemset size D1A-D1D	3	302,888	237,732	21.5%	2,125	1,651	22.3%
	4	289,378	236,657	18.2%	2,023	1,709	15.5%
	5	269,118	192,707	28.4%	2,261	1,812	19.9%
	6	192,429	151,765	21.1%	1,808	1,508	16.6%
Average transaction Size D2A- D2C	6	85,252	70,395	17.4%	1,315	1,157	12.0%
	8	1,423,733	1,011,309	29.0%	4,323	3,168	26.7%
	10	4,062,134	3,120,808	23.2%	6,210	4,752	23.5%
Number of items D3A-D3C	1000	266,885	211,606	20.7%	1,944	1,612	17.1%
	2000	111,987	89,735	19.9%	1,290	1,099	14.8%
	1000	19,622	17,841	9.1%	544	520	4.4%
Number of large itemsets D4A-D4D	200	2,169,093	1,612,910	25.6%	4,477	3,336	25.5%
	300	362,707	284,524	21.6%	2,343	1,982	15.4%
	400	42,120	36,200	14.0%	855	709	17.1%
	500	21,920	23,644	-7.9%*	675	599	11.3%

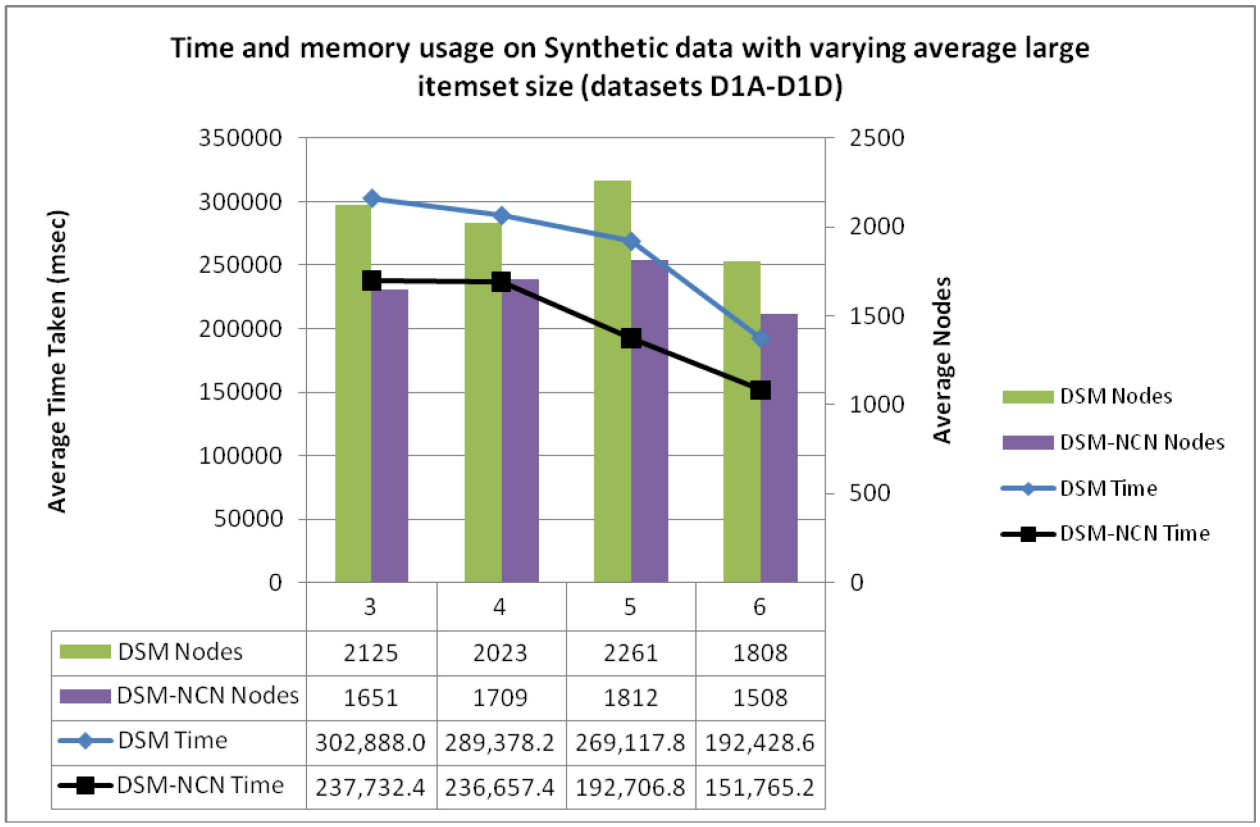
Table 5.4: Average time and average number of nodes created on the synthetic datasets.

Table 5.4 shows that DSM-NCN implementation performs better overall in terms of processing time (only one test takes a longer time to run*) and uses less memory to maintain closed nodes in comparison to DSM.

In the next section, we will provide analysis on the performance differences between DSM and DSM-NCN implementations as well as offer graphical comparison.

5.3.2 Result analysis of Synthetic data experiments

We next investigated the effect of varying the average large itemset size characteristic. Figure 5.1 exhibits a chart of average time taken and memory usage across average large transaction size of 3, 4, 5, and 6. For this and subsequent experiments line graphs will be used to represent time taken (on the primary y-axis) and bar graphs to represent nodes created (on the secondary y-axis).

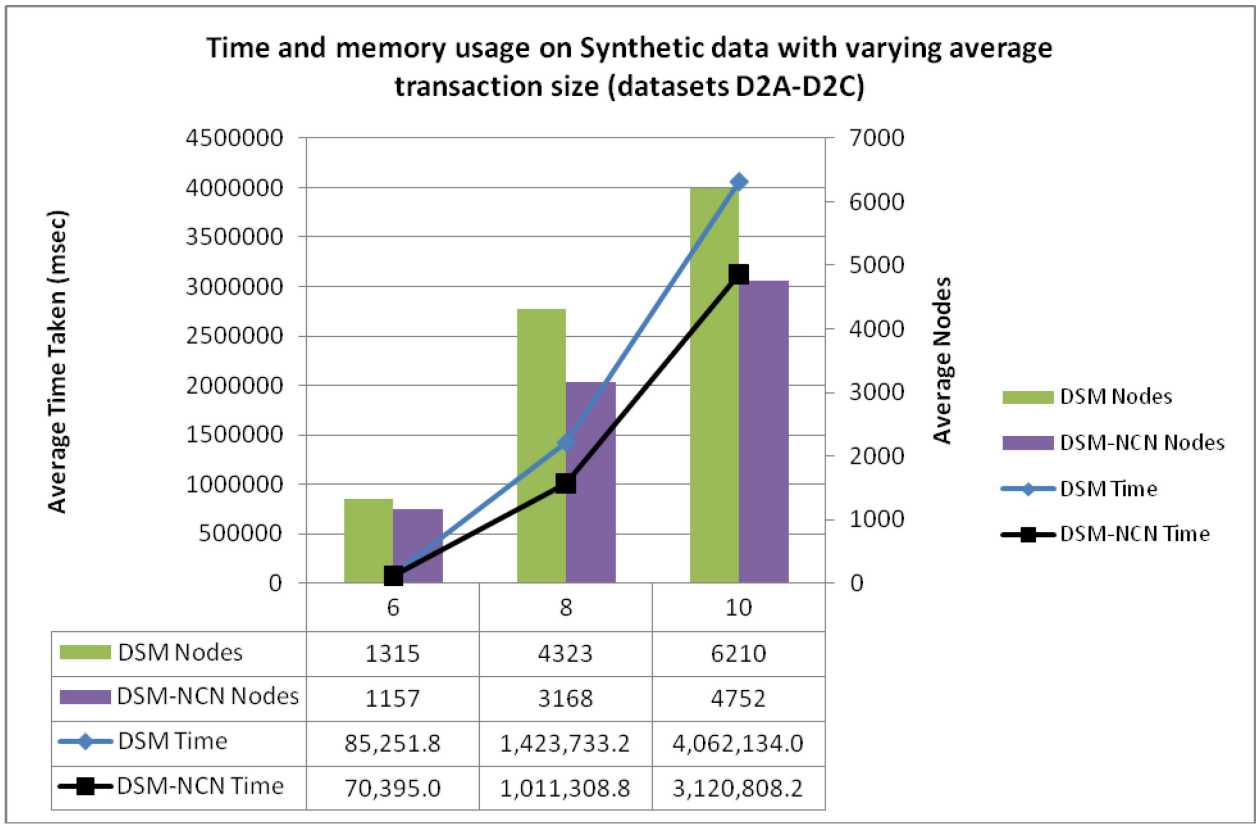


	3	4	5	6
DSM Nodes	2125	2023	2261	1808
DSM-NCN Nodes	1651	1709	1812	1508
DSM Time	302,888.0	289,378.2	269,117.8	192,428.6
DSM-NCN Time	237,732.4	236,657.4	192,706.8	151,765.2

Figure 5.1: Graph comparing time and memory usage on varying average large itemset size (datasets D1A-D1D).

From Figure 5.1 we can see that as size of large itemset increases both time and memory usage improves. This is due to the fact that larger frequent itemsets will have less likelihood to appear in a transaction as opposed to smaller frequent itemsets. DSM-NCN implementation performs consistently better across the range of the average large itemsets size parameter in term of time taken and memory use to maintain frequent closed nodes.

The next experiment was on assessing the impact of average size of a transaction. Figure 5.2 exhibits a chart of time taken and memory usage needed to maintain frequent closed nodes for three datasets. These datasets were generated using the same settings (refer to table 4.1) except for the difference in average transaction size which varied in the range: (6, 8, and 10).

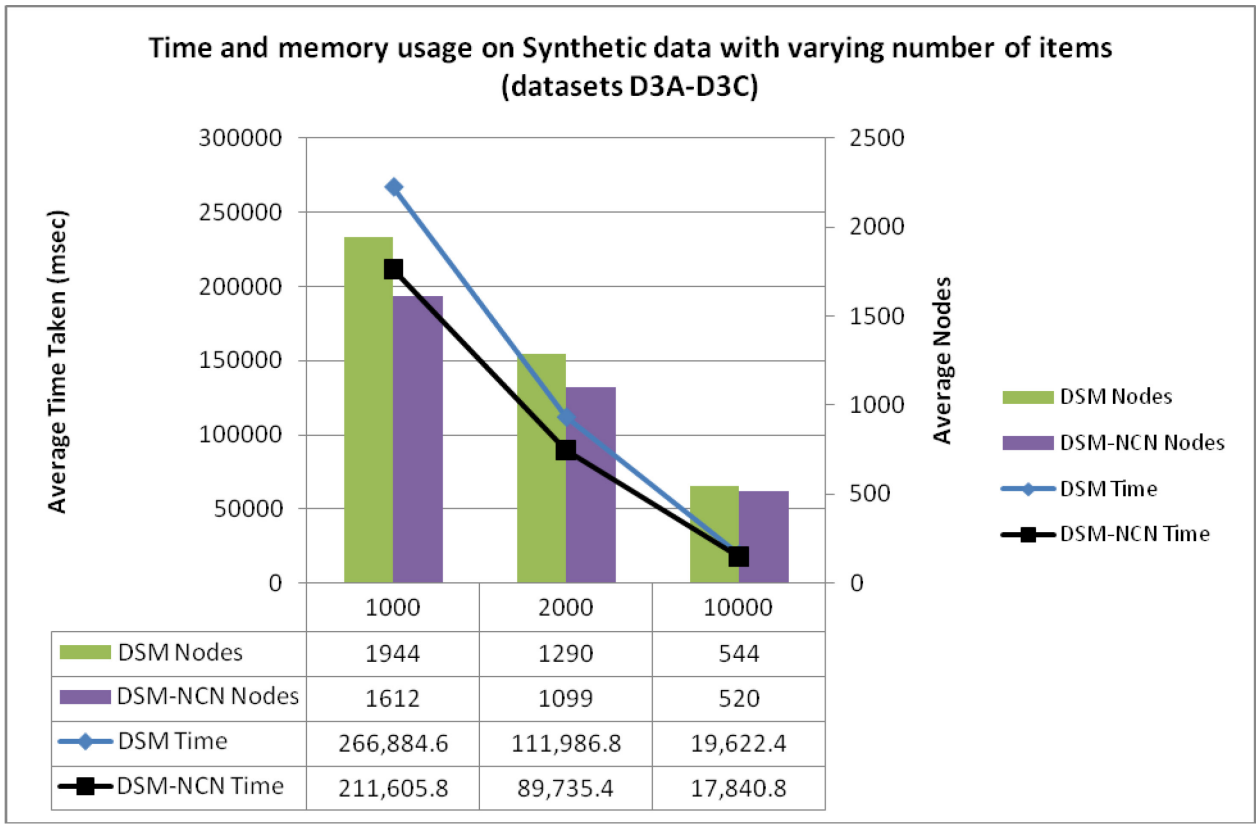


	6	8	10
DSM Nodes	1315	4323	6210
DSM-NCN Nodes	1157	3168	4752
DSM Time	85,251.8	1,423,733.2	4,062,134.0
DSM-NCN Time	70,395.0	1,011,308.8	3,120,808.2

Figure 5.2: Graph comparing time and memory usage on varying average transaction size (datasets D2A- D2C).

Figure 5.2 above shows that the variation in average transaction size has a significant effect on time and memory usage for both implementations. DSM-NCN performs better overall, while the level of improvement increases with an increase in the average transaction size (widening gap in the line graph). This suggests that DSM-NCN is more efficient at processing larger transactions.

The next experiment was on varying the number of unique items. Figure 5.3 displays the time taken and memory usage to maintain closed nodes for three datasets. These datasets were generated using the same setting (refer to table 4.1) except that they have different number of unique items (1000, 2000, and 10000).

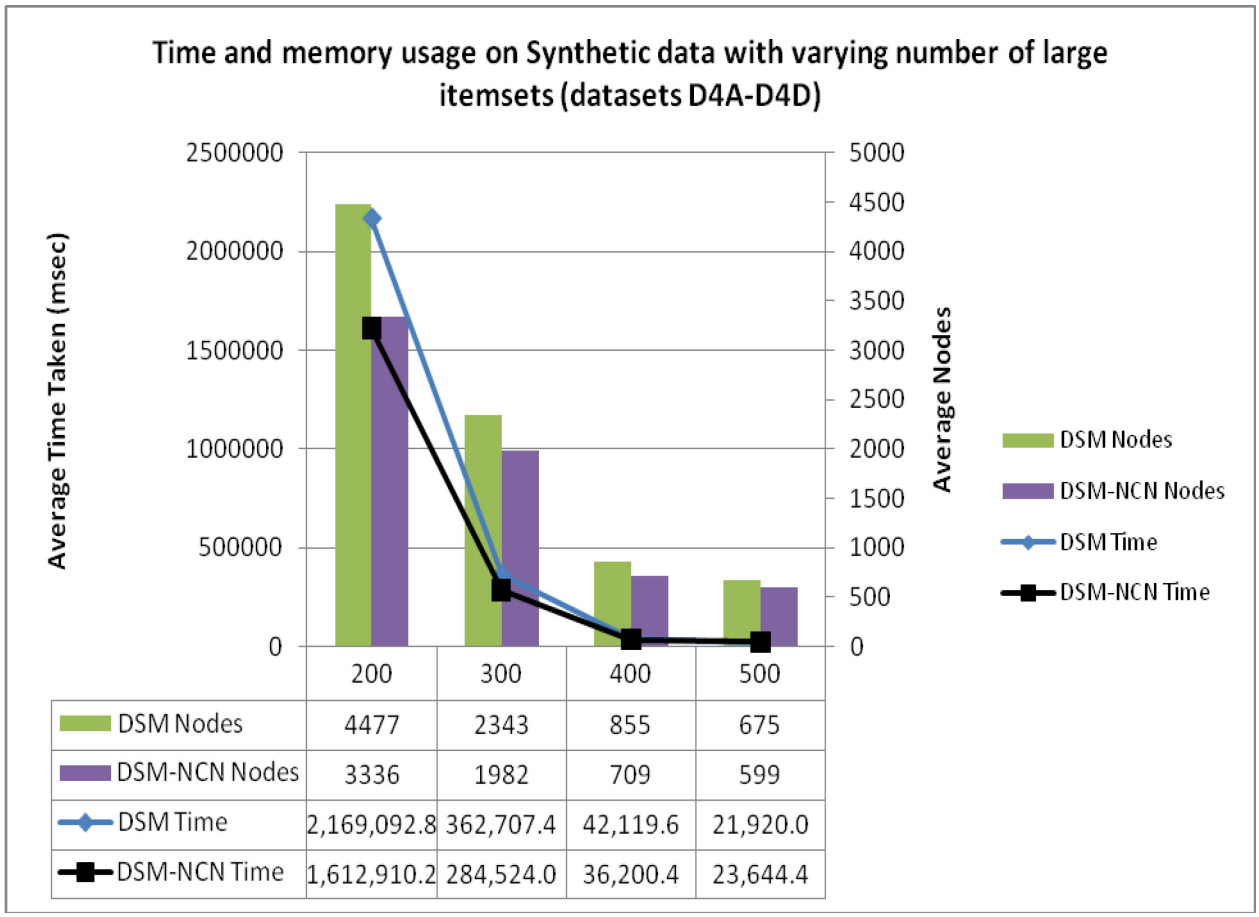


	1000	2000	10000
DSM Nodes	1944	1290	544
DSM-NCN Nodes	1612	1099	520
DSM Time	266,884.6	111,986.8	19,622.4
DSM-NCN Time	211,605.8	89,735.4	17,840.8

Figure 5.3: Graph comparing time and memory usage on varying number of unique items (datasets D3A-D3C).

Figure 5.3 shows that variation in the number of unique items has a reverse impact on time and memory usage for both implementations. As the pool size increases, the dataset becomes sparser i.e. there is a smaller probability that a given group of items will occur together and this in turn will results in lesser itemsets to be processed. DSM-NCN performs better in all the experiments, while the level of improvement increases in the reverse direction of the number of unique items. From this evidence we can suggest that datasets with a smaller pool of unique items will have more frequent closed nodes which in turn will result in a greater improvement of DSM-NCN over DSM implementation.

The next experiment was on varying the number of large itemsets available in dataset. Figure 5.4 displays the time taken and memory usage to maintain closed nodes for the four datasets. These datasets were generated using the same settings (refer to table 4.1) except for the differences in the number of large itemsets which varied in the range: (200, 300, 400, and 500).



	200	300	400	500
DSM Nodes	4477	2343	855	675
DSM-NCN Nodes	3336	1982	709	599
DSM Time	2,169,092.8	362,707.4	42,119.6	21,920.0
DSM-NCN Time	1,612,910.2	284,524.0	36,200.4	23,644.4

Figure 5.4: Graph comparing time and memory usage on varying number of large itemsets (datasets D4A-D4D).

Figure 5.4 above shows the result of varying the number of large itemsets available for the dataset. The result of time taken and memory usage show similarity to the effect produced by the previous experiment on varying the number of unique items (datasets D4A-D4C). As there are less large itemsets available in the pool there is more opportunity for the same large itemsets to appear in transactions and subsequently increase the proportion of frequent closed nodes. From the graph above we can see that DSM-NCN has a better performance advantage when the pool size is small. This advantage decreases as pool size increases until any gains in using NCN is outweighed by the extra processing overhead of generating NCNs.

5.3.3 Per pass analysis of synthetic data experiments.

From the previous analysis we have provided explanations for the improvements that DSM-NCN has been able to achieve. In this section we will examine how DSM and DSM-NCN perform with respect to time taken per pass. We use the Figure 5.5 to illustrate how per pass time is measured in our experimentation.

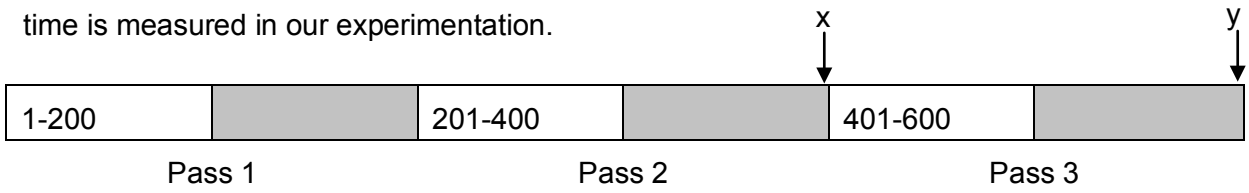


Figure 5.5: Example DSM and DSM-NCN per pass process from pass 1 to 3.

Suppose that the frame size is 200. The start time of pass 3 (x) is the time right before transaction 401 has been processed. The end time of pass 3 (y) is time after main and overhead processes of transactions 401-600 have been completed and the pass time in milliseconds for pass 3 is then given by y-x.

From previous experiments on synthetic datasets (Experiment 2A), we have collected the per pass time data for four datasets. Each datasets was chosen to represent settings that provided the biggest difference in mining time between DSM and DSM-NCN implementations so that the differences can easily be identified. The following four characteristics were chosen: average large itemset size = 5 (dataset D1C), average transaction size = 8 (dataset D2B), number of unique items = 1000 (dataset D3A), and number of large itemsets = 200 (dataset D4A).

During the experiments the time taken per each pass (in milliseconds) was collected for all synthetic datasets for both implementations. Figures 5.6 to 5.9 display trend lines graphs using a moving average method with a window size of 30. The trend lines show the average time taken from five samples per pass. Windows size of 30 were chosen as it provide a good compromise between reducing variation between passes while retaining trends that occur naturally within the mining process.

The per pass analysis on all datasets (synthetics and real world) is shown in Figures 5.6 to 5.11, the time taken for the first two passes for both implementations were omitted as they were running identical code across the two implementations.

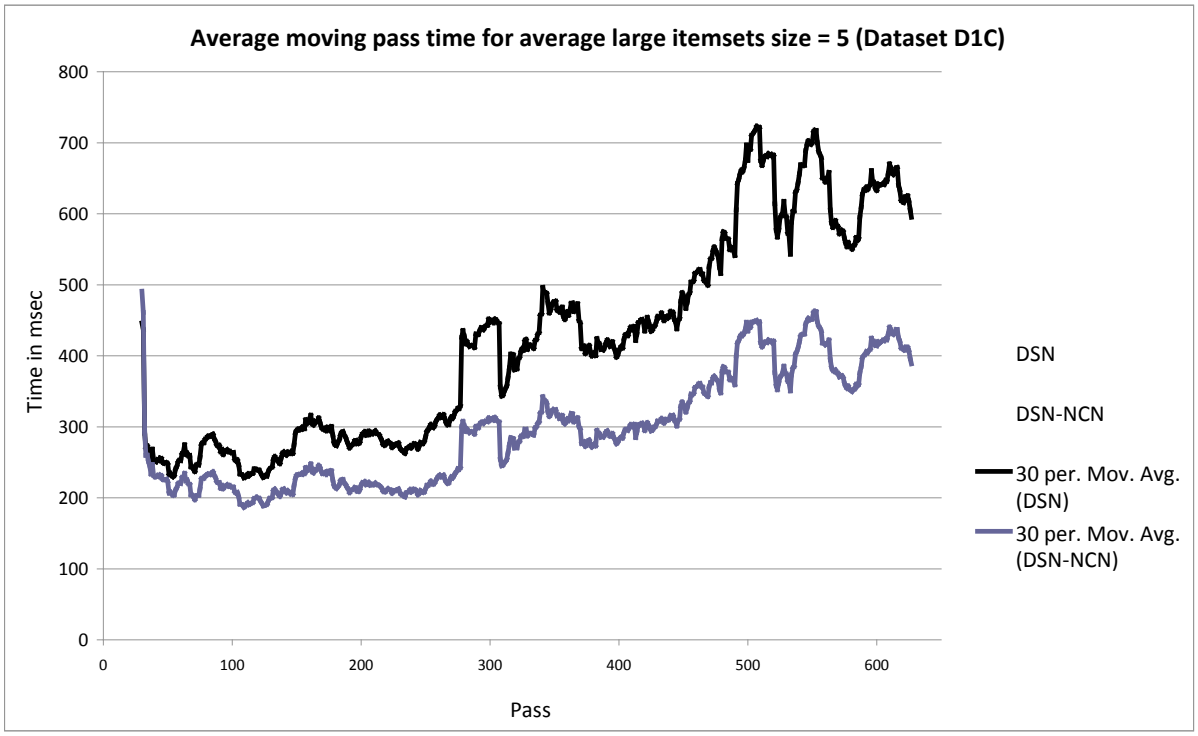


Figure 5.6: Moving average (window size 30) of per pass time taken on average large itemset size = 5 with s=0.05, delta=0.1.

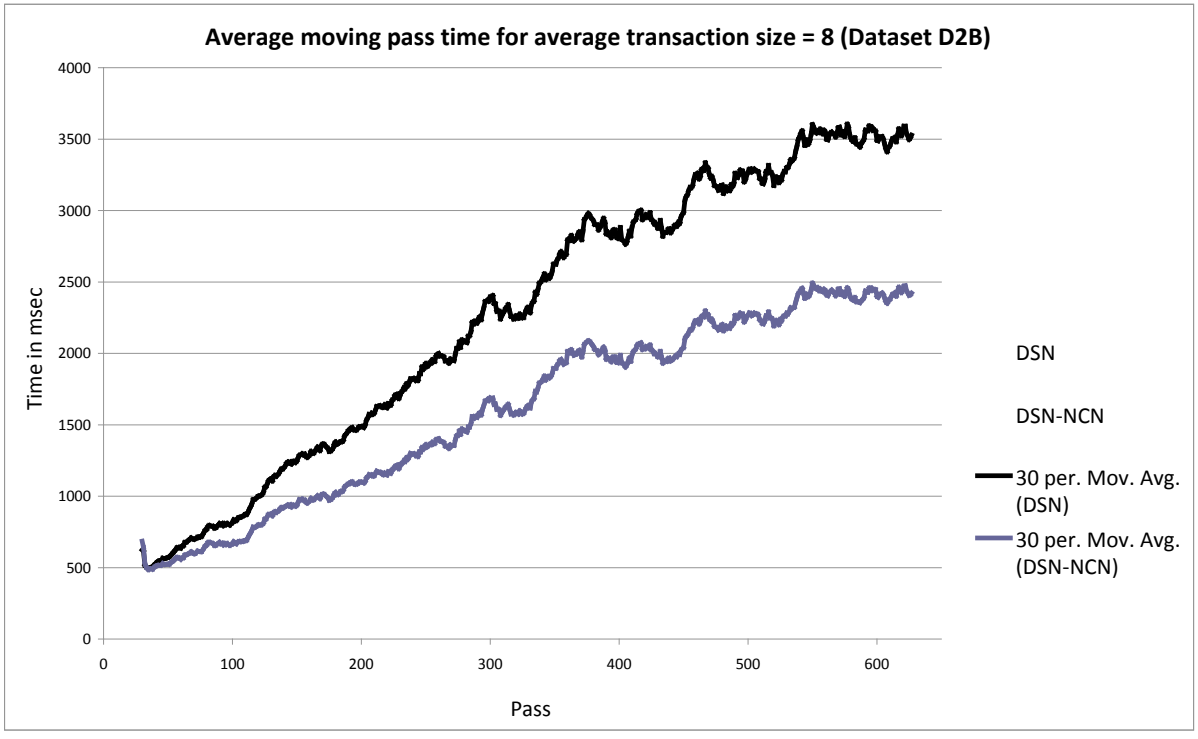


Figure 5.7: Moving average (window size 30) of per pass time taken on average transaction size = 8 with s=0.05 delta=0.1.

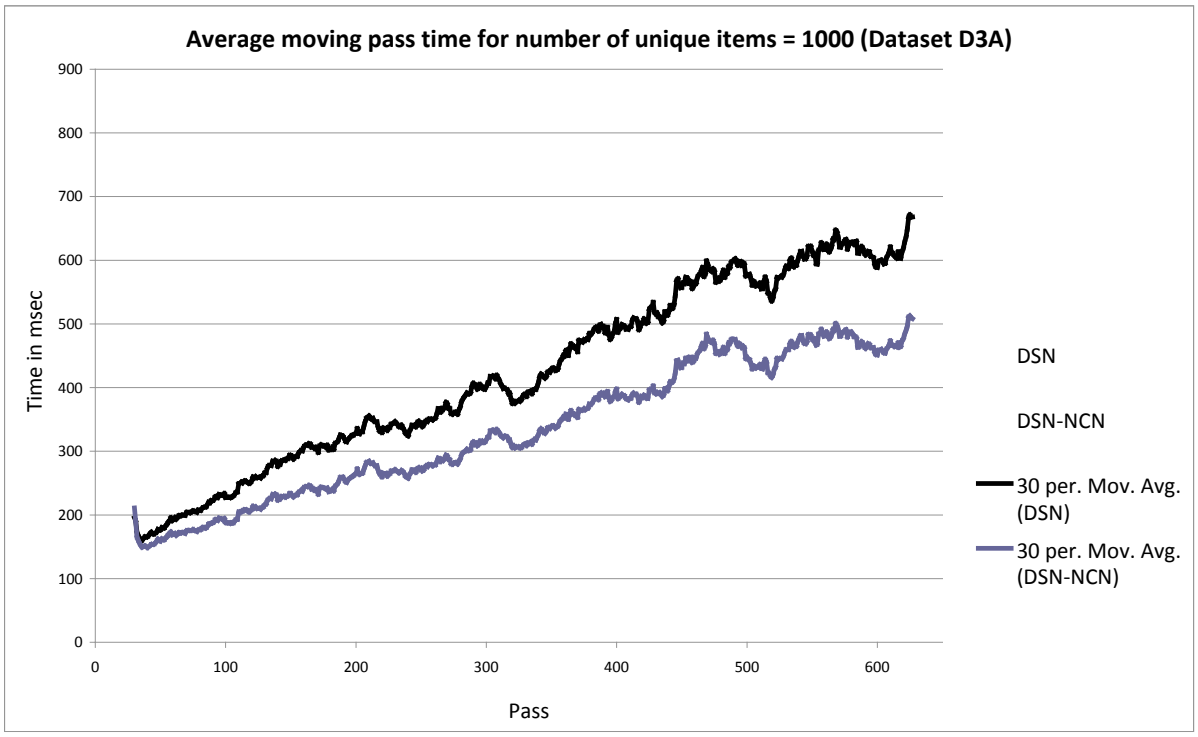


Figure 5.8: Moving average (window size 30) of per pass time taken on number of unique items = 1000 with s=0.05, delta=0.1.

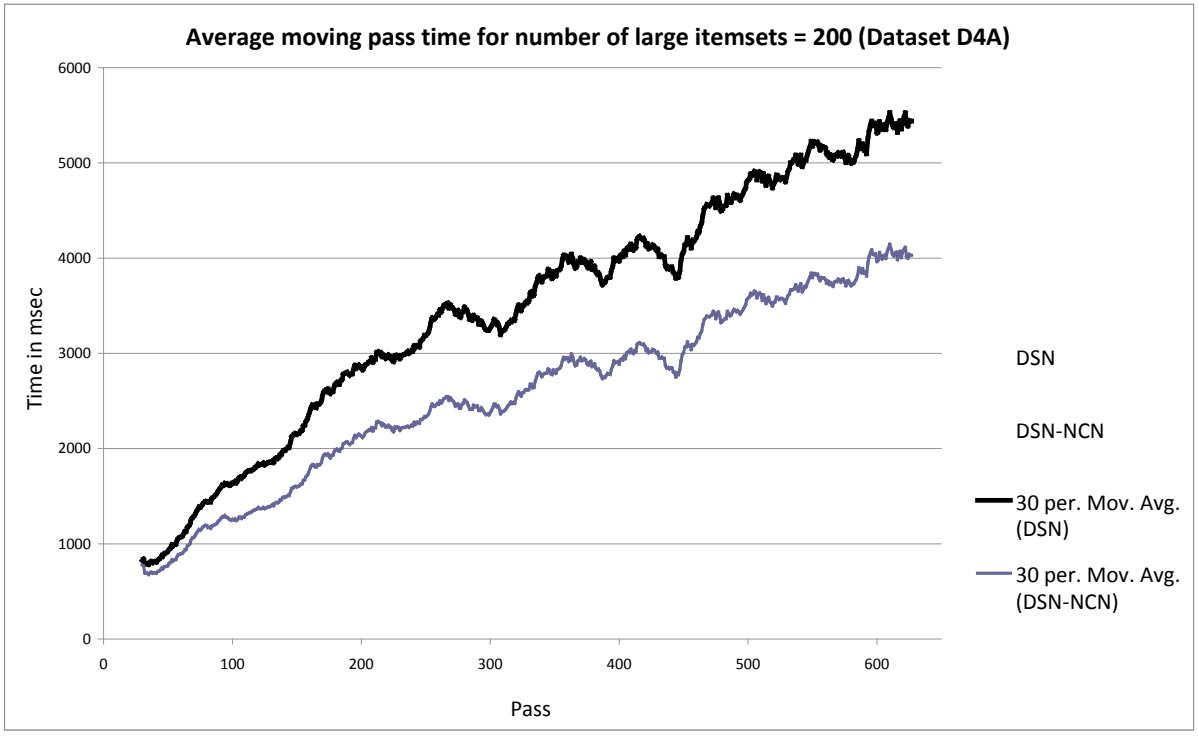


Figure 5.9: Moving average (window size 30) of per pass time taken on number of large itemsets = 200 with s=0.05, delta=0.1.

From the per pass analyses represented in Figures 5.6 to 5.9 the following observations can be made.

- At the very beginning of the experiment (Starting at pass 3), the time taken on DSM-NCN is higher than DSM implementation due to overhead needed to produce Near Closed Nodes (NCNs) and the lack of sufficient numbers of NCNs to benefit closed node generation.

- After stage 1, the average time taken from DSM-NCN drops to match those from the DSM. Thereafter, pass times for both implementations start to increase at a consistent rate.

- Rate of increase in per pass mining time of DSM is greater than that of DSM-NCN. These shown by overall improvement in DSM-NCN's run time.

From observation of stage 3 we can predict that DSM-NCN implementation could provide much better improvement if we increase the size of the datasets (as gap gets wider over time). Datasets with 100,000 transactions that we have used are too small to demonstrate the true benefit of the DSM-NCN implementation over DSM.

5.3.4 Experiment 2B: Findings and Analyses from experiments of real world data

Experiments on synthetic datasets were presented in previous sections. It provides the opportunity to control key parameters which is not possible with real world datasets. However, it is necessary that experimentation with synthetic data is augmented by experimentation with real world datasets as it provides the opportunity on assessing the extent to which synthetic data reflects conditions present in actual data. By extension; it also enables us to test whether the performance advantages of DSM-NCN over DSM translate into real world datasets. Table 5.5 shows the result summary of time and memory performance on synthetic dataset experiments.

First we experiment with the kosarak_click_stream dataset (Goethals & Zaki, 2003) with varied minimum support values of 0.1 and 0.05, and Delta values of 0.1, 0.05, and 0.01. The support setting of 0.01 was omitted from this experiment due to time constraints.

Experiment	Support	Delta	Average Total Time		% Diff	Average nodes Created		Trans per pass
			DSM	DSM-NCN		DSM	DSM-NCN	
Varied Support/Fixed Delta	0.1	0.1	676,729	548,446	18.96%	367	234	79
	0.05	0.1	3,761,316	1,972,024	47.57%	2,757	1,654	159
Varied Delta/Fixed Support	0.05	0.1	3,761,316	1,972,024	47.57%	2,757	1,654	159
	0.05	0.05	2,876,190	1,479,278	48.57%	2,099	1,259	187
	0.05	0.01	2,294,839	1,156,172	49.62%	2,051	1,261	251

Table 5.5: average time and average number of nodes created on kosarak_click_stream.

Table 5.5 shows that varying minimum support has a major impact on both processing time and memory usage that impact on both implementations. As minimum support decreases there are more closed nodes which require more processing time and memory. However decreasing the Delta value improves speed and memory usage as more transactions can be processed per pass thus reducing the amount of overheads involve in tree pruning and NCN generation (only for DSM-NCN).

DSM-NCN provides significant improvement with an almost 50% faster speed advantage at minimum support of 0.05, while offering around a 19% improvement at the 0.1 support level. An increment of 1% in speed advantage is achieved when Delta is decreased in the range of 0.1, 0.05, and 0.01. In terms of memory usage DSM-NCN takes around 40% less memory compared to the DSM implementation.

Experimentation on the kosarak_click_stream datasets provide the best result for DSM-NCN implementation compared to all other experiments. At settings of s=0.05 and Delta=0.01 it take around 50% less time to complete than DSM. We used a moving average per pass time to understand how both implementations perform at various stages of the mining process.

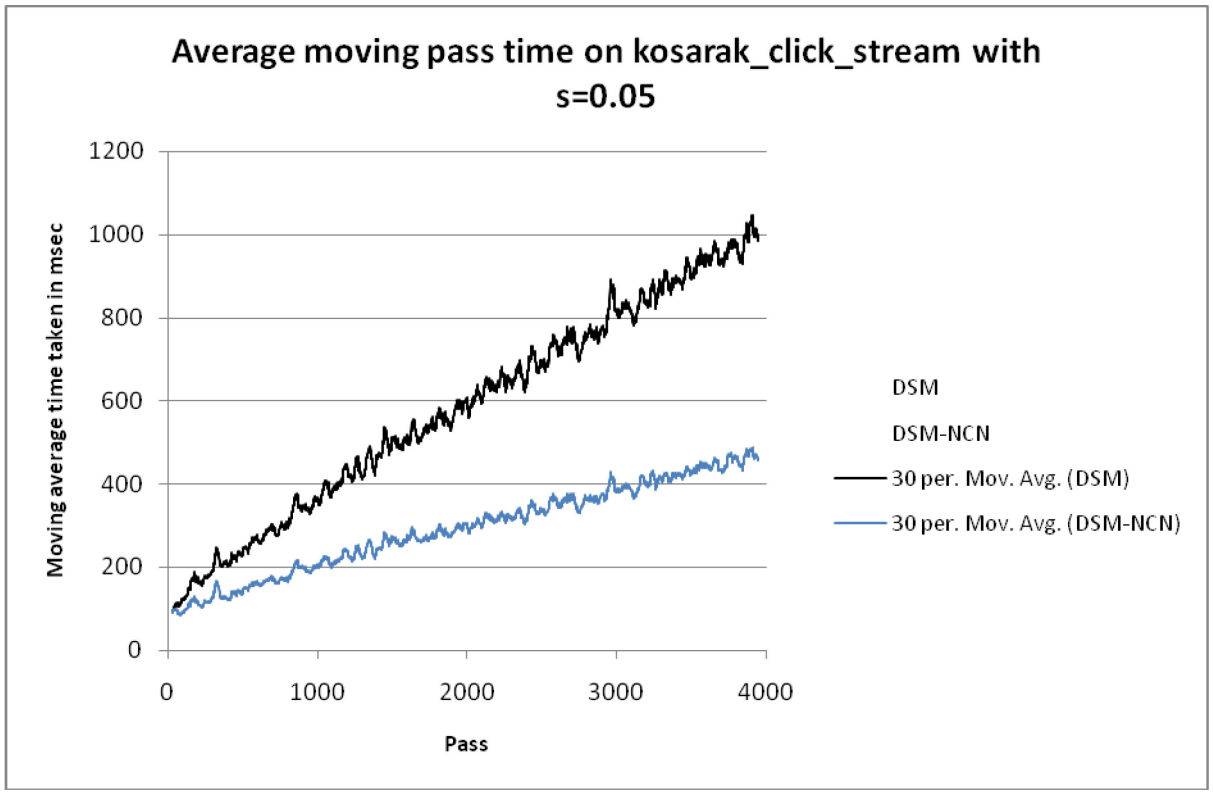


Figure 5.10: Moving average (window size 30) of per pass time taken on kosarak_click_stream with s=0.05, delta=0.01.

From average per pass time it shows that at the initial stage (first 100 passes) both algorithms perform similarly in terms of pass time. As more data streamed in, the average per pass time continued to grow. Figure 5.10 clearly shows constant rate of increase for both algorithms, however increase rate of DSM-NCN is lower. As a consequence the gap in processing widens continuously as the stream progresses in length. This result mirrors the trends observed with synthetic datasets (in previous section) but with much greater scale.

We experimented with the retail dataset using a support range of 0.1, 0.05, and 0.01 and Delta range of 0.1, 0.05, and 0.01. Table 5.6 exhibits the results of the retail datasets.

Experiment	Support	Delta	Average Total Time		% Diff	Average nodes Created		Trans per pass
			DSM	DSM-NCN		DSM	DSM-NCN	
Varied Support/Fixed Delta	0.1	0.1	15,837	15,588	1.58%	*366*	298	79
	0.05	0.1	55,084	47,254	14.21%	1,609	1,329	159
	0.01	0.1	57,299,777	54,678,019	4.58%	44,809	43,005	627
Varied Delta/Fixed Support	0.05	0.1	55,084	47,254	14.21%	1,609	1,329	159
	0.05	0.05	36,675	31,156	15.05%	1,266	1,051	187
	0.05	0.01	31,311	25,449	18.72%	1,266	1,036	251

Table 5.6: average time and average nodes created on retail datasets.

Table 5.6 shows decreasing the minimum support setting causes processing time and memory usage utilisation to increase similar to the experiment conducted with the kosarak_click_stream datasets. Interestingly processing time for support of 0.01 were more than 1,000 times greater than at 0.05 and gave rise to 30 times more closed nodes. This explosive increase in time and memory is a direct result of an exponential increase in the number of frequent itemsets at support levels between 0.05 and 0.01. Decreasing the delta value reduces both time taken and memory usage similar to the trend observed with the kosarak_click_stream datasets.

Interestingly in retail, DSM-NCN does not provide the same level of improvement success as the result for kosarak_click_stream. This may be due to the fact that the number of transactions in retail is much smaller than the latter (88,162 compare to 990,002 in kosarak_click_stream). Thus it may not be representative of a data stream environment to the same extent as kosarak_click_stream.

Settings of s=0.05 and Delta=0.01 provide the best improvement for retail. We used a moving average for the per pass time to understand how both implementations perform at various stages of the mining process.

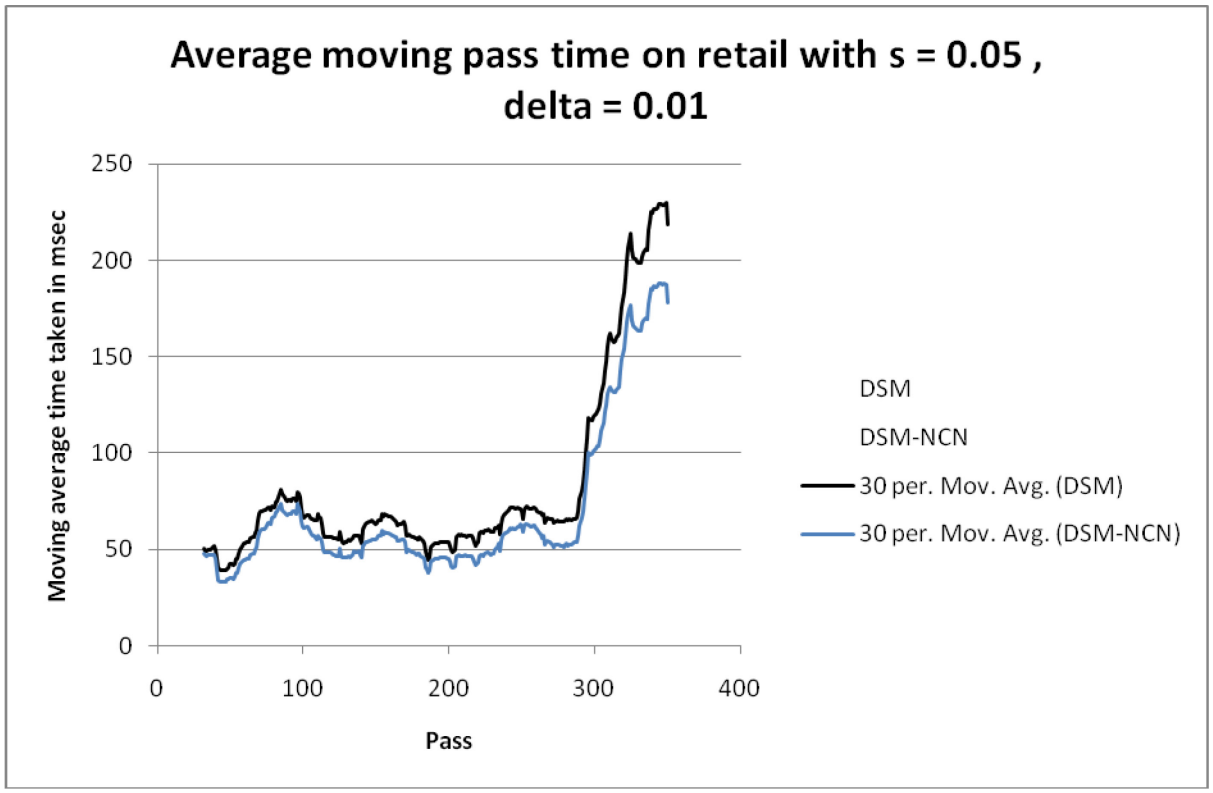


Figure 5.11: Moving average (window size 30) of per pass time taken on retail with s=0.05, delta=0.01.

From Figure 5.11 it can be seen that the improvement for the moving average time for DSM-NCN is barely significant for passes below 300. However we can see the significant differences between average moving times at passes beyond 350. These observations provide support to our earlier assertions about the performance advantage of DSM-NCN on the **retail** dataset.

Overall the DSM-NCN algorithm provides improvement in all of the retail datasets and its performance characteristics are similar to that of the kosarak_click_stream set of experiments. However time improvement for DSM-NCN when minimum support is set at 0.01 decreases to 4.58 % (compared to 14.21% improvement when support set at 0.05). This could be due to the majority of frequent itemsets existing between the support of 0.05 and 0.01 not been identified as NCNs and thus the benefits of DSM-NCN were not fully utilised.

5.4 Finding and Analysis Summary

In experiment 1, DSM and DSM-NCN implementations perform equally in terms of accuracy. Both implementations generated the same set of association rules on all three test files which included both synthetic and real world datasets.

In term of speed performance, varying in datasets characteristics and/or experiment parameters have major impact in the mining time. From observation, increase amount of frequent itemsets or reduce minimum support will greatly increase mining time.

Experiment 2A and 2B exhibit that DSM-NCN implementation has performed better on almost all experiments on synthetic data (24 out of 25 experiments perform better). DSM-NCN is faster in all experiments with real world datasets. These results agreed with our previous hypothesis, as NCN implementation bypasses many close nodes generation processes that require CPU intensive tree traversal operation.

In term of memory utilization, varying various datasets characteristics and/or experiment parameters have a major impact on the utilization. From observation, increase in the number of large frequent itemsets in the transaction will generally increase the number of frequent closed itemsets maintained by both implementations.

In experiment 2A and 2B, DSM-NCN performs better across all synthetic and real world datasets in term of memory utilisation (created fewer nodes). This confirmed our initial hypothesis, as NCN implementation bypass many of closed nodes generation processes that generate a lot of itemsets while maintained minimum amount of Near closed nodes.

5.5 Chapter Summary

The experimentation carried out on both synthetic and real world data confirmed all of our hypotheses that the use of near closed itemsets helped to improve both speed performance and memory utilisation while accuracy performance was maintained. The greatest performance was obtained when present with dense datasets and low minimum support (> 0.05).

Chapter 6 Conclusion

This chapter will provide some insights into possible directions for this research. It will provide overall observations, a summary of the research; design, implementation, experimentation, and the insights gained from this research.

6.1 Observations

In the previous chapter, we have shown that the Near Closed Nodes (NCN) technique provides up to fifty percent improvement in term of speed performance compare to the original DSM implementation with no loss in accuracy. The experiments conducted on both synthetic and real world dataset shown that NCN implementation improve both speed and memory usage in almost all circumstances.

In far as the NCN implementation goes, it is a small add-on that was built on top of DSM implementation. As discussed in Chapter 3, the NCN routine was added to mineData() routine to avoid and reduce the amount of work required to generate new closed nodes. After NCN has been incorporated within DSM, only a small fraction of code was added and all of the original implementation has been maintained. The minimal amount of changes suggested that there is a lot of scope for further optimisation with the DSM-NCN implementation.

The experimental results suggested that the performance advantage of NCN implementation increases with a greater number of closed nodes. This phenomenon can be seen from the experiment conducted on synthetic data as we artificially increase the number of frequent itemsets in the datasets, or as we decrease the minimum support setting on the experiment conducted with real world data.

During the design stage of this research attention was focused on two optimisation techniques: optimization of powerset generation the use of Near Closed Nodes. Optimizing powerset generation procedures offer a high level of improvement as they occupy a major part of the mining process. However the NCN technique that was chosen was quite a conservative one compared to what will be discussed in the next section.

6.2 Future Work

In previous research (Kadam, 2009), the author incorporated and experimented with concept drift and compared performance to FPDM2 (Yu et al., 2004). However in this research we

could not investigate the effect of concept drifts due to time constraint and scope of our dissertation.

Furthermore, we would like to incorporate much larger datasets into the experimentation to uncover any greater benefit of DSM-NCN. Based on the per pass analyses in Chapter 5 we have uncovered a growing gap between DSM and DSM-NCN. It suggests that DSM-NCN may possess greater speed when used with a real data stream.

The author of the DSM implementation has suggested various techniques to improve the DSM implementation including deferring update of nodes' support information until the end of frame, the use of estimation techniques, use of more efficient powerset generation techniques, and the use of maximal frequent itemsets.

From observations there are several other improvements that can be made on DSM implementation and the NCN extension. There are two major DSM optimisations on frequent closed itemsets generation procedures which we would like to investigate further.

One major problem with the DSM implementation is the amount of powersets that are required to generate at each transaction. These powersets are necessary for frequent closed itemsets generation, but they required a great deal of resources to generate and process. DSM had incorporated the Apriori principle preventing some unnecessary powersets generation. Nevertheless other pruning techniques could be incorporated to further reduce powerset generation. These techniques had been used by implementations such as CLOSET+ (Wang, Han, & Pei, 2003), CLOSET (Pei et al., 2000), and CHARM (Zaki & Hsiao, 2002).

The other major constraint we encountered was the amount of resources consumed to identify frequent closed nodes and support information employed by DSM implementation. As the number of closed nodes glow it will take correspondingly longer to identify and generate new closed nodes.

We believe that by increasing the capability of NCN generation we could further the improve speed of the implementation. At the moment the NCN algorithm is designed to heuristically detect some NCNs from closed node subsets. This design was chosen to minimize overhead from NCN processes and the possibility of false-positive results. However, if we increase the number of NCNs generated by using more aggressive NCN algorithms we could further increase the number of NCNs generated and potentially further improve run time performance.

6.3 Conclusion

Data mining will be an increasingly important knowledge generation task in most organisations as data continues to grow in size. In our opinion data stream mining possesses one of the most challenging and exciting fields of data mining research. As hardware capabilities and improved mining techniques continue to grow it is natural that we take advantage of emerging data mining applications.

In this research we have developed a simple method which provides improvements to both speed and memory utilisation for a closed itemset mining implementation. We selected near closed nodes (NCN) add-on to be implemented on top of the current DSM implementation. NCN is used to reduce time taken for DSM to determine frequent closed itemesets and support information. Experimental studies show that NCN add-on provides increase in performance on both mining time and memory usage of DSM algorithm.

From the experiments we have found that NCN is particularly robust to changes of key data parameters. It outperforms the original implementation in most of our experiments. In terms of performance in closed nodes mining NCN particularly excels at low minimum support or in dense datasets.

References

Agrawal, R., Imieliński, T., & Swami, A. (1993). *Mining association rules between sets of items in large databases.* Paper presented at the Proceedings of the 1993 ACM SIGMOD international conference on Management of data, Washington, D.C., United States.

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., & Verkamo, A. I. (1996). Fast discovery of association rules. *Advances in knowledge discovery and data mining, 12*, 307-328.

Agrawal, R., & Srikant, R. (1994). *Fast algorithms for mining association rules*.

Bamshad, M., Honghua, D., Tao, L., & Miki, N. (2001). *Effective personalization based on association rule discovery from web usage data.* Paper presented at the Proceedings of the 3rd international workshop on Web information and data management, Atlanta, Georgia, USA.

Borgelt, C., & Kruse, R. (2004). Apriori-Association Rule Induction/Frequent Item Set Mining.

Charikar, M., Chen, K., & Farach-Colton, M. (2004). Finding frequent items in data streams. *Theoretical Computer Science, 312*(1), 3-15.

Chi, Y., Wang, H., Yu, P. S., & Muntz, R. R. (2004). *Moment: Maintaining closed frequent itemsets over a stream sliding window*.

Ching-Jui, H. (2005). Efficient Algorithms for Mining Closed Itemsets and Their Lattice Structure. *IEEE Trans. on Knowl. and Data Eng., 17*(4), 462-478.

Cristofor, L. (2006, 3 October). *ARtool Info Index*. Retrieved August, 2009, from http://www.cs.umb.edu/~laur/ARtool/

Domingos, P., & Hulten, G. (2001). *A General Method for Scaling Up Machine Learning Algorithms and its Application to Clustering*.

Gaber, M. M., Zaslavsky, A., & Krishnaswamy, S. (2005). Mining data streams: a review. *ACM Sigmod Record, 34*(2), 26.

Goethals, B., & Zaki, M. J. (2003). *Frequent itemset mining implementations repository*. Retrieved August, 2009, from http://fimi.cs.helsinki.fi/

Gouda, K., & Zaki, M. J. (2001). *Efficiently mining maximal frequent itemsets*.

Haixun, W., Wei, F., Philip, S. Y., & Jiawei, H. (2003). *Mining concept-drifting data streams using ensemble classifiers.* Paper presented at the Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, Washington, D.C.

Han, J., Pei, J., Yin, Y., & Mao, R. (2004). Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery, 8*(1), 53-87.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *Management information systems quarterly, 28*(1), 75-106.

Hulten, G., Spencer, L., & Domingos, P. (2001). *Mining time-changing data streams.* Paper presented at the Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, San Francisco, California.

Jiang, N., & Gruenwald, L. (2006a). *CFI-Stream: mining closed frequent itemsets in data streams.* Paper presented at the Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, Philadelphia, PA, USA.

Jiang, N., & Gruenwald, L. (2006b). Research issues in data stream association rule mining. *ACM Sigmod Record, 35*(1), 19.

Kadam, O. V. (2009). *Novel applications of Association Rule Mining-Data Stream Mining.* AUT, Auckland.

Kifer, D., Ben-David, S., & Gehrke, J. (2004). *Detecting change in data streams.* Paper presented at the Proceedings of the Thirtieth international conference on Very large data bases - Volume 30, Toronto, Canada.

Lee, W., & Stolfo, S. J. (2000). Data mining approaches for intrusion detection: COLUMBIA UNIV NEW YORK DEPT OF COMPUTER SCIENCE.

Mao, G., Wu, X., Zhu, X., Chen, G., & Liu, C. (2007). Mining maximal frequent itemsets from data streams. *Journal of Information Science, 33*(3), 251.

Orlikowski, W. J., & Baroudi, J. J. (1991). Studying information technology in organizations: Research approaches and assumptions. *Information Systems Research, 2*(1), 1-28.

Pasquier, N., Bastide, Y., Taouil, R., & Lakhal, L. (1999). Discovering frequent closed itemsets for association rules. *Lecture Notes in Computer Science*, 398-416.

Pei, J., Han, J., & Mao, R. (2000). CLOSET: An efficient algorithm for mining frequent closed itemsets. *DMKD'00*.

Ravikumar, P., & Lafferty, J. (2004). *Variational Chernoff bounds for graphical models*.

Straub, D., Gefen, D., & Boudreau, M. C. (2004). The isworld quantitative, positivist research methods website. *Electronic Source*.

Tan, P. N., Steinbach, M., & Kumar, V. (2005). *Introduction to data mining*: Pearson Addison Wesley Boston.

Wang, J., Han, J., & Pei, J. (2003). *CLOSET+: searching for the best strategies for mining frequent closed itemsets.* Paper presented at the Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, Washington, D.C.

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, USA.

Yu, J. X., Chong, Z., Lu, H., & Zhou, A. (2004). *False positive or false negative: Mining frequent itemsets from high speed transactional data streams*.

Zaki, M. J., & Hsiao, C. J. (2002). *CHARM: An efficient algorithm for closed itemset mining*.

Zheng, Z., Kohavi, R., & Mason, L. (2001). *Real world performance of association rule algorithms*.

Zhu, Y., & Shasha, D. (2002). *Statstream: Statistical monitoring of thousands of data streams in real time.* Paper presented at the Proceedings of the 28th international conference on Very Large Data Bases, Hong Kong, China.