A Thesis Presented
To

The Faculty of the School of
Science, Engineering and Environment

The University of Salford

Analyzing Frequent Patterns in Data Streams Using a Dynamic Compact Stream Pattern Algorithm

In Partial Fulfillment

Of the Requirement for Degree of
Doctor of Philosophy

By

Oyewale Ayodeji

- April 2022 -

Table of Contents

List of Figures

List of Tables

ACKNOWLEDGEMENT

Crowning of a long journey that the making of this dissertation happens, I would like to thank some of many people who have been most helpful to me in ways unimaginable.

First, I would like to thank Dr Chris Hughes (Supervisor) for his support and guidance over the years, and for the numerous and fruitful discussions that laid the foundation of the research presented here. In particular, I thank him for passing knowledge on to me, and teaching me how to research and solve new problems with persistence. I also thank Professor Mohammed Saraee (Joint Supervisor) for valuable discussions on statistical modelling and association rule mining that led to the discovery of this research. I am grateful to both of them for many helpful discussions and brainstorms on stream data mining and data mining at large, and for collaborations on some projects. Their help in cultivating my curiosity in machine learning and data mining as it is indispensable to the formation of this dissertation. I thank the colleagues and friends at the Science, Engineering and Technology Department. Thank you for your helpful discussions and paper proofreading, and the enjoyable environment you have helped to maintained.

Special thanks go to Busayo Oyewale, my wife and my personal editor and my new born baby Kian Oyewale for the sleepless nights. She has not only kept me cheerful and happy throughout the development of this dissertation, but also showed her exemplary proofreading skills in helping to make sure this research is a success. Above all, I will like to render my profound appreciation to my parents, MR Kehinde Olufemi Oyewale and Mrs. Gbonjubola Oyewale for their endless support all through the course of this research. For their emotional, financial and inspirational encouragements.

Abstract

As a result of modern technology and the advancement in communication, a large amount of data streams are continually generated from various online applications, devices and sources. Mining frequent patterns from these streams of data is now an important research topic in the field of data mining and knowledge discovery. The traditional approach of mining data may not be apropriate for a large volume of data stream environment where the data volume is quite large and unbounded. They have the limitation of extracting recent change of knowledge in an adaptive mode from the data stream. Many algorithms and models have been developed to address the challenging task of mining data from an infinite influx of data generated from various points over the internet. The objective of this thesis is to introduce the concept of Dynamic Compact Pattern Stream tree (DCPS-tree) algorithm for mining recent data from the contineous data stream. Our DCPS-tree will dynamically achieves frequency descending prefix tree structure with only a single-pass over the data by applying tree restructuring techniques such as Branch sort method (BSM). This will cause any low frequency pattern to be maintained at the leaf nodes level and any high frequency components at a higher level. As a result of this, there will be a considerable mining time reduction on the data-set.

Keywords:

Data Mining, Frequent Pattern, Stream data, Compact Stream Pattern, Interactive Mining

CHAPTER ONE

1.0 BACKGROUND OF STUDY

Many organizations need to understand and monitor the rate of data influx that are generated from various sources. Most of these datasets generated are in the form of a stream (data stream), thereby posing the challenge of being continuous. In real-life scenario, stream data come from live observations or in some cases in the form of sensors which are subject to noise. This problem is usually handled by pre-processing the data. Which is why it is of great importance to design a light-weight pre-processing technique that can guarantee quality of the mining results. More often than not, data processing consumes more time in the knowledge discovery process. The challenge here is to automate such a process and integrate it with the mining techniques.

Data preprocessing in this research work involves the use of a middleware which serves as a functioning hidden translation layer which enables a communication and interaction, and data management between the operating system and the application running on it. It can be referred to as plumbing as it connects two applications together so that data can be easily passed between the pipes created by the middleware. The use of a middleware makes it possible to perform requests such as formatting of data an or allowing the system to run custom functions based on the user's specified need or requirements.

Recently, frequent patterns in data streams have been a challenging task in the field of data mining and knowledge discovery. Frequent pattern mining is one of the most recent and important research topics in the field of data mining. Lavrac et al. (2010) defined frequent pattern mining " as searches for implicit, previously unknown, and potentially useful pieces of information—in the form of frequently occurring sets of

items (also known as patterns)—that are embedded in the data.". From the inception of frequent pattern mining, many studies have been conducted. It is important to note that, most of these studies focus on traditional mining of precise data. Precise data are data type that are certain to be present in a dataset or database transaction. The user is always certainly sure to find the data within the database. Apriori algorithm was one of the earliest Frequent Pattern algorithm that was used for mining through a generate and test approach. An important observation from the Apriori algorithm is the bottleneck that is faced during the result generation process. This algorithm requires multiple database scan for patterns. Han et al. (2000) provided a solution to this problem by designing a tree-based mining frequent pattern algorithm called Frequent Pattern growth (FP-growth). The technique of this algorithm makes use of the prefix-tree data structure. A gain in performance is achieved by relying upon the compactness nature of FP-tree, where data is stored based on the frequency nature of the items in a frequency-descending order. However, the design of the FP-tree needs two database scans as well as a prior knowledge of the dataset support threshold. In the first database scan, FP-growth finds all frequent level-1 patterns. Then, it scans the database the second time to construct a tree data structure called an FP-tree. This tree data structure contains all information in the database. This approach of mining may suffer from two major limitations: the memory size and run-time inefficiency causing a higher time mining process. A technique called Compact Stream tree was proposed by Tambeer S. K. Et al, (2009). It is an efficient technique to discover the recent dataset of frequent patterns from a very fast data stream over a sliding window.

The objective of this thesis is to introduce the concept of dynamic Compact Pattern Stream tree (DCPS-tree) algorithm for mining recent data from the

continuous data stream. The DCPS-tree will dynamically achieves frequency descending prefix tree structure with only a single-pass over the data by applying tree restructuring techniques such as Branch sort method (BSM). This will cause any low frequency pattern to be maintained at the leaf nodes level and any high frequency components at a higher level. In this DCPS-tree, the sliding window will consist of many batches of transactions, and each transaction data will be for each batch that is maintained at each node of the tree structure. In order to discover the complete data-set of the recent frequent patterns, the Frequent Pattern growth (FP-growth) mining method is implemented to the DCPS-tree of the current sliding window. As a result of this, there will be a considerable mining time reduction on the datasets.

Conclusively, the mining of the patterns frequency from dynamic data streams is more challenging when compare to the mining of traditional static transaction databases, as a result of the following data streams characteristics:

i) Data streams are continuous and unbounded.

ii) Data in the streams are not uniformly distributed.

In this thesis, we propose a Dynamic Compact Stream Pattern algorithm that will successfully overcome the above two challenges. This study, therefore, conceptualized handling enormous data as a stream mining problem that applies to continuous data stream and proposes an ensemble of unsupervised learning methods for efficiently detecting data knowledge in the stream of data.

1.1 STATEMENT PROBLEM

The vast amount of data generated by enterprise networks has led to increasing adoption of big data techniques in delivering actionable analytic from collected

data. Various companies and market in an attempt to gain sufficient market share in a competitive industry require analytic solutions to make actionable decisions in real time to latter predictions. Given the vast amount of applications introduced in different organizations and domains, the wide range of collected data provides a haven for machine learning algorithms to help in analyzing and proscribing anomalous events before they occur. Traditionally, data mining techniques have been applied to gain insights into large unstructured data. However, few studies have applied CSP algorithm for effectively analyzing stream data.

Let us reflect on the following vital points :

1. The traditional approach of mining data is not appropriate for a large volume of data stream environment where the data volume is very large and unbounded.
2. The traditional approach of mining often suffers from two major limitations: the memory size and run-time inefficiency .
3. Data found in streams are usually non-uniformly distributed and poses the challenge of effective mining.

There are the reasons that necessitated the desired to develop an algorithm that can be applied to address the above listed problem.

1.2 THESIS CONTRIBUTIONS

The gains of this research work are numerous. The proposed system will definitely achieve the following contributions:

1. The Dynamic Compact Stream Pattern tree algorithm could be used in solving the problems of mining which support single database scan;

2. It could be used and applied in different industrial fields such as sensor network analysis and network monitoring system, where real time data are needed for processing;
3. It will add knowledge to the existing knowledge of data science where data mining algorithm is a requirement.

1.3 OBJECTIVE OF THE STUDY

At the end of this research work, we should be able to develop a dynamic compact prefix-tree (DCP-Tree) algorithm that would be used to mine frequent patterns from a stream of data in various domains.

And the specific objectives are:

i. design a Dynamic Compact Stream Pattern tree (DCSP-tree) algorithm for mining frequent pattern in a dynamic data streams environment ;

ii. detect new patterns in real-time streams of high-speed data and adapting quickly to changing realities.

v. Design and implementation of a prototype of stream mining system for stream processing and normalization of data.

1.4 THESIS ORGANIZATION

This report covers a detailed study on analyzing stream data using a compact stream pattern algorithm based on constructing a prefix tree in a lexicographical order. The report is organized as follows:

1. Chapter one: The first chapter gives a general introduction and overview of the research study by stating the problem, the objective and contributions .
2. Chapter two: highlights the general background on data streams, existing data stream algorithms and compact data structures. Historical reviews, applications, characteristics and the general architecture of the basic ideas were discussed in this chapter.
3. Chapter three: presents the analytic review on the general concept of data streams and CSP algorithm, in-depth approach and applications of stream data. The chapter also highlights the general overview of results when compared to the frequent pattern (FP) growth algorithm. The methodology was adopted from the related literature. Finally, the presentation of a preliminary test of the proposed CSP algorithm. The analysis determined the performance evaluation of the proposed algorithm against the existing protocol.
4. Chapter four: presents the implementation and discussion of the algorithm. In the implementation section, the implementation of the design concept with a detailed explanation of the concept formulation is highlighted. The algorithm and state-based model for the Netflix dataset is presented, with concluding remarks which summarizes the outcome of the implementation procedures.
5. Chapter five: highlights the key evaluation parameters, he rate of data flow. The chapter also presented some generic results of the algorithm by predicting the mining times and run-time efficiency. The chapter also presents the expansion and results of the analytic models
6. Chapter six: deals with the general summary , conclusion and recommendation of the Thesis.

CHAPTER TWO

2.0 REVIEW OF EXISTING LITERATURE

An increasing investment has been made in the study area of analyzing stream data during past years Ascarza et al., (2016). Since the introduction of stream mining by Agrawal *et al* (1993), it has been actively and widely studied by the data mining and knowledge discovery research community. Finding meaningful patterns from streams of data has become one of the most important and challenging problems for a wide range of online applications. Various researches focused on discovering and extracting useful information from the most recent data elements since processing the recent data is usually important and of utmost essence for the applications that handle stream-oriented data.

The intensive rivalry and replete markets have left major companies ranging from telecoms to retail, network traffic, click and web stream mining with little slack to ignore high detection rate. A wide span of stream mining algorithms has been developed in the last years. Most advanced models make use of contemporary machine learning classifiers such as random forests (Breiman; 2001; Yoon Koehler & Ghobarah 2010). Customarily, platforms intended for streaming data can only respond based on data entered for each set.

Consistent with Yaya, Xiu, Ngai, Weiyun, 2009 & Huang, Kechadi, Buckley, Kiernan, Keogh, Rashid, 2010, one of the biggest challenges for researchers in discovering meaningful patterns from stream data and the ability to make prediction in the imbalanced nature of data (Desamparados & Josep, 2016). Nowadays business firms pay more attention to make firm and effective analysis on their enormous data (ZhenYu

Chen et al., 2012; Coussement & Van den, 2008). Making sense out of stream data has been extremely outstretched as a concern inmany fields including telecommunication (Umman Tuğba, & Şimşek Gürsoy,2010; Rashid ,2016; Idris et al.,2012; Bingquan et al., 2012). Therefore, it is stoutly recommended that companies in competition within the same business environment operate their own algorithmic systems fortified with business intelligence and data mining tools to label and isolate an array of incoming data elements for better business advantage. Research in stream data has attracted a great amount of attention in literature due to the gravity of the problem within many organizations and the purpose of making a single pass over data. Consequently, the need to accurately predict an event in near real time before its occurrence is of great importance. An extensive study by Coussement & Van den Poel (2008); Xie et al., 2009 preside over the use of Support Vector Machines (SVM) in the context to stream data mining.

Only a few representatives of those using data mining is mentioned. Most recognized algorithms are neural networks, decision tree, logistic regression, and genetic algorithm and cluster analysis. Various researches discussed strategies for combating streaming data using data mining techniques, but few have examined and studied stream mining by classifying it as a stream of data. Verbeke et al., (2011) proposed two novel data mining techniques for stream data mining, the first was named AntMiner+ which uses Ant Colony Optimization to gather rules from data and second named Active Learning Based Approach for support vector machine rule extraction and experiments were conducted with C4.5, RIPPER, SVM, logistic regression.

Experiment proved that ALBA combined with C4.5/RIPPER results in higher accuracy than the AntMiner+. Khan et al., (2010) with the aid of decision tree, logistic regression and neural network have been able to make analytics and ultimately

proposed that demographic features have the lowest effect on the data when compared to other usage details. Owczarczuk (2010) used logistic regression, Nie et al., (2011) used logistic regression and decision tree model and Keramati, Seyed, Ardabili (2011) focused on Binomial logistic regression model for large datasets and detected useful patterns. In addition, a study by Shim et al., (2012) used decision tree, neural network and logistic regression and was able to ascertain decision tree to show the most successful algorithm.

Accuracy and exactness of a prediction is very vital as the sole of a company is based on these predictions, Mozer *et al*., (2000) used neural network, logistic regression in order to attain better prediction accuracy for stream data. Several algorithms have been used to analyse static data, some of which are eminent algorithms including neural networks, decision tree, logistic regression and cluster analysis which do not put into consideration the influx and persistence of the incoming data. Therefore, "developing an approach that can effectively work on streams of data that are incessant and capable of processing millions of instances per second is highly crucial". The major drawback faced with data mining techniques is the inability to model data incrementally. Models built sometimes while using data mining techniques can gradually lose track of incoming data which can lead to opportunity lost, inability to discover and tap i n t o new markets, and reduced business velocity.

Due to the infinite amount of continuous measurements and time-evolving trends of stream data, Haixun, Wei, Philip, Jiawei (2003) proposed a general framework for mining concept-drifting data streams using weighted ensemble classifiers where ensemble classification models are trained from sequence chunks of

data streams. The ensemble approach tends to enhance both the efficiency of the learning model and the accuracy in performing classification. The idea of ensemble methodology is to build a predictive model by integrating multiple models thereby being able to improve prediction performance.

Hossein, Mostafa, & Mohammad (2014) gave a lack of proper insight into ensemble methods for stream data which led to the comparative analysis of the four standard ensemble methods comprising of Bagging, Boosting, Stacking and Voting using C4.5 Decision tree, Artificial Neural Network (ANN), Support Vector Machine (SVM) and Reduced Incremental Pruning to Produce Error Reduction (RIPPER) with the introduction of two distinct sampling techniques; that is 'oversampling for basic sampling and Synthetic Minority Over-sampling Technique (SMOTE) for advanced sampling'. Hossein et al., (2014); Martens et al., (2009) rated SVM as the best data mining algorithm in terms of performance and it was observed that Boosting has the best result among all other methods. Data stream mining techniques from research have been envisaged to offer the assurance of discovering new patterns and anomalies in high speed streams of data. Borja, Bernardino, Alex, Gavalda, Manzano-Macho (2013) arrived at a proof-of-concept in which the technology to perform real-time, high-throughput prediction on streams of items is obtainable. An approach using stream mining platform MOA (Massive Online Analysis) was used in the implementation of a decision trees called Hoeffding tree algorithm by inducing Hoeffding bound, which gives certain level of confidence on the best attribute to split the tree. An alternative approach to supervised learning is unsupervised learning which can be effectively applied to unlabeled data i.e. data in which no points are explicitly identified as anomalous or non-anomalous.

Tree algorithms are one of the most important forms of unsupervised learning (Cook & Holder; 2007; Eberle & Holder, 2007), but it has been traditionally limited to static and finite-length dataset. Parveen et al., (2013) applied decision tree to stream data which got better classification accuracy when compared with a baseline classifier which is applicable to static data stream. The ability to mine stream data addresses the rare class problem by building a model that only considers the most recent entry. The model is trained over a series of fast flowing data and the ensuing findings are classified as the result. Moreover, the approach is only applicable to bound–length static data streams. Instead, data related to stream is typically continuous and evolves over time; therefore, the data is of unbounded length. Hence, effective classification models must be adaptive (i.e. able to handle evolving concepts in the data) and highly efficient in order to build the model from huge amount of evolving data.

2.1 WHY IS REAL-TIME ANALYSIS CRUCIAL?

Information is emerging as the new currency of business, and remains the basis for true competitive differentiation, innovation, value creation, growth, and risk mitigation. Mere having access to information before industry competition is not enough today. Streaming continuous data allows companies/organizations to analyse data as soon as it becomes available allowing the ability to analysis risks before they occur. Data processing and analytics must occur in real-time and be complemented by real-time action-ability as it can help companies identify new business opportunities and revenue which will eventually result into an increase in profits, acquisition of new customers, and improved service. Above all, it provides a form of security protection as it gives companies a fast way to swiftly associate various event patterns so as to identify relationship.

	Traditional	**Stream**
Number of Passes	Multiple Pass	Single pass
Processing time	Unlimited	Limited/Restricted
Memory Usage	Unlimited	Limited/Restricted
Result Outcome	Accurate and Exact	Approximate

Table 2.0 Basic Differences between traditional and stream data processing (Adaptive:
João & Pedro, 2004)

Figure 2.1 above illustrates the significant distinctions between data residing in a database (traditional databases) and an influx stream of data. The data available in a database can be continually queried as many times as the need arises, unlike stream data which doesn't have the luxury of continuous query over a particular dataset. Which implies that only one single pass can be made over a given dataset. The main problem in handling data streams is memory constraint because we are restricted to a single scan of the database. As stated earlier, in traditional algorithms we are able to perform multiple scans over available data. This is not possible in data stream because there isn't enough memory space to store all the transaction and their counts. In regard to this, memory size is bounded and can contain the huge amount of data that arrives continuously. There are many algorithms for mining stream data. Based on result, they are categorizing as exact algorithms or approximate algorithms. In exact algorithms, The result consist of all the itemsets which satisfy support values greater than or equal to threshold support. To Produced accurate result in stream data, additional cost is

needed. In approximate algorithms, the result is approximate result with or without an error guarantee.

Managing data in motion is different from managing static data (also known as data at rest). To deal with data in motion, event stream processing relies on:

2.1.1 Assessment**:** With massive volumes of streaming data, it is simply not practical to store it all. Much of what is generated is irrelevant for any action, analysis or even archiving. Compact stream pattern algorithm standardizes the incoming data, determines if the data is relevant and if any downstream processing is needed. If not, the pattern can be discarded without taking up processing bandwidth.

2.1.2 Aggregation**:** Event stream processing is used to continuously calculate metrics across defined intervals of time to understand real-time trends. This type of continuous aggregation would be difficult with traditional tools because of the time lag and I/O overhead associated with storing data on disk and then running a query.

2.1.3 Correlation: An individual event may not be significant as a single data point. But when you can establish its relationship to multiple events from a single stream or even multiple streams, then you can set the context to assess these events. Monitoring patterns and correlations (such as identifying that Event A was followed by B and then C within a given time window) provides more accurate understanding of a situation than analyzing Event B in isolation.

2.1.4 Analysis: One of the advantages of streaming analytics is gaining as much insight as possible while the data is still in motion. Conducting systematic interpretation while events are happening ensures operational procedures are tied with existing conditions. Adding out-of-stream (i.e., traditional) analytics to events from data streams enables

you to further enrich events and put them in a business context. This approach can help you to use event stream processing to create additional revenue, improve a customer experience or reduce costs and risk.

2.2 SETBACKS OF ALGORITHMS IN LEARNING FROM DATA STREAMS

The challenging problem for stream mining is the ability to permanently maintain an accurate analytical model. These concerns require intuitive algorithms that can tweak the current models whenever new datasets are available and accessible. Furthermore, older information should be retained or rendered outmoded. Learning from data stream requires iterative learning algorithms that take into account concept drift. Solutions to these problems require new sampling and randomization techniques, and new approximate incremental algorithms. Hulten and Domingos distinguishes desirable properties of learning frameworks that are able to mine continuous, high-volume, unbounded stream of data as they arrive. This proposes learning models ought to have the option to be able to process incoming data and answer queries in real time.

2.3 COMPACT DATA STRUCTURE

In an age where collecting, updating, storing and retrieving of information becomes significant, the introduction of a dynamic, adept and compact data structure is of great essence. This is as a result of the finite memory size and the persistence convergence of the stream data. An efficient and compact data structure is needed to store, update and retrieve the collected information. This is due to bounded memory size and huge amounts of data streams coming continuously. Failure in developing such a data structure will largely decrease the efficiency of the mining algorithm because, even if

we store the information in disks, the input and output operations will increase the processing time. The data structure needs to be incrementally maintained since it is not possible to rescan the entire input due to the huge amount of data. In [Chen *et al.,* 2007], the authors employ a prefix tree data structure to store item ids and their support values, block ids, head and node links pointing to the root or a certain node. In [Giannella, 2003], an FP-tree is constructed to store items, support information and node links. A thorough and unmitigated data structure is a determinant in developing an efficient algorithm. Since it is directly associated with the way we handle newly arrived information and update old stored information.

A small and compact data structure which is efficient in inserting, retrieving and updating information is most favorable when developing an algorithm to mine association rules for stream data.

2.4 CLOSELY RELATED WORK

2.4.1 Data Stream Clustering*:* Clustering has been a widely studied task in the data mining literature. However, it is more difficult to adopt illogical clustering algorithms to data streams because of one-pass constraints on the data set. The main objective of the clustering is to minimize the average distance from data points to their closest cluster centers. An algorithm that makes a single pass over the data stream and uses small space was proposed by (Guha *et a*l 2000; 2003). It uses a partitioning based approach on the entire data set for adaptation of the K-means algorithm by creating a cluster over the entire data stream. As a form to improve the proposed algorithm by Guha *et a*l 2000; 2003, Babcock *et al.,* 2002 also put forward an exponential histogram (EH) data structure to support it. The center of the attention for manyresearchers has

been the k-median enigma, which was originally posed by Weber (1993). In order to increase approximation factors in the Guha et al 2003 algorithm, Charikar *et al* 2003 proposed k-median algorithm that overcomes the problem. Dominos *et al* (2000; 2001) in a research proposed an algorithm that captured the attention of many scientists is the k-means clustering algorithm.

This algorithm scales-up machine learning algorithm known as Very Fast Machine Learning VFML which has been equally applied to K-means clustering VFKM and decision tree classification techniques known as Very fast decision tree (VFDT). However, in data streams clustering the need for online techniques that can cluster data points incrementally arises. Ordonez (2003) proposed an enhanced incremental k-means algorithm for clustering binary data streams. In order to improve incremental learning and high- q u a l i t y data stream clustering, O'Challaghan et al.
(2002) further proposed STREAM and LOCALSEARCH algorithm. Aggarwal (2014) develop an efficient and effective approach for mining fast evolving data streams "CluStream" which integrates the micro- clustering technique in conjunction with a pyramidal time window with high-level
data mining process and also discovers data evolution regularities. This technique can be used to cluster different kinds of data streams, as well as create a classifier for the data in question. The method has clear advantages over recent techniques which try to cluster the whole stream at one time rather than viewing the stream as a changing process over time.

To further the course of CluStream, Aggarwa *el al.,*(2004) later introduced the HPStream; a projected clustering for high dimensional data streams, which eventually outperformed CluStream.

The Fig 2.1 below shows a typical clustering process.

Function Clusterin
Begin

Define number of cluster centers Set the

initial cluster centers Repeat

For each input data

For each cluster, compute;

Distance between the input data and its center

Choose the closest cluster center as the winner

Update the cluster centers

End for

End for

Until (Convergence or maximum number of iterations is reached) End

Fig 2.1: A typical Clustering Process

2.4.2 Data Stream Classification

A study which distinguished existing classification streaming data algorithms with infinite data flow and concept-drift was divided into single classifier and ensemble classifier approaches. The single classifier approach extends traditional data mining techniques to handle stream data. Domingos and Hulten (2000) developed the Very Fast Decision Tree (VFDT) system which is based on the Hoeffding bound principle. It splits the tree using the current best attribute taking into consideration that the number of examples used satisfies the Hoeffding bound. Such a technique has the property that its

output is (asymptotically) nearly identical to that of a conventional learner. VFDT as an algorithm addresses the issues of limited memory, effectiveness and accuracy. Liu, Li & Zhang (2009) also presents concept- Adapting Very Fast Decision Tree (CVFDT) algorithm aimed to dealing with high-speed data evolving data streams and time varying data thereby proposing an Adaptive decision-tree learners' solution for streaming data that can incrementally learn from incoming examples and further expand rules over time. CVFDT runs VFDT over fixed sliding windows in order to have the most updated classifier. The change occurs when the splitting criteria changes significantly among the input attributes. Gama & Kosina, (2012) therefore enhanced decision making by introducing an Adaptive Very Fast Decision Rules (AVFDR) in order to handle changing data known as concept drift.

A prior work on data stream suggested that ensemble approaches which trains and updates base level classifier on successive chunk of data perform quite well. Street & Kim (2001) introduced the ensemble-based model algorithm for stream data mining which depicted the Stream ensemble algorithm (SEA) by building separate classifiers on sequential chunk of training points. If the stream analyzed has a relatively slow and steady flow, then one can train based-like classifiers on each chunk in succession as it fills up with data. A suggestion by Han, Giraud-Carrier, & Li, Apple Intel (2015) enumerates that it is far more efficient to train all base classifiers in parallel. Therefore, a parallel ensemble learning approach called UELM-MapReduce was proposed to classify high speed streams of data.

2.5 DATA MODELS

In recent times, database and data mining communities have continually focused on modern models of stream data (where data arrives in the form of continuous streams). When weighed against data in traditional databases, data streams as said earlier are unbounded and the number of prospective transactions increase over time (Haiqing & Lang, 2017). Because of this, different data models for effective processing of stream data have been suggested in different mining algorithms. The initial model recommended was based on an incremental technique. In this model, all data from inception to the current time are considered in an equal fashion. Therefore, frequent item sets are defined on the cumulative data where new data are repeatedly added as time grows. The other model is based on a sliding window. That is, despite the infinite arrival of the stream data, the frequent itemset are derived based on the most recent data that is being captured within a stipulated sliding window where the present time signifies end point of that window. One justification for such a sliding-window model is that due to temporal locality, the data in streams is bound to change with time, and many a times people are interested in the most recent patterns from the stream data. The final model falls in between the two aforementioned data models while all data are considered in frequent item set mining, they are weighted differently according to a predefined weighting function. A very commonly used weighting function is the exponentially decaying function.

The first issue addresses which parts of data streams are selected to apply association rule mining. From the definition given in Section 2.2, data streams consist of an ordered sequence of items. Each incoming set of items is seen as a “transaction”. The main

question associated with data processing model is to derive a way to extract transactions for association rule mining from the overall data streams. Due to the continuous and unbounded nature of the data streams, there is a likelihood of the transactions been extracted to change over time.

Research by Zhu and Shasha [Zhu, 2002] presented three stream data processing models, which are;

i. Landmark

ii. Damped and

iii. Sliding Windows.

The Landmark model mines all frequent itemsets in relation to the entire history of incoming data stream from a specified point in time. [Charikar, 2004, Cormode, 2003, Jin, 2003, Karp, 2003, Li, 2004, Manku, 2002, Yang, 2004, Yu, 2004] all have done extensive researches on this particular model. Despite this, this model is not appropriate for applications and domains where people are more focused only in the most recent information of the data streams. A typical example is in the stock monitoring systems, where current and real-time information and results will be more meaningful to the end users than previous data.

The Damped model is also referred to as the Time-Fading model. This mines frequent itemsets in stream data streams such that each transaction has a weight and it decreases with age. This model considers different weights for new and old transactions. This is widely suitable for applications in which old data predominantly affects the mining outcome.

The Sliding Windows model finds and maintains frequent itemsets in sliding windows. Only part of the data streams within the sliding window are stored and processed at the time when the data flows in. In Chang, (2004), Chi, (2004), Lin, (2005), the authors use this concept in their algorithms to get the frequent itemsets of data streams within the

current sliding window. The size of the sliding window may be decided according to applications and system resources. The mining result of the sliding window method totally depends on recently generated transactions in the range of the window; all the transactions in the window need to be maintained in order to remove their effects on the current mining results when they are out of range of the sliding window.

All these three models have been used in current research on data streams mining. Choosing which kind of data process models to use largely depends on application needs. An algorithm based on the Landmark model can be converted to that using the Damped model by adding a decay function on the upcoming data streams. It can also be converted to that using Sliding Windows by keeping track of and processing data within a specified sliding window.

2.5.1 Summary

Data stream being a continuous and changing sequence of data that constantly arrive at a system needs to be processed in near real-time (Chaudhry, Show, & Abdelgurefi, 2005). The dissemination of data stream phenomenon has necessitated the development of diverse stream mining algorithms. Several studies (Babcock, Babu, Datar, Motwani, & Widom, 2002; Ganti, Gehrke, & Ramakrishnan, 2002; Mata & Ramesh, 2011) emphasized on different approaches proposed to overcome the challenge of storing and processing of fast continuous and uninterrupted streams of data. Traditional OLAP and most data mining techniques require multiple scans on a timestamp of data and cannot be viable or doable for stream data applications as they require a single scan of the data (Han, & Kamber, 2006). As the number of applications on mining data streams grow rapidly, there is an increasing need to perform analytics and data mining on streams of data. From a near exhaustive review

of literature in stream data mining and machine learning, very few studies were found to have employed compact tree-based stream mining on real life dataset. Few researchers however have employed various algorithms both supervised and unsupervised algorithms. While this represents the evidence of novelty in that regard, the proposed methods do not take into consideration the temporal locality issues involved with stream data. This study builds on existing stream mining approaches by addressing the issue of resource awareness involved in developing a compact dynamic tree for finding frequent itemset.

2.6 MINING SCHEME

Using comparably similar approaches employed in data stream mining, there are two general strategies for taking machine learning concepts and applying them to data streams. The *wrapper approach* which aims at maximum reuse of existing schemes, and *adaptation approach* which looks for new methods tailored to the data stream scenery.

Coherent to this research work, the wrapper approach which involves collected of known samples into a batch with the intent to induce a traditional batch learner model. The model(s) required to then be chosen and combined to some degree to form predictions. The impediment of this approach specifically is defining the appropriate training set sizes, and also that training times will be out of the control of a wrapper algorithm, other than the indirect influence of adjusting the training set size. When wrapping around complex batch learners, training sets that are too large could stall the learning process and prevent the stream from being processed at an acceptable speed. Training sets that are too small will induce models that are poor at generalizing to new

examples. Memory management of a wrapper scheme can only be conducted on a per-model basis, where memory can be freed by forgetting some of the models that were previously induced

2.7 SCALABILITY ISSUES IN FREQUENT PATTERN

Stream data continuously flow with great speed and they are very large in size (Al-Khateeb, Masud, Khan and Thurasisingham, 2012). This agrees to the characteristics of big data. Big data is a data whose scale, diversity and complexity requires new architecture, techniques, algorithms and analytics to manage it and extract value and hidden knowledge from it. Hence, big data researchers are looking for tools to analyse, manage, summarize, visualize and discover knowledge from the collected data in a timely manner and scalable fashion. Adding heterogeneous dataset to the system, represented as transactions could provide the basis for discovering interesting structural patterns and anomalies, which may alert an organization to the potential market opportunities.

In other words, frequent patterns are majorly associated with recurrent itemsets which are predominant in a given or specified transaction. A transaction on the other hand is a set of distinct items (symbols). Association rule mining finds frequent itemsets which are satisfying minimum support threshold value, based on that strong association rules is generated. The associations rule generates set of rules which satisfy user defined threshold value and Based on that one can develop marketing strategies. However, while this approach demonstrated its effectiveness in a variety of domains the issue of scalability has limited this approach when dealing with domains containing millions

and large expanse of incoming data. In addition, many rules that of interest are dynamic i.e. they change as data stream arrive over time. This further complicates the analysis because a static rule cannot be analyzed; snapshots would need to be analyzed over a period of time. Moreover, the streaming instance offer an opportunity for methods that can update the current set of patterns based on the changes to the incoming data rather than repeated analyses on the whole incoming data. Frequent pattern mining was first proposed by Agrawal et al., (1993) for market basket analysis in the form of association rule mining. It analyses customer buying habits by finding associations between the different items that customers place in their "shopping baskets". Since the first proposal of this new data mining task and its associated efficient mining algorithms, there have been hundreds of follow-up research publications, on various kinds of extensions and applications, ranging from scalable data mining methodologies, to handling a wide diversity of data types, various e x t e n d e d mining tasks, and a variety of new applications. With over a decade of substantial and fruitful research, it is time to perform an overview of this flourishing field and examine what more to be done in order to turn this technology a cornerstone approach in data mining applications

2.8 EFFICIENT AND SCALABLE METHODS FOR MINING FREQUENT PATTERNS

The concept of frequent itemset was first introduced for mining transaction databases (Agrawal et al. 1993). Let I = {i*1*, i*2*,...,i*n*} be a set of all items. A k-itemset α, which consists of k items from I, is frequent if α occurs in a transaction database D no lower than θ|D| times, where θ is a user-specified minimum support threshold (called min_sup in our text), and |D| is the total number of transactions in D. This article

introduces three main basic frequent itemset mining algorithms namely: Apriori, FP-growth and Eclat, coupled with their various extensions.

2.8.1 Apriori Principles and algorithm

Since there are usually a large number of distinct single items in a typical transaction database, and their combinations may form a very huge number of itemsets, it is challenging to develop scalable methods for mining frequent itemsets in a large transaction database. Agrawal and Srikant (1994) observed an interesting downward closure property, called Apriori, among frequent *k*-itemsets; a k-itemset is frequent only if all of its sub-itemsets are frequent. This implies that frequent itemsets can be mined by first scanning the database to find the frequent 1st-itemsets, then using the frequent 1st-itemsets to generate candidate frequent 2nd-itemsets, and further checks against the database to obtain the frequent 2nd-itemsets. Apriori *TID* generates candidate itemset before database is scanned with the help of Apriori-generating function. Database is scanned only first time to count support, rather than scanning the database it scans the candidate itemset. This variation of Apriori performs well at higher level where as the conventional Apriori performs better at lower levels.

Apriori Hybrid on the other hand is a combination of both the Apriori and Apriori *TID*. It uses apriori *TID* in later passes of database as it surpasses at high levels and Apriori in first few passes of database. This process iterates until no more frequent k-itemsets can be generated. This is the essence of the Apriori algorithm (Agrawal and Srikant, 1994). Few of its extensions include sampling approach proposed by Toivonen, 1996; incremental mining (Cheung et al, 1996) and integrated mining with relational database systems (Sarawagi et al, 1998). The resulting extensions were put together so as to further reduce the number of scans made over a database.

2.8.2 Equivalence Class Transformation (ECLAT)

ECLAT algorithm uses vertical database format whereas in Apriori horizontal data format (TranscationId, Items) has been used, in which transaction ids are explicitly listed. While in vertical data format such as ECLAT (Items, TransactionId) Items with their list of transactions duly maintained. ECLAT algorithm with set intersection property uses depth-first search algorithm (Borgelt, 2003). In first scan of the database, a TID (TransactionId) list is maintained for each single item. k+1 Itemset can be generated from k Itemset using apriori property and depth first search computation together. This process continues, until no candidate Itemset can be found. One advantage of ECLAT algorithm is that to count the support of k+1 large Itemset there is no need to scan the database; it is because support count information can be obtained from k Itemsets. This algorithm avoids the overhead of generating all the subsets of a transaction and checking them against the candidate hash tree during support counting (Srinivasan *et al*.,1997).

2.8.3 Structured data

In some challenging applications of data mining, data better described by sequences (for example DNA data), trees (XML documents), and graphs (chemical components). Tree mining in particular is an important field of research (Bifet & Gavaldà,(2008)). XML patterns are tree patterns, and XML is becoming a standard for information representation and exchange over the Internet; the amount of XML data is growing, and it will soon constitute one of the largest collections of human knowledge.

There are two limitations of this algorithm:

a) Complex candidate itemset generation process which consumes large memory and enormous execution time.

b) The excessive database scans for candidate generation

2.8.4 Frequent Pattern (FP) Growth Algorithm

In the field of data mining, the most popular algorithm used for pattern discovery is FP Growth algorithm. To deal with the two main drawbacks of Apriori algorithm in Jian & Jiawei 2000; a novel, compressed data structure named as FPtree is constructed, which is prefix-tree structure storing quantifiable information about frequent patterns. Based on FP tree a frequent pattern growth algorithm was developed.

FP Tree in many cases involves a two-step approach;

i. The foremost approach when constructing the frequent pattern tree is scanning the database or repository involved twice. The first pass of the database scans the data, calculate and registers a support count for each item, infrequent patterns are deleted from the list and the remaining patterns are sorted and arranged in descending order.

ii. In second pass, FP Tree is built using FP growth algorithm through which frequent patterns are extracted from Tree.

Conditional FP tree base and Conditional FP tree are constructed based on the properties associated to the *node link* as well as the *prefix path*. The Conditional FP tree is constructed for the frequent items of pattern base. Once Conditional FP tree is constructed, frequent patterns needs to be extracted. The application of FP growth

algorithm is beneficial compared to other algorithms in accomplishing three significant objectives;

i) The database is scanned only twice, and the computational cost is decreased dramatically.

ii) Secondly, no candidates' itemset are generated.

iii) Thirdly, the divide and conquer approach is utilized which subsequently reduces the search space.

The major drawback associated with FP growth algorithm is the inability to employ incremental mining. As the database is updated with new transactions, FP tree needs to be equally updated and the process needs to be duplicated.

Various other algorithms have been proposed to discover frequent patterns in incremental databases and to maintain association rules in dynamically updated databases (Chang & Yang, 2003; Koh & Shieh, 2004; Li, Deng &Tang 2006; Hong, Lin & Wu 2008). Many of which are strongly dependent on adjustments made on FP tree structure which are tree based. This implies that each proposed approach still requires two database scans for both the construction of the FP-tree structure and incremented section so as to update the tree structure). Leung *et al* 2007 proposed the CanTree which analyses the contents of a transaction database with only a single-pass and stores them as a prefix-tree in a canonical order. It also imbibes the FP-growth which is based on the divide-and-conquer approach to discern the frequent patterns. The simple tree construction process of CanTree enables it to outperform other algorithms based on the FP growth in incremental and interactive mining. However, since it is not similar to the FP-tree in compactness, the mining phase takes more time than the FP- tree approach when the number of frequent patterns in a database is reasonably large. FP-tree from extensive research has been seen to be highly compact

tree structure that enables decidedly efficient mining. In spite of this, it only handles the frequent items in a database, and it is a two-pass solution. Contrariwise, CanTree offers a single-pass solution which maintains a comprehensive amount of information in the database suitable for incremental and interactive mining. However, it incurs very high mining time due to the canonical and unquestioned order of its tree structure.

Chris & Han, 2004 proposed a paper 'Mining frequent Patterns in Data Stream at Multiple Time Granularity' using FP Stream to extract the complete set of frequent patterns with an appropriate error bound. This algorithm used the tilted time window model to extract the frequency itemsets. To guarantee the completeness of the frequent pattern, the less frequent ones are deleted right after the batch of that transaction has been processed. Due to the limitation in memory size, the entire incoming stream of data cannot be stored. As a result, FP stream splits the data into three stages: Frequent, Sub-Frequent and Sparse. The FP-stream structure consists of an in-memory frequent pattern tree with tilted-time window embedded.

In the year 2009, Pauray S.M Tsai proposed weighted sliding window (WSW) technique. This model allows users to specify various parameter for mining like size of window, weight of window and number of windows. In each window, every transaction has weight and if the weight satisfies minimum weighted threshold value then it is considered as frequent itemset. For large window size, execution time of this model decreases. This happens because for the small window size, number of frequent itemset is small due to small number of transactions. This model uses Apriori algorithm for candidate generation and this may take more memory and time. Therefore, instead of Apriori, we can use another algorithm like 'eclat' to improve mining. In the year 2011, Jing, Peng, Jianlong and Li proposed new algorithm called as Hybrid streaming, H-stream for short. They use H tree for storing and maintaining historical and potential

frequent itemset. It is used for collaborative and comparative frequent pattern mining. The main advantage of this algorithm is that it can efficiently mine frequent pattern from multiple streams.

The Landmark computational model was used to mine and stores most frequent items and their counts. It gave accurate result but for these accurate results, additional cost is needed. This algorithm takes 2 scans to generate exact result items set (Karp & Shenker, 2003). The main improvement of this algorithm is low memory usage. This algorithm discards infrequent items and their information Infrequent items may become frequent in future therefore it needs to be stored It requires 2 passes to generate exact result so one can improve this by single scan. In additional it gives no guarantees regarding false positive.

2.9 CONCLUSION

This section summarizes all data stream mining algorithms of all types. In relation to clustering algorithms, HPStream [31] is a projection-based clustering algorithm which models a high scalability. It incrementally updates and is efficient for high spatial datasets. Notwithstanding, it is decidedly complicated. CluStream [28] follows a micro clustering approach in addition to the concepts of pyramidal time frame. It is time and space efficient, can detect temporal changes in data, highly detailed and precise in nature. The main disadvantage associated to CluStream is that it predominantly supports only offline clustering. Search and Locale Search [24] algorithms are KMedians that make use of incremental learning. It is much faster but has low clustering quality and accuracy. VFKM (Domingos P., Hutten G. (2002)) is a K-Means algorithm which is faster and utilises minimal memory storage. Nonetheless, it requires multiple passes over the dataset to complete its processing. D-Stream by Chen & Tu

(2007) is a density-based clustering algorithm which exhibits high quality and efficiency. It can detect concept drifts in real time. However, it is highly complex in nature.

For classification Gaber, Krishnaswamy & Zaslavsky (2006) in their study of LWClass, it is a classification-based algorithm which is based on class weights. It exhibits high speed and consumes less memory. Its drawbacks are time consuming and can't be adapted to concept drifts. Ondemand stream classification Muthukrishnan (2003) uses micro-clustering approach. It exhibits dynamic updates, high speed, and consumes less memory space. VFDT and CVFDT very effective algorithms that produce decision trees. They are high speed and consume less memory space. However, they can't be adapted to streaming data, and they are time consuming and costly to study. GEMM and FOCUS by Ganti, Gehrke & Ramakrishnan (2002) are meant for generating decision trees and frequent item sets respectively. They are also associated with incremental mining approach and can likewise detect concept drifts. However, they are very time consuming and also costly study.

Ensemble-based classification uses combination of various classifiers algorithms. Its qualities include a single pass, dynamic update, ability to detect drifts, and hig level of accuracy. However, it has low speed, minimal storage, and consumes a lot of time. Law & Zaniolo, (2005) in the study of ANNCAD used incremental classification and exhibits a dynamic update. Its drawbacks are same as that of Ensemble-based classification. SCALLOP is suitable for scalable classification of numerical data streams. It exhibits dynamic update. It also suffers drawbacks same as that of ANNCAD and Ensemble-based classification. With regards to techniques related to frequent patterns and time series analysis, approximate frequent counts generate frequent item.

It exhibits an incremental update, simplicity, consume less memory, and complete processing in a single pass. However, it generates approximate output with more error range possibility. FPStream also generates frequent item sets. The major objective of this article is to analyze and clarify the various data mining techniques and data stream mining challenges in real time applications. The data mining techniques that act on data streams are classified into clustering, classification, frequency counting and time series analysis. A survey on these techniques reveal the facts that from the classification techniques VFDT, CVFDT, CDM, on demand stream classification, ensemble-based classification, and ANNCAD are applicable and feasible for mining data streams while GEMM, FOCUS, OLIN, SCALLOP are not feasible; with respect to clustering techniques VFKM, CluStream, AWSOM, and HPStream are applicable while stream and locale search, and D-Stream are partially feasible; with respect to time series analysis, FPStream and applicable to mining data streams. Finally, we are concluding that due to unique characteristics of data streams, a considerable extensive research will still need to be carried out.

CHAPTER THREE

3.0 PRELIMINARIES

The algorithm we are going to describe act on massive data that arrive readily and that cannot usually be stored most of the time. This algorithm works in few passes over the data and use limited space (less than linear in the input size).

This research work, therefore, conceptualizes the handling of enormous data as a stream mining problem that applies to continuous data stream and proposes an ensemble of unsupervised learning methods for efficiently detecting anomalies in stream data.

Finding frequent patterns from data streams is important and a challenging problem, since capturing the stream content memory efficiently with a single-pass and efficient mining have been major issues. The FP-growth mining technique is one of the efficient algorithms where the achieved performance gain is mainly based on the highly compact frequency-descending FP-tree structure that ensures the tree to maintain as much prefix sharing as possible. However, the two database scans and prior threshold knowledge requirements of the FP-tree restrict its use in data stream. DSTree uses the FP- growth mining technique to mine exact set of recent frequent patterns from stream data with a single-pass. However, it provides poor compactness in tree structure and inefficient mining phase, since it uses frequency-independent canonical order tree structure. Therefore, in this paper, we propose a novel tree structure, called CPS-tree (Compact Pattern Stream tree), that constructs an FP- tree like compact prefix-tree structure with a single-pass over stream data and provide the same mining performance as the FP- growth technique through the efficient tree restructuring process.

In this chapter, we will analyse the Dynamic Compact Stream Pattern Algorithm in the context of mining from a data stream using a Tree-base structure. This algorithm makes use of two different models:

i) Dynamic stream tree model.

ii) and Compact pattern stream model.

The Dynamic Compact Stream Patter Algorithm approach has proved much advantageous on compactness, space utilization reduction and time effectiveness. The Compact Pattern Stream introduces the concept of dynamic tree restructuring techniques in handling data stream at runtime and facilitate an efficient Frequent Patter data mining mode.

3.1 TREE-BASED MODEL ANALYSIS

The tree-based algorithm is based on set-enumeration concepts. The candidates can be explored with the use of a subgraph of the lattice of itemsets (see Fig. 3.0), which is also referred to as the lexicographic tree or enumeration tree. These terms will, therefore, be used interchangeably. Thus, the problem of frequent itemset generation is equivalent to that of constructing the enumeration tree. The tree can be grown in a wide variety of ways such as breadth-first or depth-first order. Because most of the discussion in this section will use this structure as a base for algorithmic development, this concept will be discussed in detail here. The main characteristic of tree-based algorithms is that the enumeration tree (or lexicographic tree) provides a certain order of exploration that can be extremely useful in many scenarios. It is assumed that a lexicographic ordering exists among the items in the database. This lexicographic ordering is essential for efficient set enumeration without repetition.

To indicate that an item *i* occurs lexicographically earlier than *j*, we will use the notation $i \leq_L j$. The lexicographic tree is an abstract representation of the large itemsets with respect to this ordering. The lexicographic tree is defined in the following way:

- A node exists in the tree corresponding to each large itemset. The root of the tree corresponds to the *null* itemset

- Let $I = \{i1, \ldots ik\}$ be a large itemset, where $i1, i2 \ldots ik$ are listed in lexicographic order. The parent of the node *I* is the itemset $\{i1, \ldots ik-1\}$. This definition of ancestral relationship naturally defines a tree structure on the nodes that is rooted at the *null* node.

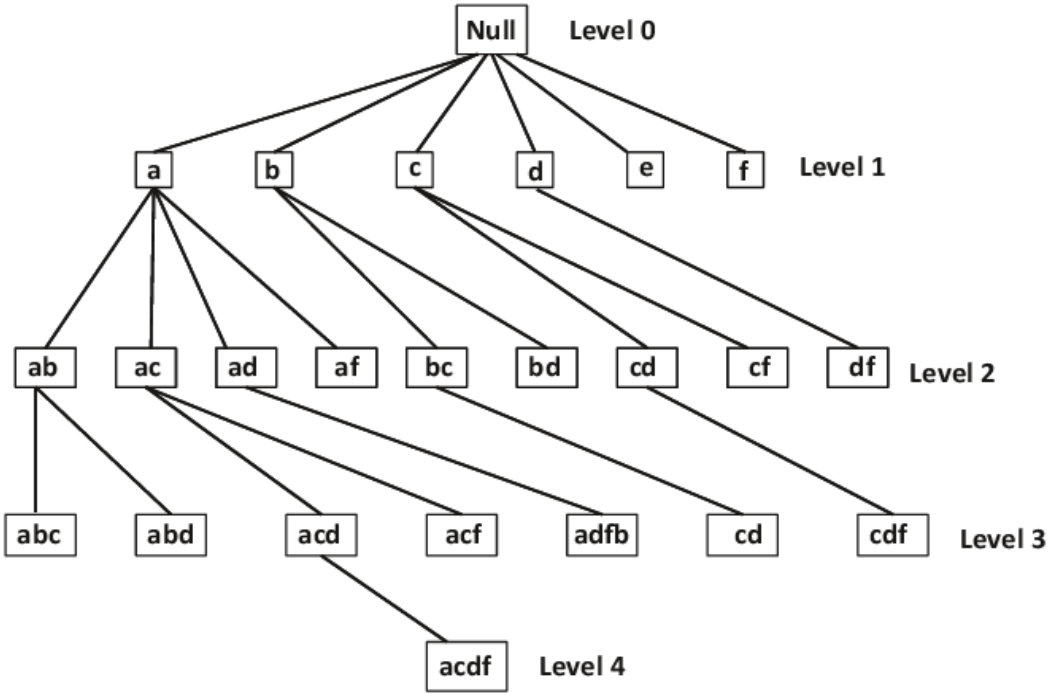


Fig 3.0: Lexicographical Pattern

It is important to point out that virtually all non-maximal and maximal algorithms, starting from *Apriori*, can be considered enumeration-tree methods.

3.2 COMPACT STREAM PATTERN TREE MODEL

CSP Tree helps achieves a frequency-descending structure by capturing a subset of data from a database or a series of dataset and through dynamic restructuring; it reorganizes itself using an efficient tree mechanism. Unlike its subsidiary which is the FP tree algorithm, to accelerate the tree transversal process it maintains an item list which can be referred to as the *I*-list. The construction process basically consists of two distinct parts, insertion and restructuring. The tree building initiates with an insertion phase and ends with a restructuring phase.

Notwithstanding, these two stages are repeatedly carried out several times in the development of the tree construction. As a result of the dynamic nature of the incessant streaming data, the transformation from insertion phase to the restructuring phase can be achieved after analyzing the data in the stream.

3.2.1 Insertion: This insertion phase is similar as that of FP tree algorithm which inserts data (transactions) into the CP Tree in a lexicographical order (*I*-list) and updates the frequency count of its respective items in that order.

The insertion phase captures the content of the incoming stream into the tree according to the current sort order of the I-list.

3.2.2 Restructuring: the restructuring phase rearranges the aforementioned *I*-list according to the frequency-descending order of the incoming dataset and restructures the tree nodes according to a new *I*-list. The use of an efficient tree restructuring mechanism is paramount to scale down the overall tree restructuring running cost. The reason for restructuring is to reposition the nodes of an already existing tree based on the *I*-list order.

To restructure the CSP Tree, we use an actual tree restructuring mechanism known as branch sorting technique and path adjusting method. Both of this technique are proposed to handle tree nodes with only one count and pane counter parameters.

3.3 BRANCH SORTING TECHNIQUE MODEL

In the process of performing the restructuring, the fundamental procedure to be carried out is to reorder the items in *I* in a frequency-descending order to obtain a sorted list. Let's say the sorted list is *Iclass,* therefore based on the sorted list, the tree *T* is restructured. BST is an array-based technique that restructures all branches one after the other starting from the root of the tree named *T.* Individual sub-tree concealed under each child of the *root* is treated as a branch. Therefore, *T* contains as many branches as the number of children it has under the *root*. There is a likelihood of each branch having several paths and several branching nodes, which implies that different node can have more than one child. BST sorts each path in the branch in relation to the sorted list (*Iclass*) by removing it from the entire tree and grouping it to form an adhoc array. The adhoc array in turn is inserted into the tree in a sorted order.

Ultimately, the Branch Sorting Technique aborts as soon as all the branches in *T* have been processed. This produces a final tree that has been sorted named T*class*. In most cases, during processing it is possible that a path in the tree *T* is found already arranged in the *Iclass*. Such path is referred to as *class path*, which implies as restructuring is taking place, any path found to be a *class path* will be skipped or omitted from the processing. The *status* of this information about the *class path* is passed to all branching nodes in the same path indicating that path from each branching node is up to the null root has been sorted.

It may be speculated that the Branch sorting technique will require a dual computation since it removes a path from the tree structure and inserts it into the tree in a sorted order. Instead it is rather less than one would expect for removing and inserting a transaction of the most recent window to and from the tree, thereby sorting out the all identical transactions in the current window within one application. Generally, BST helps to further reduce the cost of processing and the information carried in individual branching node in a sorted path may automatically cut down the time required only looking at the processing cost for the sub-path instead of the entire path.

Axiom 1: Let A = {i*1*, i*2*,..……, i*n*} be a path in a CSP Tree sorted in a canonical sort order

where i*n* is the current tail-node of the path. After restructuring *A*

If node i*j*, j [1, n - 1] symbolizes the new tail-item of A,

then i*k* becomes the new tail-node for the path that has been rearranged.

Based on the above discussion of the theorem, the figure below shows the sorting algorithm

```
Algorithm (Branch Sorting Technique)
Input: T and I
Output: Tclass and Iclass
Method:
        Begin
          Compute Iclass from I in freq-descending order using
          merge sort technique For each branch Bi in T
                for each unprocessed Aj in Bi //from the root up to
                    the tail-node If Aj is a sorted path

                        Process_Branch(Aj);
                    Else Sort_Path(Aj);
                    End If
                End for
                End for
        End for
```

Terminate when all the branches are sorted and output
Tclass and Iclass End

These two phases are implemented consecutively always starting with the insertion phase and finishing with the restructuring phase. This implies that the Restructuring phase is always executed after the insertion phase. The tree is refreshed each time as the window slides for a ready-to-mine platform with the exact information about frequent item set accompanied with rules is provided for the current window. In cases where a rule item is associated with multiple classes, only the class with the largest frequency is considered. Restructuring of the tree can be done using either Path Adjusting method. When all the frequent item set are obtained, using bottom-up FP-Growth mining technique, confidence of the various rules is calculated and is sorted in the memory.

3.4 SLIDING WINDOW

A sliding window algorithm places a buffer between the application program and the network data flow. For most applications, the buffer is typically in the operating system kernel, but this is more of an implementation detail than a hard-and-fast requirement. The sliding window technique inspects every time window at all scales and location over a time stamp which means our data will be classified according to the most recent item set provided. Typically, sliding window algorithms serve as a form of flow control for data transfers.

Datasets in a sliding window are often described in a structure

$$(Di) = \frac{\{S0,S1,\ldots.Si\}i \leq n-1}{\{Si,Si+1,\ldots.Sn+i-1\}i \geq n-1}$$

If the timespan (length, l) of a window is denoted with *n*

However; data at a point Ti in the window is denoted by

Where; Di: Dataset present in the sliding window

Si: Data values of the dataset at the point i

3.5 APPROACH TO CSP MODEL

The algorithm uses data passed to it to construct a CSP tree which is always in ready state to be mined. There are several techniques out there in building this kind of algorithm; most of them work by firstly constructing FP tree then restructuring the FP tree into CSP. At first, transactions in the data stream are inserted into the CSP-tree based on a predefined item order (e.g., lexicographical item order). This item order of the CSP-tree is maintained by a list, called the I-list, with the respective frequency count of each item. After inserting some transactions, if the item order of the I-list deviates significantly from the current frequency- descending item order, the CSP-tree is dynamically restructured by the current frequency- descending item order and the I-list updates the item order with the current one.

In contrast the technique used in this algorithm allows us direct building of frequency-descending item order list from data. This is achieved by using creating a track table that keeps track of the items which can stand for the *l*-list in other techniques and construct CSP tree from it, this can be shown in the figure 3.1 below. This will save the algorithm from iterating over tree nodes several times during the creation and modification of tree. In dealing with continuous data stream different systems has been used in dealing with movement of data; sliding window is used in this work where data are captured in pane and panes are housed in window, when new data enter

the stream the window slides removing some old panes while inserting new ones; depending on sliding size of the window.

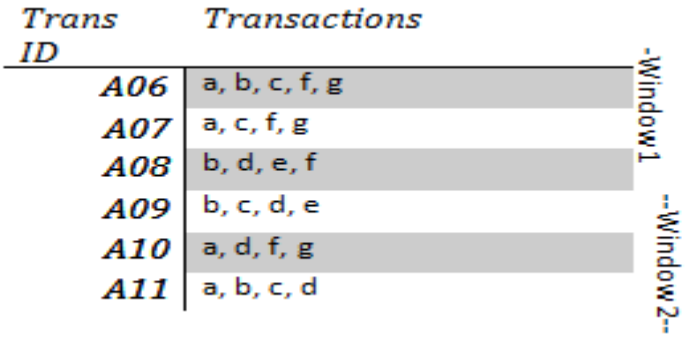


Figure 3.1: Transaction with Window

A stream of data of window *n* = size 2, pane size = 2

During sliding of the window, the data with transaction id from A06 to A09 are being processed in the first window, the reason for this is that a window is of size 2 which means a single window can have a minimum of two panes, and each pane in return holds two transactions, and each window therefore contains four transactions. The window slide is one, which makes the window to move one pane at a time, one pane contains two transactions. Therefore, window 2 contains transactions with id from A08 to A11.

On the first, the run algorithm does the following:

7. Format data from source
8. Creates an empty window
9. Creates an empty pane
10. Fills the pane with transaction data
11. Insert the pane into window
12. Repeats step three to five until the window is full
13. Construct CSP tree from the table

With the constructed tree, mining can occur. Due to the nature of our data where we are not dealing with discrete data but continuous, a function is created called refresh, this function can be triggered by any event and what it does are the following;

1. Check data source if there is change.
2. If there is no change the algorithm continues resting but if there is, the change will be added to algorithm data

3. Reconstruct the tree.

The figure 3.2 shows the architectural design of the data-item flow of the data stream mining. Each component in the figure represents the requirements and algorithm of the model.

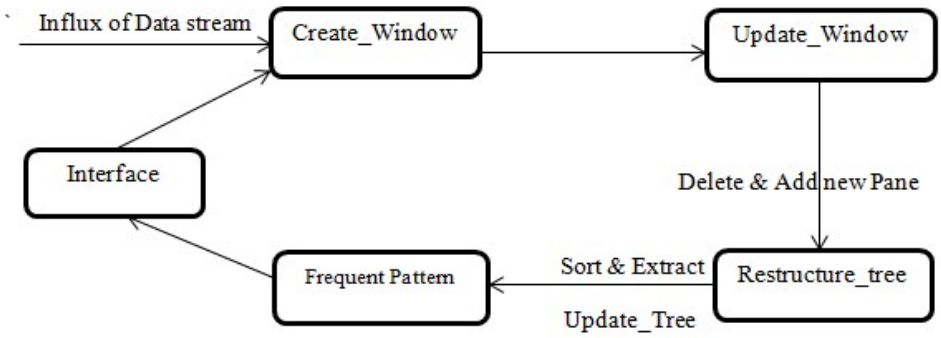


Figure 3.2 Data Flow Architecture

The system architecture invokes requests made by inserting the transactions from data stream with a single scan of the data. With the middleware already active, a successful connection should be established which makes easy passage of the data. The transaction is imported into the current sliding window with its respective support value, here we use sliding window in the next update window module which take the input from create window module and delete the old panes and add new panes to mine latest frequent patterns then updated window pass as input to restructure module which perform the extraction, sorting and reinsertion operation then pass the sorted, extracted data as input to final mine module which mine all the transactions over dynamic data streams and find the latest frequent patterns which are greater than

threshold value and finally display the set of latest frequent patterns over the dynamic data stream. The runtime rates of both algorithms are very important and vital in the analysis of this research work. We evaluate performance of the proposed CSP algorithm through extensive experiments. The algorithm is written in Python. In performance evaluation we consider the run time and memory usage for different threshold values.

3.6 DYNAMIC COMPACT STREAM PATTERN TREE CONSTRUCTION

A complete tree is nothing but an instance of a Node class with Null Node as it name but has no parent. Every CSP tree start with a Null Node which will be the parent of all nodes;

```
tree = Node ("Null Node", 1)
```

To create a node value and count of the node are mandatory properties that must be fulfilled before the node can be created. In this case we created a node with Null Node as its value and support count of 1, this is where everything begins. The next node created is going to be added to this node as its child:

```
def add_to_tree(self, items, in_tree, count):
```

```
        if items[0] in in_tree.children:
            in_tree.children[items[0]].inc(count)
        else:
            in_tree.children[items[0]] = Node(items[0], count, in_tree)
            if self.trackTable[items[0]][1] is None:  # update header table
                self.trackTable[items[0]][1] = in_tree.children[items[0]]
    else:
        self.update_track_table(self.trackTable[items[0]][1],in_tree.children[items[0]])
        if len(items) > 1:
            self.add_to_tree(items[1::], in_tree.children[items[0]], count)
```

The function above does the job of building a tree by creating a node from an item and adding it to the tree as child. It firstly checks whether the item already present in the tree is present. It will simply increase the support count of the node. Otherwise, new Node will be created as seen on line 5 of the code.

The rest of the code checks and update track table accordingly.

Formatting of data

The algorithm doesn't work on arbitrary data, it expects a dictionary with the item sets as the dictionary keys and the frequency as the value.

```
def create_init_set(data_set):
    ret_dict = {}
    for trans in data_set:
        if frozenset(trans) in ret_dict.keys():
            ret_dict[frozenset(trans)] += 1
        else:
```

```
ret_dict[frozenset(trans)] = 1

return ret_dict
```

This code snippet creates new dictionary and fills it with the transaction, it is the dictionary that will be supplied to the algorithm.

3.6.1 Creation of new window

Creation of new window requires;

- Size of window: maximum number of pane the window will contain
- Size of pane, and
- Sliding size

```
Self.window = Window (self.windowSize, self.paneSize, self.slideSize)
```

3.6.2 Creation of new pane

Creation of empty window is followed by creation pane once data is available and to create a pane the following parameters are required;

- Size of the pane: maximum number of transactions a pane can contain
- Age of the pane: minimum is maximum technique is used in considering how old a pane is and if it is ripe for popping off the window.

3.6.3 Filling of the pane with transaction

The code snippet above does the job of creation of new pane object and calling insert_transaction method of the object to push the transaction item into the pane. It is also noticeable that after all items in a transaction finished the putting, the pane is inserted into the window.

```
for trans in
self.frequentItem
:
    pane = Pane(self.paneSize, self.paneAgeCounter)
```

```
            self.paneAgeCounter += 1
        for item in trans:
        pane.insert_transaction(item)

    self.window.insert_pan
    e(pane)
```

3.7 SOLUTION MINING APPROACH

Mining or discovering frequent pattern item-sets consist of the first step of association rule mining process. Data mining is termed as the practice of spontaneously searching large stores of data to ascertain patterns, trends and development that go beyond simple analysis (oracle.com). It extracts hidden predictive information from datasets. Data mining uses sophisticated algorithms to wedge the data and evaluate the probability of future events. Data mining entails analyzing a set of persistent relations, set of well-defined operations, and highly optimized query processing and transaction management component which requires less frequent update and a snapshot of the database used for processing queries. However, the past two decades have witnessed several classes of application with additional set of requirements than those provided by traditional database management systems. A typical example is data warehouses developed for Online Analytical Processing (OLAP). OLAP requirements have resulted in new techniques for data consolidation from multiple sources of database (Chakravarthy & Jiang, 2009). As a result, data mining has successfully accommodated these advances by adding new capabilities to store and process structured, unstructured, and multi-media data types. In addition, knowledge discovery research has prompted mining of large data sets directly from a relational

DBMS using novel mining operators and the Structured Query Language (SQL). Organizations with substantial amount of data are specifically starved of valuable knowledge. Data mining tools could best help these organizations to extract hidden patterns of useful information (Vivek Bhambri, 2013).

In recent years, there has been emerging class of data intensive applications such as data processing, traffic monitoring, stock data, and telecom call records which need to process data at a high input-rate. These applications are different from traditional database management system applications where data is at rest with respect to; data arrival rate, update frequency, processing requirements and queries asked. Data mining which is synonymous to Knowledge Discovery in Data (KDD), has helped diverse business organizations in increasing tangible profit that can be measured in terms of amount of money, number of customers and customers loyalty. It entails analyzing a set of persistent relations, set of well-defined operations, and highly optimized query processing and transaction management component which requires less frequent update and a snapshot of the database is used for processing queries. Data mining consists of five major elements: Extract, transform, and load transaction data onto the data warehouse or management system; Store and manage the data in a multidimensional database system; Provide data access to business analysts and information technology professionals; Analyze the data by application software; Present the data in a useful format, such as a graph or table.

3.8 EXPERIMENTAL SETUP

The following sections are organized as follows: A comprehensive experimental setup on the CSP tree model, and real dataset from KDD-CUP 2000 competition . All

programs are written in Python 3.3 version and run on a time-sharing environment with Windows 7 Operating System on a 2.66 GHz machine and a RAM memory of 2 gigabyte (GB). Runtime includes tree construction, tree restructuring and mining time.
Data Preprocessing: In real-life scenario, stream data come from live observations or in some cases in the form of sensors which are subject to noise. This problem is usually handled by preprocessing the data. Which is why it is of great importance to design a light-weight preprocessing techniques that can guarantee quality of the mining results. Data pre- processing consumes most of the time in the knowledge discovery process. The challenge here is to automate such a process and integrate it with the mining techniques.

Data preprocessing in this research work involves the use of a middleware which serves as a a functioning hidden translation layer which enables a communication and interaction, and data management between the operating system and the application running on it. It can be referred to as plumbing as it connects tow applications together so that data can be easily passed between the pips created by the middleware. The use of a middleware makes it possible to perform requests such as formatting of data and or allowing the system to run custom functions based on the user's specified need or requirements.

Data representation: Stream data are essentially high dimensional data. Defining algorithms that work directly on the raw persistent data would be computationally a bit expensive. The main motivation of representation is thus to emphasize characteristics of data in a concise way.

3.8.1 Characteristics Of The Datasets

Some of the popularly used datasets are DARPA 1998 data set, DARPA 1999 data set and KDD Cup 1999 data set which are available in the MIT Lincoln Labs.

The dataset used in this study represents the KDD-CUP 2000 competition. It contains 77,512 sequences of click-stream data and comprises of 3340 distinct items. The average length of sequences is 4.62 items with a standard deviation of 6.07 items. The table 3.1 below provides a brief illustration of the general overview of the dataset characteristics.

Data	#Rows	#Trans	#Items	Trans field	Itemset	AvgTL	(AvgTL/I)x100
Sales	999	920	2	Transaction_date	Products	1.08	54.18
Telecom	3334	2093	6	Area code	State	1.59	26.53
Crime	7585	30	132	Cdatetime	Crimedescr	244.63	185.33
BMS 1	77512	77512	10	—	—	4.62	46.22
Call	10454890	566	305	Date	Service	1657.39	543.41

Table 3.2: Dataset Characteristics

- #Rows: contain the total number of rows in the dataset
- #Transactions: includes the number of translated transaction data
- #Items: are the numbers of translated item data For example in the Sales dataset, having just Product1 and Product2 as items which makes #items to be 2.

  Transaction Field: contains the selected column to guide mining, this is required for multi columns dataset. The dataset to be mined needs a strong guide upon which it

- can be easily mined and analyzed. For sales dataset Transaction_date was chosen. The date column is having date and time, if we use the date as it is the mining will be too narrowed or restricted, so as to have a wider view of the analysis a middleware is used to strip the time away so that we can analyse product that are purchased the same day instead of the same seconds. Items Field: This column
- contains the item needed to be mined.
- AvgTL: Average transaction length is the average lengths of each transaction, this is useful due to the fact that each transaction has varied length, the average will show center tendency of the transaction.
- (AvgTL/I) x 100: AvgTL divided by number of item multiply by 100, this value is useful, by bringing diversification set in each transaction to comparable scale.

3.8.2 Data Preprocessing

Data preprocessing is crucial aspect in the process of data mining. If data input to algorithm is not in proper format, then it cannot be processed efficiently. So pre-processing is needed and in which existing data transform into new data which is in proper format and suitable for processing. Different data mining tools available in the market have different formats for input which makes the user forced to transform the existing input dataset into the new format.

Mining Association rule involves quite a bit of memory and CPU costs. A noticeable drawback is that processing time is always limited to only one online scan. So there is need of real time maintenance and updating association rule. However, stream data if we update association rules too frequently, the cost of computation will increase drastically.

Most of the resources such as memory space and CPU in data stream mining are vital. One cannot ignore the resources availability, a typical example is when the main memory is totally used up in processing, data will be lost and it leads to errors or inaccuracies in results. These challenges need to be considered in order for our algorithm to perform at its utmost best.

Essentially, the processing of stream data requires two layers:

i.) a storage layer and

ii.) a processing layer.

The prerequisite of the storage layer is to provide and maintain a detailed prioritization and strong consistency to enable fast, inexpensive, and replayable reads and writes of substantial streams of data. The processing layer is responsible for continually using up data from the storage layer, running computations on that data, and then notifying the storage layer to delete data that is no longer needed or deleting the limited storage area for a new set of data to come in.

3.8.3 Data Processing Approach:

Data preprocessing in this research work involves the use of a middleware which serves as a functioning hidden translation layer which enables a communication and interaction, and data management between the operating system and the application running on it. It can be referred to as plumbing as it connects tow applications together so that data can be easily passed between the pips created by the middleware as shown in the figure 3.3 below. The use of a middleware makes it possible to perform requests such as formatting of data and or allowing the system to run custom functions based on the user's specified need or requirements.

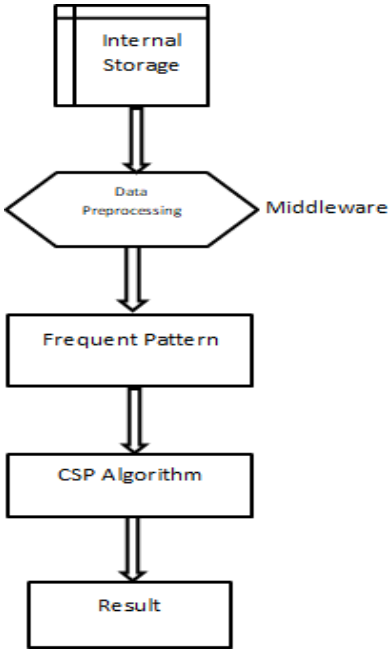


Figure 3.3: Framework for CSP Algorithm

The internal memory represents a temporary repository for the incoming dataset where it is passed to the data processing interface (middleware). This is where the incoming influx of data is converted to the computers understandable language for data analysis to be made possible. After this is done, an actionable result is passed to make inference.

Data preprocessing in this research work involves the use of a middleware which serves as a functioning hidden translation layer which enables a communication and interaction, and data management between the operating system and the application running on it. It can be referred to as plumbing as it connects two applications together so that data can be easily passed between the pipes created by the middleware. The use of a middleware makes it possible to perform requests such as

formatting of data and or allowing the system to run custom functions based on the user's specified need or requirements.

Data representation: Stream data are essentially high dimensional data. Defining algorithms that work directly on the raw persistent data would be computationally a bit expensive. The main motivation of representation is thus to emphasize characteristics of data in a concise way.

The middleware class is created to build skeleton of conversation between the algorithm and data it is meant to process. The constructor of this class require file name of the file that should be processed, the item column(s) that the algorithm should process and the transactional column the item should be matched against.

```
def _init_(self, file, transaction_field, item_field):
```

The middleware class has a function call *format_data* which is used to handle data return the algorithm specific data type. The middleware also declared an abstract method named *process_data* that must implemented by every subclass of the class.

Class

```
SalesMiddleware(Middlewar
e):
        def process_data(self):
                item_data =
                self.input_data[self.item_field]
                raw_trans_data =
                self.input_data[self.transaction_field]
                trans_data = []
```

```
    # We need only the date value we need to discard the time
            for i in range(len(raw_trans_data)):
trans_data.append(raw_trans_data[i].split()
[0])

    return item_data, trans_data
```

3.9 EXPERIMENTAL RESULTS

The algorithms for processing streams in one way or the other involves a form of summarization of the stream by making useful sample of the stream to eliminate most of the "undesirable" elements. The speed of the mining algorithm must be faster than the incoming data rate otherwise, summarization is of great effect. An algorithm can also run faster by processing less information, either by stopping early or storing less, thus having less data to process. The more time an algorithm has, the more likely it is that accuracy can be increased. The performance of an algorithm that operates on data streams is measured by three basic factors:

a. The number of passes the algorithm must make over the stream.

b. The available memory.

c. The running time of the algorithm

In addition, in the experiments, we follow the evaluation method used by many state-of-the-art Compact Stream Pattern mining algorithms. The actual class label of an incoming example becomes available immediately after its prediction.

3.10 RUNTIME ANALYSIS

Runtime **efficiency**: this represents the **measure** of amount of time needed for the CSP algorithm to execute each stream of data presented to it in comparison with that of the FPTree. CSP tree tends to mine faster as there is an influx dataset CSP to handle. This is a positive test case to know how effective CSP Tree could be over FP tree.

Total Runtime	*CSPTree*			*FPTree*			*Window Size*	*Pane Size*
	Creation	Restructure	Mining	Creation	Restructure	Mining		
	0.01738	2.95e -5	0.00026	0.01272	2.79e -5	4.19e -5	2	700
	0.02285	8.62e -6	0.00031	0.019948	8.62e -6	0.00025	3	700
	0.02567	9.44e -6	0.00043	0.02543	1.02591	0.00029	4	700
	0.02475	8.62e -6	0.00032	0.01855	9.03e -6	0.00026	5	700

Table 3.3: Telecoms dataset

table 3.3 above shows the Test runtime for the Telecoms dataset which comprises of the tree creation, tree ructuring and how long it takes to mine a given instance of the telecoms data present in the pane of a window.

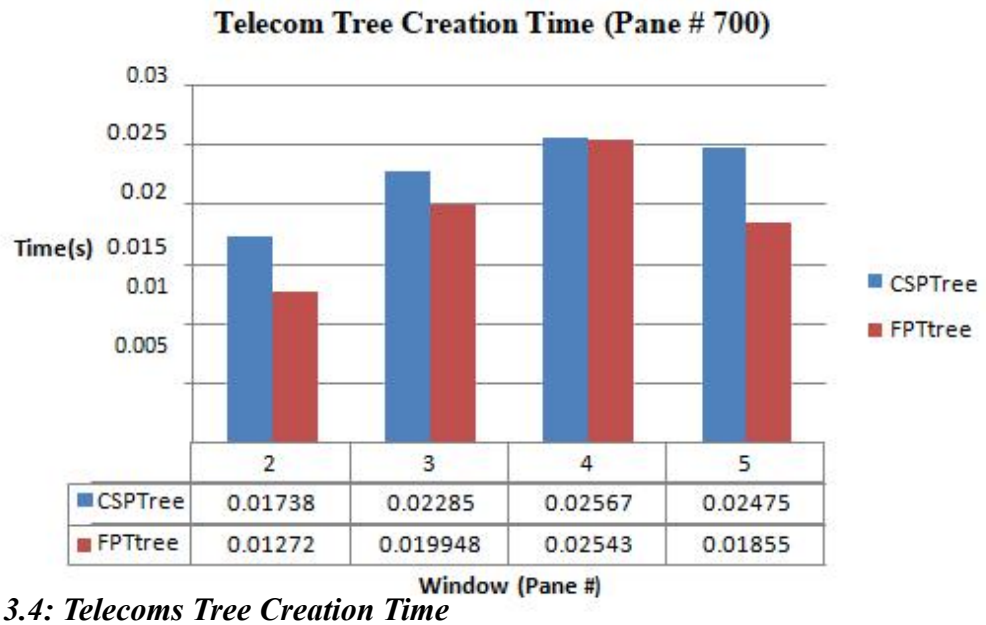


Fig 3.4: Telecoms Tree Creation Time

The fig. 3.4 above shows the tree creation time for the crime dataset between CSP Tree and FP tree. The resulting come explains that FP Tree in the first window has a faster tree creation rate compared to CSP Tree. And at each sequent window the time it takes to create the tree gradually lowers till the necessary patterns are derived.

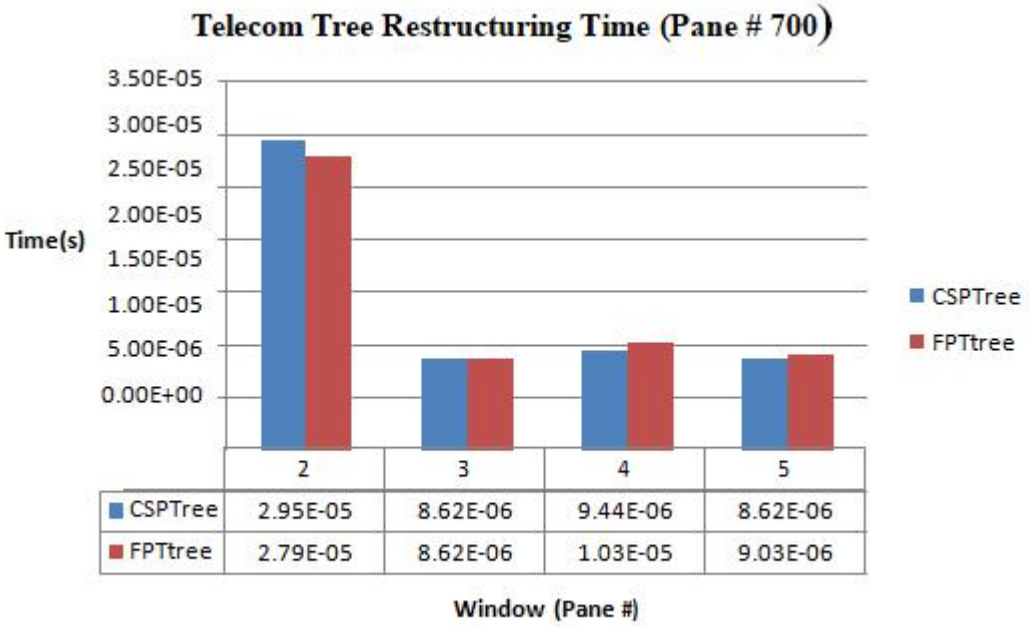
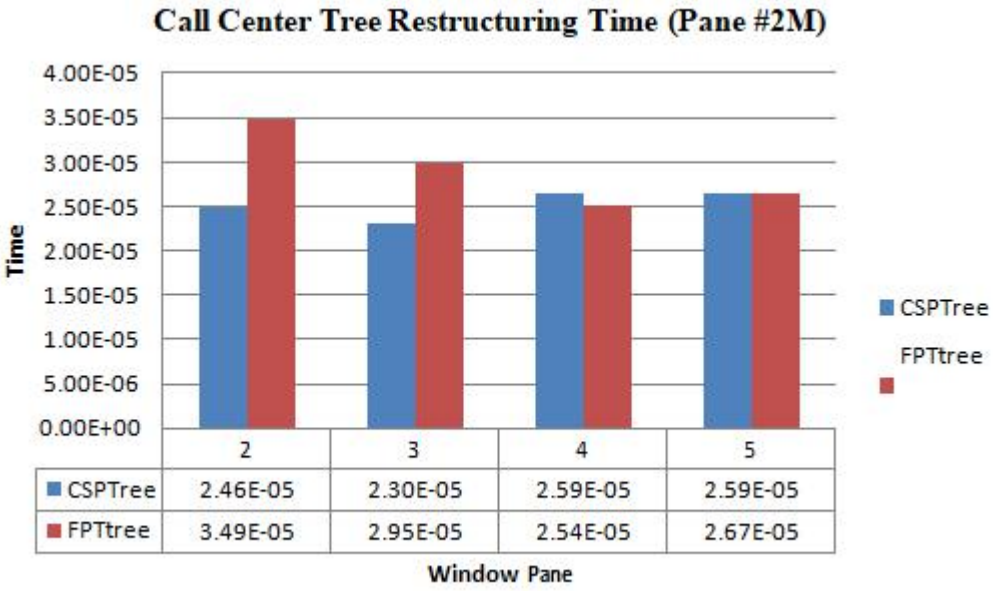


Fig 3.5: Telecoms Tree Restructuring Time

The figure 3.5 above shows the time it takes to restructure each incoming tree for the two algorithms. In the first dow, FP tree restructures faster but in subsequent windows CSP Tree outperforms it.



	2	3	4	5
CSPTree	2.46E-05	2.30E-05	2.59E-05	2.59E-05
FPTtree	3.49E-05	2.95E-05	2.54E-05	2.67E-05

Fig 3.6: Call Center Tree Restructuring Time

fig 3.6 above shows the tree creation time for the Call care dataset between CSP Tree and FP tree. The resulting outcome demonstrates and confirms that CSP Tree in the entire window has a faster rate of tree creation en compared to CSP Tree.

3.11 Summary

We have analysed the CSP-tree that dynamically processes a continuous and and unbounded data-stream using a frequency-descending prefix tree structure with a single-pass by applying tree restructuring technique. This considerably reduces the mining time. We looked at the Branch sorting method which is a new tree restructuring

technique; and presented guideline in choosing the values for tree restructuring parameters. We have shown that despite the additional overhead cost of the tree restructuring, CSP-tree achieves a remarkable performance gain on overall runtime. Moreover, the easy-to-maintain feature and property of constantly summarize full data stream information in a highly compact fashion facilitate its efficient applicability in interactive, incremental and stream data.

In approaching stream data mining, solutions can be categorized as data-driven solution and task- driven solution. The former analyses only a subset of the whole data set or totally alters the data to a summarized data size. Such an approach allows us to utilize many known data mining techniques to the case of data streams, while the latter which is the Task-based techniques methods modifies existing techniques or invent new ones in order to address the computational challenges of data stream processing. For the purpose of this research, the two approaches will be utilized in achieving an efficient algorithm.

CHAPTER FOUR

4.0 IMPLEMENTATION AND DISCUSSION

We have presented several methods for mining frequent patterns for both synthetic dataset and real application dataset. To undertake a comprehensive study of stream data, including the application of strategies and to acquire in-depth knowledge of the data streams, various previous works on data streams and the potential for improving performance were reviewed. In addition, the existing mining algorithm framework with specific consideration of fast mining techniques were reviewed in order to understand the limitations of current algorithms that have been addressed in later part of this research work.

A characterization of stream mining in real-time will be extensively examined using the sliding window technique by making use of the most recent dataset to anticipate incongruity, oddity and outliers. Thereafter, CSP Algorithm will be employed as a means to find a linear optimal hyperplane in order for the margin between positive and negative cases is maximized as the base algorithm and iteratively improving it using sorting techniques. Also, CSP tree will be used in detecting frequent patterns from data stream with the aid of Path Adjustment methods and the Branch sorting methods. It does this by performing insertions which constitute the attachment of incoming flow of data and dynamic restructuring of the new dataset. The algorithm basically involves insertion and restructuring. The proposed algorithm will be tested on telecoms data, BSM data, Crime rate datasets Sales Dataset and Call care so as to ascertain its efficacy. The dataset will be used in such a way as to mimic stream data in an effort to evaluate the effectiveness of the proposed algorithm in handling feature evolution, ability to

predict in real-time and concept shift inherent in stream data. An implementation of the resultant algorithm will be done using python programming language on Windows Operating system as the platform.

4.1 DYNAMIC COMPACT STREAM PATTERN DATA MINING

Data Stream mining refers to informational structure extraction as models and patterns from continuous data streams. Information is imperative to make wise and actionable decisions. This is quintessentially true in real life but also in computing and especially critical in several areas, such as finance, fraud detection, business intelligence or battlefield operation. Information flows in from different sources in the form of messages or events, giving a hint on the state at a given time. While data mining usually operates directly on the complete, stored data, stream data tends to make use of supporting application models and data models because of the transient nature of the data to be processed. Mining Streams of data is a process of being proactive about data thereby avoiding an eventual reactive phase of letting too much data go by, which might eventually lead to a total lose or devaluation of the actions initiated for processing. Pertinent to diverse inchoate incipient domains, nonetheless, the inflow of data and the need for it to be handled and processed on an unremitting basis is vital; with the benefit of a single pass over a static, incessant data is a trending shortcoming in the research area of mining of stream data. The starting point of continuous data streams arise spontaneously. When dealing with data streams it is usually impossible to store all the data from streams and only a small part of the data can be stored and used for computations within a limited time span. An idyllic procedure for mining data streams should have the following properties: high accuracy, fast adaptation to change and low

computation cost in both space and time dimensions i.e. short processing time (Bifet, Holmes, Pfahhringer, Kirkby, Gavalda, 2009). Delving into data streaming, when a significant amount of data needs to be quickly processed in real-time or near real time to gain insights, data in motion in the form of streaming data is the most productive approach. When organizations are planning for their future, they need to be able to analyse a preponderate sum of data. Stream data is an analytic computing platform that is focused on velocity because applications for streaming data require continuous and incessant influx of often unstructured data

to be processed and analyzed. Therefore, data is continuously analyzed and transformed in memory before it is stored on a disk. Therefore, is useful when analytics need to be done in real time while the data is in motion. This infers that "the value of the analysis decreases with time". A fundamental problem of data stream is that there is too much information to be stored or processed which implies that data has to be processed as it arrives either at a single-pass of the data or as a summarized data (approximation). In the case of data stream, approximate answers are questions or queries are allowed if there are guarantees of result quality (Korn, Muthukrishnan & Wu, 2006). A typical illustration that if you can't analyze and act immediately on a series of datasets, a deal opportunity might be lost, or a threat might go undetected or unnoticed. When the volume of the underlying data is very large, it leads to a number of computational and mining challenges:

i. With increasing volume of the data, it is no longer possible to process the data efficiently by using multiple passes. Rather, one can process a data item at most once. This leads to constraints on the implementation of the underlying algorithms.

Therefore, stream mining algorithms typically need to be designed so that the algorithms work with one pass of the data.

ii. In most cases, there is an inherent temporal component to the stream mining process. This is because the data may evolve over time. This behavior of data streams is referred to as temporal locality. Therefore, a straightforward adaptation of one-pass mining algorithms may not be an effective solution to the task. Stream mining algorithms need to be carefully designed with a clear focus on the evolution of the underlying data. Majority of organizations are used to handling data at rest which implies a system that control active transactions and therefore need to have persistence. However, in some cases, those transactions are been executed and analyzed in a data warehouse or data mart. This means that the information is being processed in batch and not in real time.

Stream mining is a new data mining area where data are incessant in nature (Masud et al. 2011). Huge and potentially infinite volumes of data streams are frequently generated by real time surveillance system, internet traffic, online transactions, communication networks and other dynamic environments. In traditional datasets, stream data flows in and out of a computer system continuously with varying update rate. They are massive, fast changing temporarily ordered and potentially infinite it may be impossible to store an entire data stream or scan through it multiple times due to its tremendous volume. However, stream data tends to be of rather low level of abstraction, whereas most analysts are interested in relatively high level dynamic changes, such as trends and deviations. To discover knowledge or patterns from data streams, it is necessary to develop single scan, multilevel, multi-dimensional stream processing and analysis methods.

This single scan, online data analysis methodology are not confined to only stream data but is

also critically important for processing non stream data that are massive. With data volumes mounting by terabytes or petabytes, stream data effectively captures today's data processing need, despite when the complete set of data is collected and can be stored in massive data storage devices, data stream system instead of random access may still be the most realistic processing mode because it is often too expensive to scan such dataset multiple times.

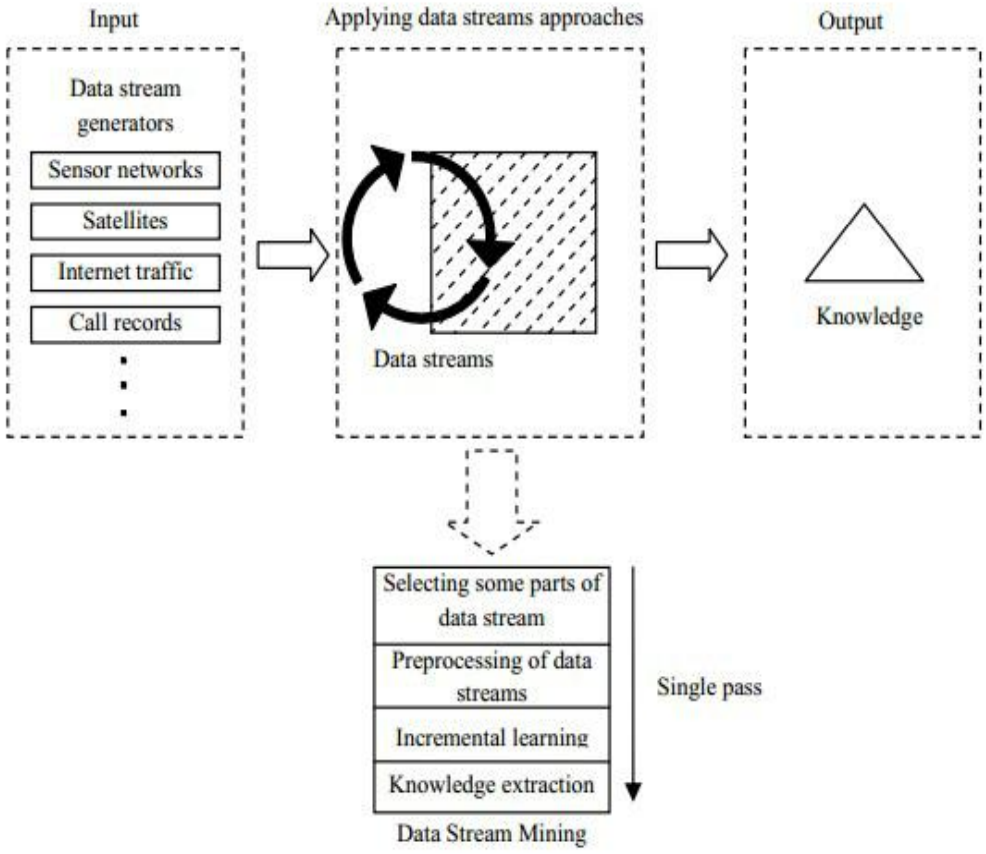


Fig 4.1 : Stream mining process (Adaptive: Mahnoosh Kholgi et al 2011)

The data stream mining task can be considered similar when compared to traditional tasks in terms of targets and objectives reasonably different in terms of data processing; this is so because of the infinite influx of high-speed data. This therefore absolutely

makes traditional data mining algorithms and methods inept of properly handling data streams.

This can be done in either of two ways;

i.by modifying an existing data/stream mining algorithm to make suitable for stream mining ii. Developing a new stream mining algorithm

Data stream can be represented as input data that comes at increasingly high rate. High rate means it stresses communication and computing infrastructure, which can make it challenging to;

- relay (R) the entire input to the program,
- compute (C) practical functions over substantial chunk of the input data presented, and
- backlog(B), acquire momentarily or extract all of continually

Set Theory

Let S represent our proposed system

S = {I, O, F, Su, Fa, φ}

Where,

I is an Input I = {DS, Ws, Ps, δ}

Where,

DS = Data Stream

Ws = Window Size

Ps = Pane Size

δ = Threshold value

O is an output = {P}

Where, P= set of frequent patterns
Success= Frequent pattern

F is a function = {LC, CW, UW, RES, MINE}

LC= LoadConnection (DS)

Input = DS (Data Stream)

Output = Imported Data

Failure = No Data imported

CW= CreateWindow (ID)

Input = Imported Data

Output = Window

Failure = No Window Created

UW = UdatedWindow (W)

Input = Window

Output = Updated Window

Failure = No Window Updated

RES = Restructure (W)

Input = Updated Window

Output = Sorted Window

Failure = No Sorted Data

MINE = Mining (Sorted Data)

Input = Sorted Data

Output = Frequent Pattern

Failure = No frequent Patterns

Fa is a failure = No frequent patterns

φ is constraints of the system
φ = {DS, Ps, Ws}

4.2 IMPLEMENTING THE PROTOTYPE INTO A REAL SIMULATION ENVIRONMENT

4.2.1 Data Preprocessing

The dataset used for this study implementation is called Netflix Shows and Movies Exploratory Data (Netflix). It is a metadata of Tv shows and movies on Netflix as of 2019. It has over 505,780 unique data values. The dataset was taken from University of Manchester Data Science stream application.

Data preprocessing is crucial aspect in the process of data mining. If data input to algorithm is not in proper format, then it cannot be processed efficiently. So pre-processing is needed and in which existing data transform into new data which is in proper format and suitable for processing.

The influx of the Netflix data stream would convey an in-depth explanation of the effectiveness of the algorithm. A view and observation into the influx shows the following few dataset items:

"Despicable me (PictureBox) was rated to be the most watched movie in the stream of data presented while Carrie, The Pink Panther (2006), LOL – HD, Django Unchained - HD_BBFC, Twilight Saga; The: New Moon – HD, Rob Roy, Reign of Assassins - HD (Subtitled), Kill for Me_BBFC, Romeo + Juliet(wt), Conan – HD, Lake Placid The Final Chapter[#] BBFC, Reign of the Assassins_BBFC, test:Playing for Keeps, Texas Chainsaw – HD, Dark Knight Rises; The – HD, Fast & Furious – HD, Pink Panther; The_MGM, Hook_BBFC, test:I Give It A Year, The test:Lords of Salem, Lions for Lambs, Dumb & Dumber, Now is Good, Three Stooges; The – HD, Midnight Cowboy, White Collar Hooligan 2[#]_BBFC, Hackers, Valkyrie – HD, Groundhog Day_BBFC and Snowmen" ect.

4.3 PATTERN SEARCH CRITERION/TYPES

In developing a pattern search for a dataset with multi-dimensional characteristics, the choice of the pattern search criterion is important. This criterion should completely

describe the dataset and truly represent the characteristics of the dataset. Some suggested search criterions are:

1. What is the pattern in the movie watched on Mondays to Sunday.
2. What pattern of movie is watched weekends?
3. What is the pattern of movie watched overall?
4. What is the pattern in the genre of overall movie
5. What is the pattern in the movie watched weekly?
6. What is the pattern in the movie watched monthly?
7. What is the pattern in the movie watched in 2013?
8. How to determine the number of transactions in a batch of transactions.
9. Provides some statistical information about the datasets used in the experimental analyses.

4.4 RESULT OF CSP PATTERN SEARCH ON MOVIE DATASET

4.4.1 Runtime Efficiency

We compared the overall runtime efficiency of CPS-tree and DSTree for mining the exact set of frequent patterns from the current window based on changes in the min_sup value. These results show that CSP-tree outperforms DSTree when the min_sup values for datasets are large-enough to produce few frequent patterns.

Memory vs Window Size

Window Size (No of Panes)	2	4	6	8	10
Memory (KB)	13.46	26.71	40.07	53.44	66.8

Pane Size = 67 KB

Runtime Vs Min_sup

Window Size (No of Panes)	2	2	2	22	2
No of Panes	10	10	10	10	10
Time (sec)	803.5159585	1203.402831	1370.541391	1856.775344	3490.327635
Min Sup	1000	900	800	700	600
Min Sup (%)	100	90	80	70	60

Since the value of (ATL/I x 100) is less than 10 for movie dataset, then the dataset is sparse.

Table 4.1: Table of values for plot for Moviedataset (Runtime Vs Min_sup)

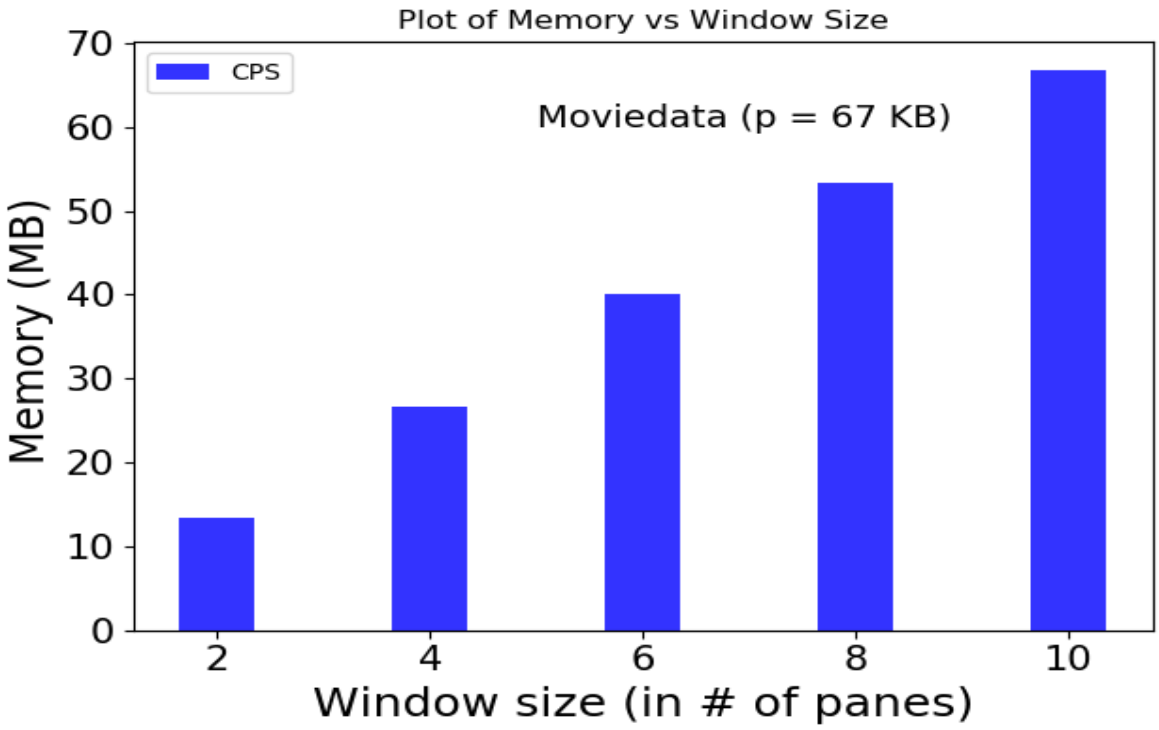


Figure 4.2: Plot of Memory against Window size

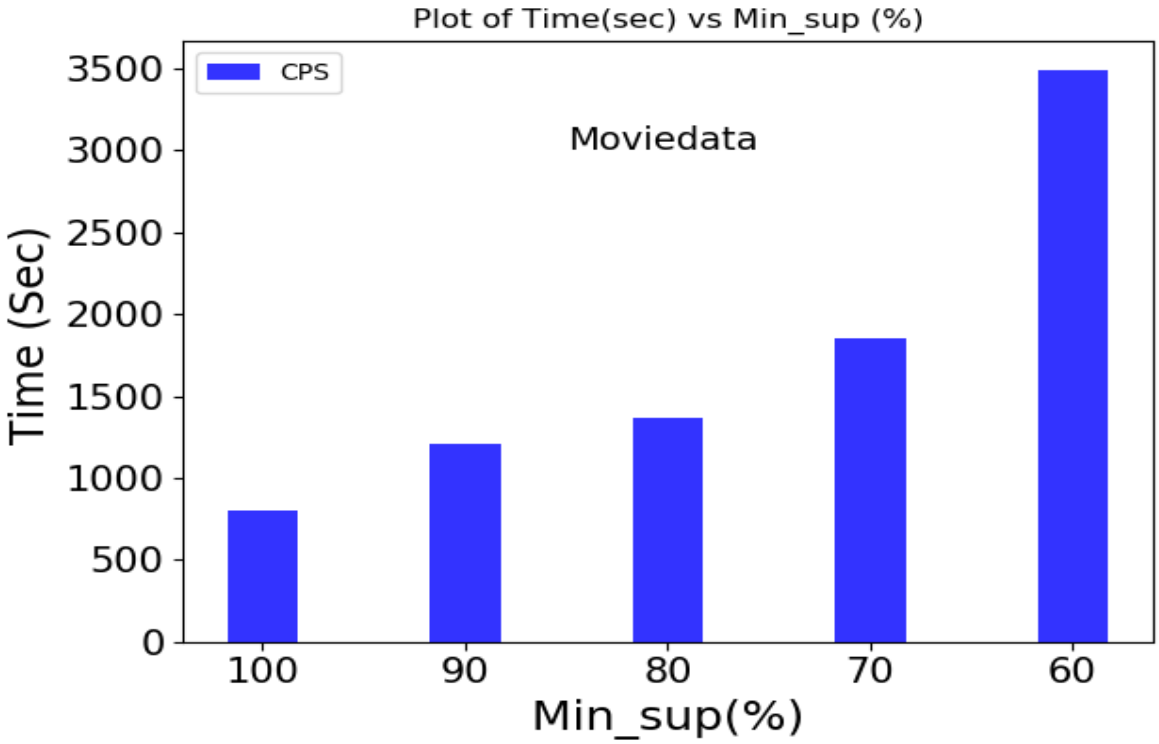


Figure 4.3: Plot of Time(s) against min_sup (%)

4.5 MOST FREQUENT PATTER FOR EACH WINDOW SIZE

Below is the result for the CPS search on the Movie dataset for window of pane size 2, 4, 6, 8, and 10 respectively.

Patterns	
Jack Reacher	*1792*
Hobbit; The An Unexpected Journey	*643*
Pitch Perfect	*82*
Impossible; The	*33*
Silver Linings Playbook	*8*
102 Dalmatians (Disney)	*1*
I Give It A Year	*1*
A Good Day to Die Hard	*1*
Seven Psychopaths	*1*
Here Comes the Boom	*1*

Jack Reacher 1792
Hobbit; The An Unexpected Journey 649
Pitch Perfect 89
Impossible; The 30
Silver Linings Playbook 9
102 Dalmatians (Disney) 1
I Give It A Year 1
A Good Day to Die Hard 1
Seven Psychopaths 1
Here Comes the Boom 1
Lincoln 1

Jack Reacher 1807
Hobbit; The An Unexpected Journey 638
Pitch Perfect 89
Impossible; The 38
Silver Linings Playbook 7
102 Dalmatians (Disney) 1
I Give It A Year 1
A Good Day to Die Hard 1
Seven Psychopaths 1
Here Comes the Boom 1
Madagascar 3: Europes Most Wanted 1

Jack Reacher 1780
Hobbit; The An Unexpected Journey 630
Pitch Perfect 93
Impossible; The 38
Silver Linings Playbook 9
102 Dalmatians (Disney) 1
I Give It A Year 1
A Good Day to Die Hard 1
Seven Psychopaths 1
Here Comes the Boom 1
Lincoln 1
Madagascar 3: Europes Most Wanted 1

Jack Reacher 1777
Hobbit; The An Unexpected Journey 624
Pitch Perfect 74
Impossible; The 31
Silver Linings Playbook 7
102 Dalmatians (Disney) 1
I Give It A Year 1
A Good Day to Die Hard 1
Seven Psychopaths 1
Here Comes the Boom 1
Madagascar 3: Europes Most Wanted 1
Shes All That (Miramax)_BBFC 1

4.5.1 Observation

Each search result gives patterns with “Jack Reacher “ as the movie with the highest frequency, but the lowest frequency movie in each patter vary due to the changing size of the number of window. This allows more items in the movie dataset to qualify to be amongst the searched items

4.6 ANSWERS TO SOME QUESTIONS ON THE RESULT

Question 1: Reason for using a particular min_sup?

Using minimum criteria of 1000 means that the only items that are counted as frequent item must h.ave a frequency greater than or equal to the minimum support. More of the items on the database are having a frequency above 1000.

Question 2: Reason for converting them to transactions?

The CSP algorithm reads patterns that are present in transactions. Therefore, the dataset are converted from mere list of items to a list of transactions.

Question 3: Why random selection?

The effectiveness of the algorithm is independent of whether it is time-based or random selection.

Question 4: How effective is the algorithm?

We can determine the effectiveness of the CPS algorithm by running it over [different sets of database].

Question 5: How the CPS algorithm works?

[['r', 'z', 'h', 'j', 'p'], ['z', 'y', 'x', 'w', 'v', 'u', 't', 's'], ['y', 'r', 'x', 'z', 'q', 't', 'p'], ['z', 'y', 'x', 'w', 'v', 'u', 't', 's'], ['y', 'r', 'x', 'z', 'q', 't', 'p'], ['y', 'r', 'x', 'z', 'q', 't', 'p'], ['z', 'y', 'x', 'w', 'v', 'u', 't', 's'], ['y', 'r', 'x', 'z', 'q', 't', 'p'], ['z'], ['r', 'x', 'n', 'o', 's'], ['y', 'r', 'x', 'z', 'q', 't', 'p'], ['z'], ['r', 'x', 'n', 'o', 's'], ['y', 'r', 'x', 'z', 'q', 't', 'p'], ['z'], ['r', 'x', 'n', 'o', 's'], ['y', 'r', 'x', 'z', 'q', 't', 'p'], ['r', 'x', 'n', 'o', 's'], ['y', 'r', 'x', 'z', 'q', 't', 'p'], ['z'], ['r', 'x', 'n', 'o', 's'], ['y', 'r', 'x', 'z', 'q', 't', 'p'], ['y', 'z', 'x', 'e', 'q', 's', 't', 'm']].

The dataset above consists of 23 transactions. The main CPS code is 'app_main = AppMain(2, 12, 1, data_one, 1)'. This line of code uses a window containing 2 panes, with each pane containing 12 transactions. This covers the entire 23 transactions present in data_one. Using a minimum support of 1, it captures all the elements present in data_one i.e {'y', 'm', 'v', 'n', 'h', 'o', 's', 'p', 'w', 'z', 't', 'q', 'x', 'u', 'r', 'e', 'j'}as the 'most frequent item'.

Question 6: Give a simple analysis on the movie dataset?

Dataset	#Trans. (T)	#Items (I)	MaxTL (MTL)	AvgTL (ATL)	(ATL/I)/ 100
Movie Data	17066	366453	38	21	0.00570612

Table 4.3: Movie Dataset Characteristics

From the table 4.3 above Since the value of (ATL/I x 100) is less than 10 for movie dataset, then the dataset is sparse.

The movie dataset has 637 different locations. It has 13 genres with comedy with the highest frequency of 101701 and animation with the lowest frequency of 87. It is rated from 0.0 to 6.0 with 3.5 having the highest frequency of 69145and 6.0 having the least frequency of 5.

4.7 EXPLANATION OF MOVIE DATA ANALYSIS RESULT

Case 1:

Window Size: 2

Number of Pane: 10

Minimum support: 1000

Number of items in movie dataset: 366453

Number of items in a transaction: 36

Number of transactions in a pane: 1703

Number of transactions: 17033

The maximum transactions length is: 38

The minimum transactions length is: 2

The average transaction length is : 21

ATL/I x 100 = 0.005730612111239368 (Since the value of (ATL/I x 100) is less than 10 for movie dataset, then the dataset is sparse)

Data size of panes is : 6848Kb

Datasize of data_set is : 142696Kb10. Write codes to get the dataset characteristics

a) Number of transactions (T)

b) Number of items (I)

c) Maximum transaction length (MTL)

d) Average transaction length (ATL)

4.8 PRODUCING AND MAINTAIN ASSOCIATION RULES

Mining Association rule involves quite a bit of memory and CPU costs. A noticeable drawback is that processing time is always limited to only one online scan. So there is need of real time maintenance and updating association rule. However stream data, if we update association rules too frequently, the cost of computation will increase drastically.

4.8.1 Resource Awareness

Most of the resources such as memory space and CPU in data stream mining are vital. One cannot ignore the resources availability, a typical example is when the main memory is totally used up in processing, data will be lost and it leads to errors or inaccuracies in results. These challenges need to be considered in order for our algorithm to perform at its utmost best.

4.8.2 Scalability

One of the main challenges in data streaming is the issue of scalability. Data stream being a very recent research field is experiencing exponential advancement in a way much faster than most computer resources. The processors follow Moore's law, but the size of data is increasingly exploding. Consequently, research resolutions should be geared towards developing scalable frameworks and algorithms that will be able to contain data stream computing methods, effective resource allocation strategy, parallelization and complementing issues to cope with the ever-growing size and complexity of data.

4.8.3 Timeliness

Time is of the essence for time-sensitive processes such as mitigating security threats, impeding fraud, or responding to a disaster of any sort. There is a need for architectures or platforms that will enable continuous processing of data streams which can be used to maximize the timeliness of data. The main challenge is implementing a distributed architecture that will aggregate local views of data into global view with minimal latency between communicating nodes. Consistency Achieving stability in data stream computing environments is non-trivial as it is demanding to determine which data are needed and which nodes should be consistent. Hence a satisfactory system structure is required.

4.8.4 Heterogeneity and incompleteness

Data streams are heterogeneous in design, organisations, semantics, accessibility and granularity. The challenge here is how to effectively handle an always ever-increasing data, extract meaningful content out of it, aggregate and correlate streaming data from multiple sources in real-time. A competent data presentation should be designed to reflect the structure, diversity and hierarchy of the streaming data.

4.8.5 Accuracy

One of the main objectives of big data stream analysis is to develop effective techniques that can accurately predict future observations. However, as a result of inherent characteristics of big data such as volume, velocity, variety, variability, veracity, volatility, and value, big data analysis strongly constrains
processing algorithms spatiotemporally and hence stream-like requirements must be taken into consideration to ensure high accuracy.

4.8.6 Atomicity

Independent transaction constitutes an indivisible entity of processing. This property refers to the ability of the stream mining algorithm to guarantee that either all of the tasks of a transaction are performed or none of them are.

4.9 METHOD OF ENSURING DATA SET INTEGRITY IN DATA STREAM

Individual transaction arrives a logically consistent set of operations. The system has to ensure the realization of every transaction will be accomplished with either an occurrence or relapse. Contingent to a failure, the system has to assure that no changes will be incurred on the incoming dataset. The execution of this process in a multi-access environment establishes one of the basic factors influencing the efficiency of the whole computer system.

In 1983, T. Härder and A. Reuter (1983) proposed the suitability of transaction properties prescribed:

4.9.1 Consistency – a stream of data is in a persistent and rational state if there is a potential of achieving an unambiguous and systematic pattern from the information accrued in a data set. The consistency of the database plays an important role in actualizing the transaction. This property refers to the dataset being in an appropriate state when the transaction begins and when it terminates.

4.9.2 Isolation – a transaction happens independently of different operational systems and sources of data including different transactions. The changes introduced in a data set are visible for other transactions only after the termination of the transaction. This property refers to the ability of the application to make operations in a transaction appearing separately from every other operation.

4.9.3 Persistent – in case of a system breakdown, the data set will not lose its consistency. This property refers to the guarantee that once the user has been notified of success, the transaction will persist and will not be undone.

4.10 CONCLUSION

After conducting a thorough literature review, it is clearly understood that it is evident that the existing problems of data stream are still enormous and not much work has been focused on analysing stream data, especially with regards to tree structures. In view of this, this research focuses on designing an algorithm that mines frequent pattern in comparison to other single pass algorithms. The designed framework will be implemented in order to test the proposed algorithm. Different scenarios will be created based on the defined parameters in order to obtain different results for analysis and to propose future work.

CHAPTER FIVE

5.0 EVALUATION

We evaluated the performance of the dynamic CSP algorithm on various dataset through some extensive experiments. In our performance evaluation we consider the run time and memory usage for different threshold values. The runtime rates of the algorithms under study are very important and vital in the analysis of this research work. For our experimental evaluation, we use eight publicly available datasets from the UCI machine learning repository. Recall that pattern mining methods require the data to be discrete. Our performance focuses predominantly on finding frequent pattern within a single scan over dynamic data stream using effective compact sliding window tree structure which avoid the duplicate items in transactions, and provide accurate frequent patterns. Additionally, the algorithm extract mining results regarding the latest data over data streams, and can gain the resulting pattern more quickly, so this approach is very useful to find frequent pattern over wide range of data stream like retail, telecoms, marketing and network, Call records and other related data stream. The extensive experiments presented in this research work shows that the proposed algorithm could mine latest frequent patterns more effectively than the previous algorithms and it gives outstanding performance in terms of runtime, memory usage.

5.1 **EXPERIMENTAL EVALUATION**

Runtime efficiency: this represents the measure of amount of time needed for the CSP algorithm to execute each stream of data presented to it in comparison with that of the FPTree.

Test Runtime	*CSPTree*			*FPTree*			*Window Size*	*Pane Size*
	Creation	Restruct	Mining	Creation	Restruct	Mining		
	1.87	2.97	0.00040	2.53	3.64	0.00021	2	10K
	3.55	4.65	0.00062	5.96	6.52	0.00044	3	10K
	6.17	6.76	0.00107	11.02	11.02	0.00073	4	10K
	7.90	9.23	0.00301	13.20	15.09	0.00221	5	10K

Table 4.1: Runtime analysis

The table 4.1 above shows the Test runtime for the BMS dataset which comprises of the tree creation, tree restructuring and how long it takes to mine a specific dataset present in the pane of a window.

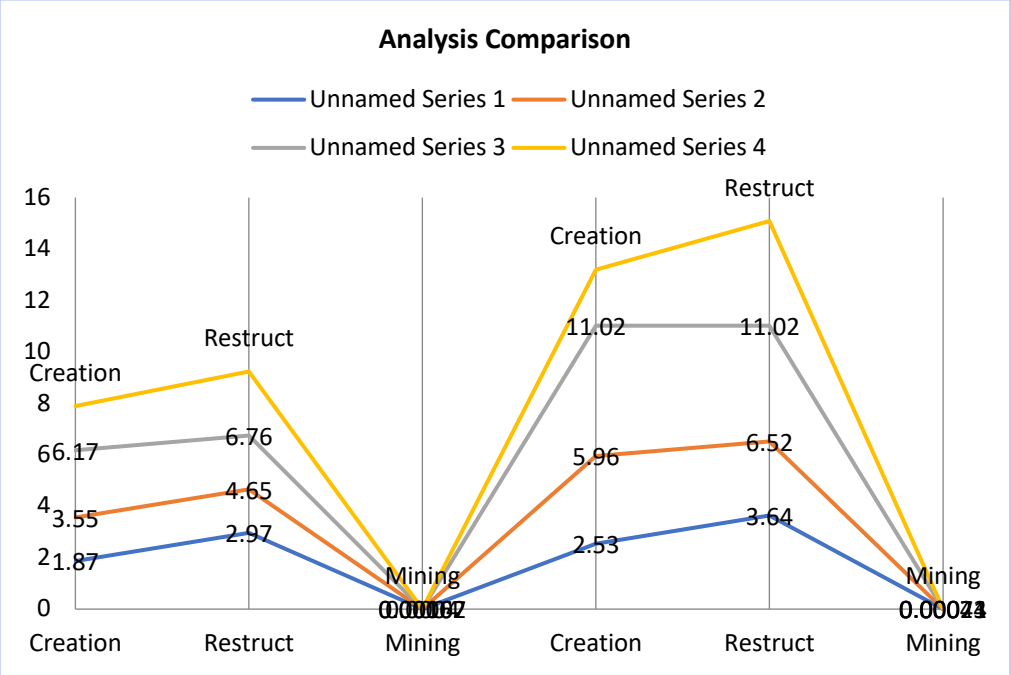
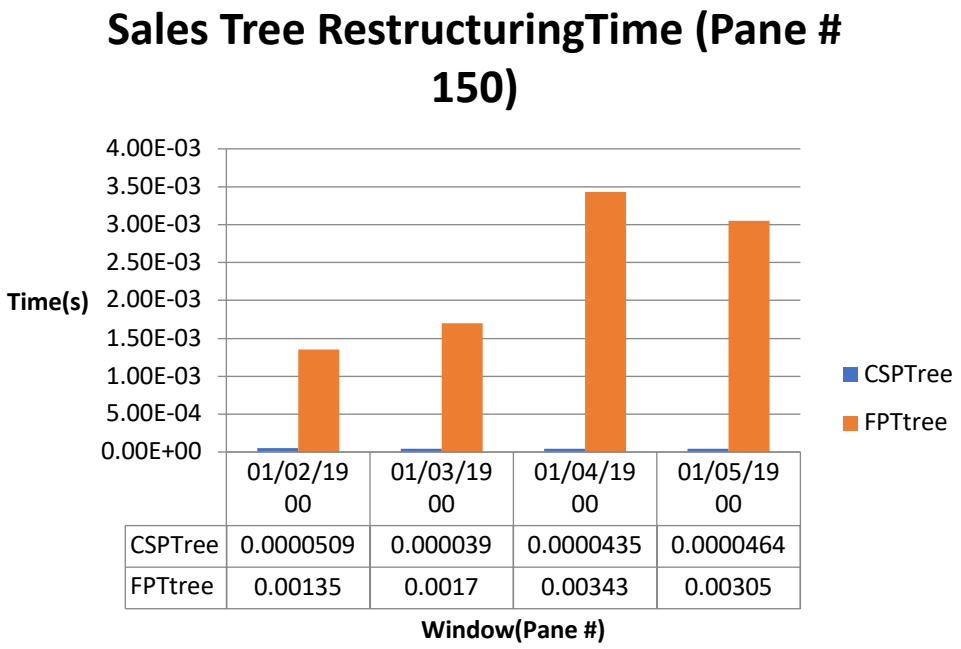


Figure 5.1: BMS Tree Creation Time

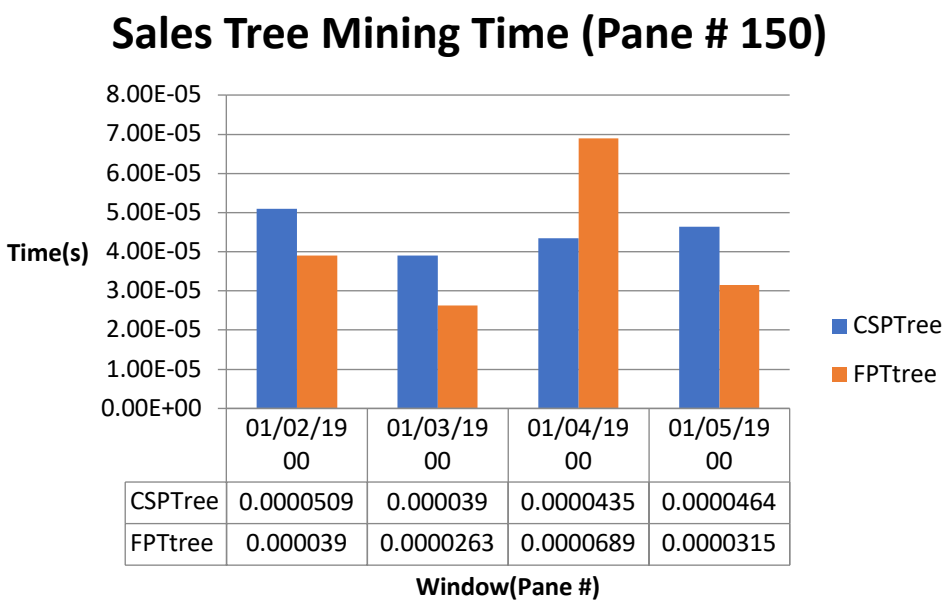
The fig 5.1 above shows the tree creation time for BMS dataset between CSP Tree and FP tree. The resulting outcome explains that CSP Tree has a faster rate of tree creation when compared to FP The resulting outcome shows that CSP Tree has a faster rate of restructuring each tree when compared to FP Tree with the BMS dataset.



	01/02/1900	01/03/1900	01/04/1900	01/05/1900
CSPTree	0.0000509	0.000039	0.0000435	0.0000464
FPTtree	0.00135	0.0017	0.00343	0.00305

Figure 5.2: CSP AND FP Comparison on BSM DATA

The table 5.2 above shows the Test runtime for the Sales dataset which comprises of the tree creation, tree restructuring and how long it takes to mine a specific dataset present in the pane of a window.

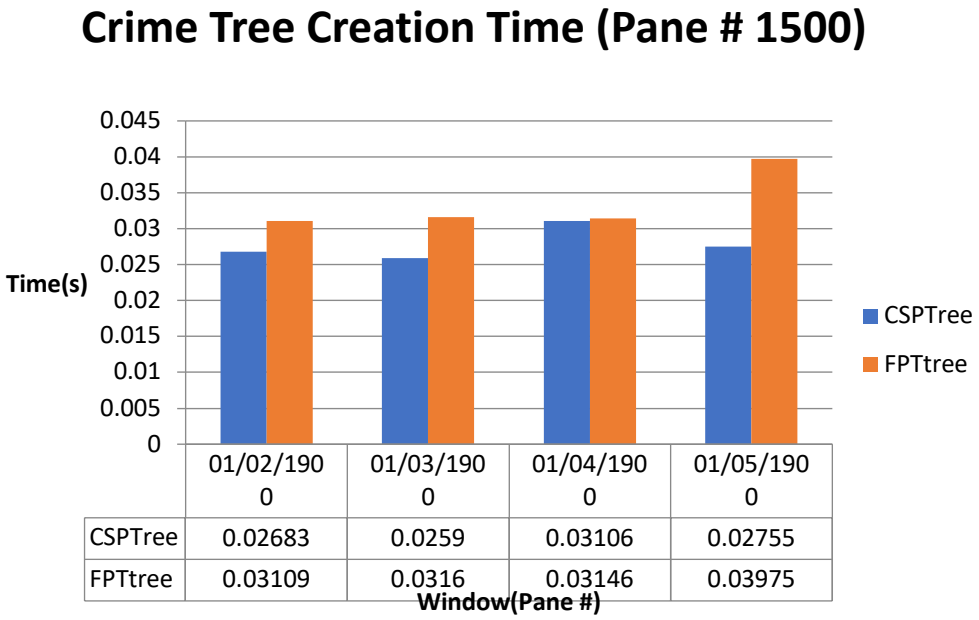


	01/02/1900	01/03/1900	01/04/1900	01/05/1900
CSPTree	0.0000509	0.000039	0.0000435	0.0000464
FPTtree	0.000039	0.0000263	0.0000689	0.0000315

Figure 5.3: Sales Tree Creation Time

Fig 5.3 above shows the tree creation time for Sales dataset between CSP Tree and FP tree. The resulting outcome explains that FP Tree has a faster rate of tree creation when compared to CSP Tree. This is so because of the limited amount of data presented in the form of a stream. It can be said to be a negative test case.

For each step of the restructuring process, CSP tree again proves to be the most effective and efficient algorithm in comparison to FP tree



	01/02/1900	01/03/1900	01/04/1900	01/05/1900
CSPTree	0.02683	0.0259	0.03106	0.02755
FPTtree	0.03109	0.0316	0.03146	0.03975

Figure 5.4: Mining time for Sales Dataset

In fig 5.4 above due to the limited arrival of data, FP tree tends to mine faster as there is no enough dataset for CSP to handle. Also, this is a negative test case to know how effective FP Tree.

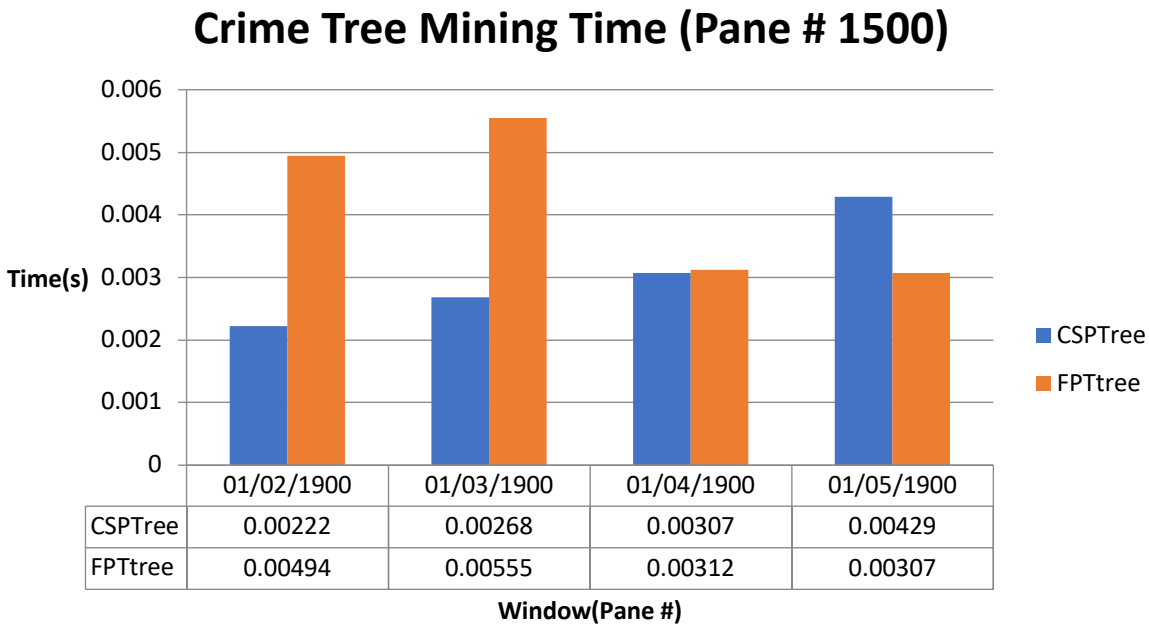


Figure 5.5 CSP & FP comparison

Fig 5.5 above shows a graphical comparison between CSP Tree and FP Tree. As discussed earlier, due to the inconsistency and the scarce proportion of the dataset, FP tree seems to outperform CSP tree.

Total Runtime	CSPTree			FPTree			Window Size	Pane Size
	Creation	Restruct	Mining	Creation	Restruct	Mining		
	0.02683	1.85e-5	0.00222	0.03109	1.72e-05	0.00494	2	1500

	0.02590	1.64e-5	0.00268	0.03160	1.81e-05	0.00555	3	1500
	0.03106	8.08e-5	0.00307	0.03146	1.72e-05	0.00312	4	1500
	0.02755	2.01e-5	0.00429	0.03975	1.48e-05	0.00307	5	1500

Table 5.2: Crime Dataset

The table 5.2 above shows the Test runtime for the Crime dataset which comprises of the tree creation, tree restructuring and how long it takes to mine a specific crime rate present in the pane of a window.

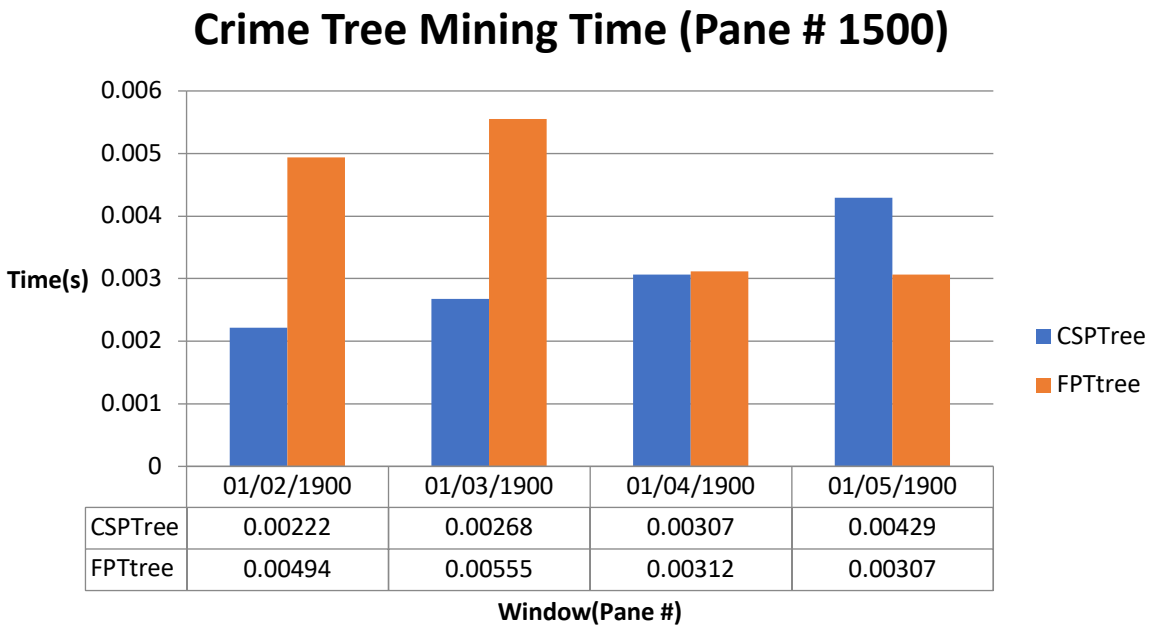


Figure 5.6 Crime Tree Creation Time

The fig 5.6 above shows the tre creation time for the crime dataset between CSP Tree and FP tree. The resulting outcome explains that CSP Tree has a faster rate of tree creation when compared to FP Tree. This is so because of the consistent arrival of the stream data presented. CSP Tree outperforms FP tree at every level of creation in the window. It can be said to be a positive test case.

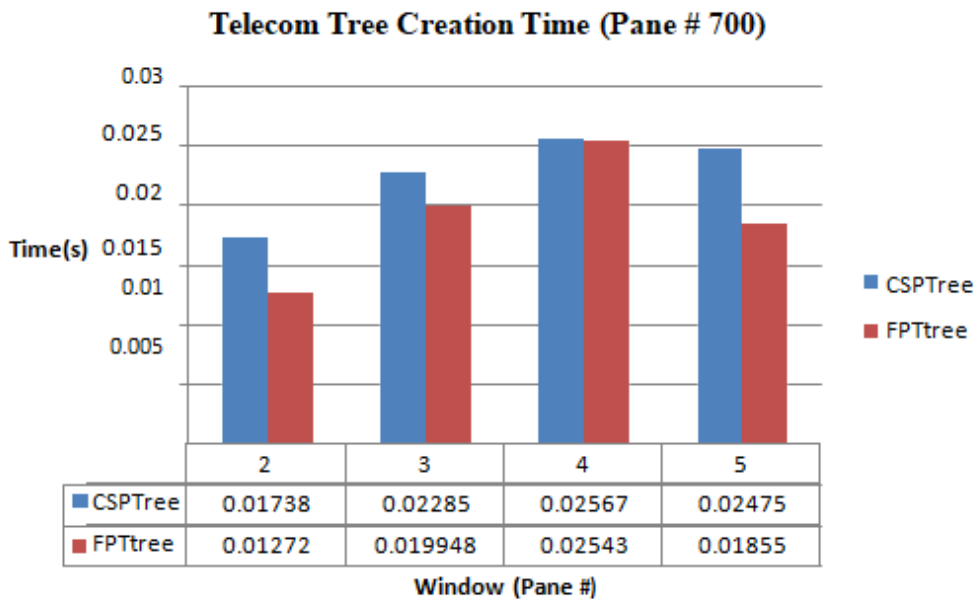


Figure 5.7: Telecoms Tree Creation Time

The fig 5.7 above shows the tree creation time for the crime dataset between CSP Tree and FP tree. The resulting outcome explains that FP Tree in the first window has a faster tree creation rate compared to CSP Tree. And at each subsequent window the time it takes to create the tree gradually lowers till the necessary patterns are derived.

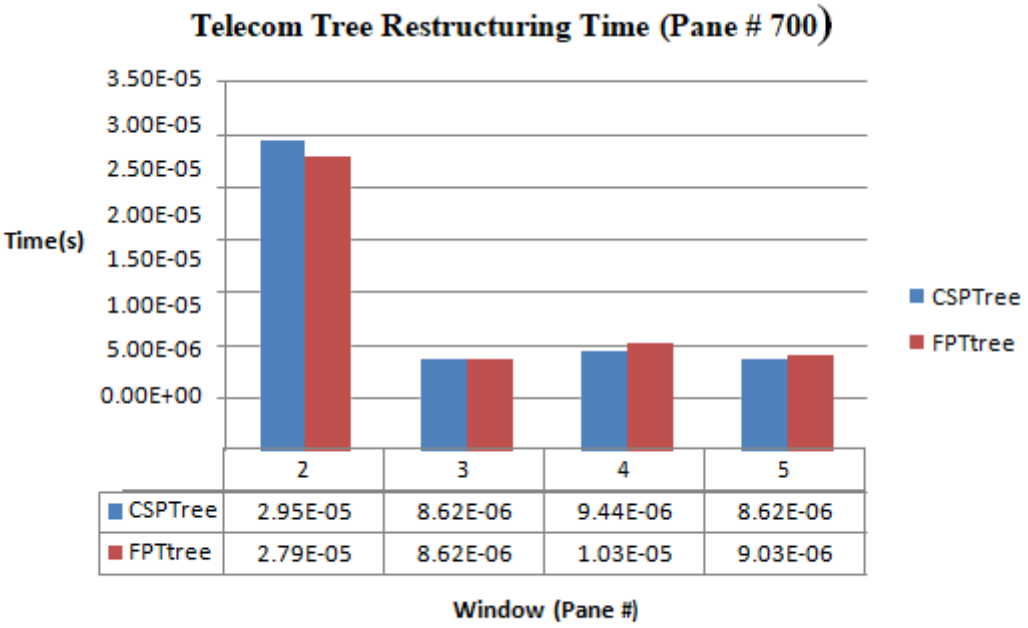


Figure 5.8: Telecoms Tree Restructuring Time

Figure 5.8 above shows the time it takes to restructure each incoming tree for the two algorithms. In the first window, FP tree restructures faster but in subsequent windows CSP Tree outperforms it.

Total Runtime	CSPTree			FPTree			Window Size	Pane Size
	Creation	Restruct	Mining	Creation	Restruct	Mining		
	6.476	2.46e-5	0.00447	10.50	3.49e-05	0.33	2	2M
	5.97	2.30e-5	0.00443	11.07	2.95e-05	0.09	3	2M
	6.015	2.59e-5	0.00437	9.00	2.54e-05	0.09	4	2M
	5.84	2.59e-5	0.00277	10.56	2.67e-05	0.06	5	2M

Table 5.3: Call Care Datasets

The table 5.3 above shows the Test runtime for the Call care dataset which comprises of the tree creation, tree restructuring and how long it takes to mine a given instance of the telecoms data present in the pane of a window.

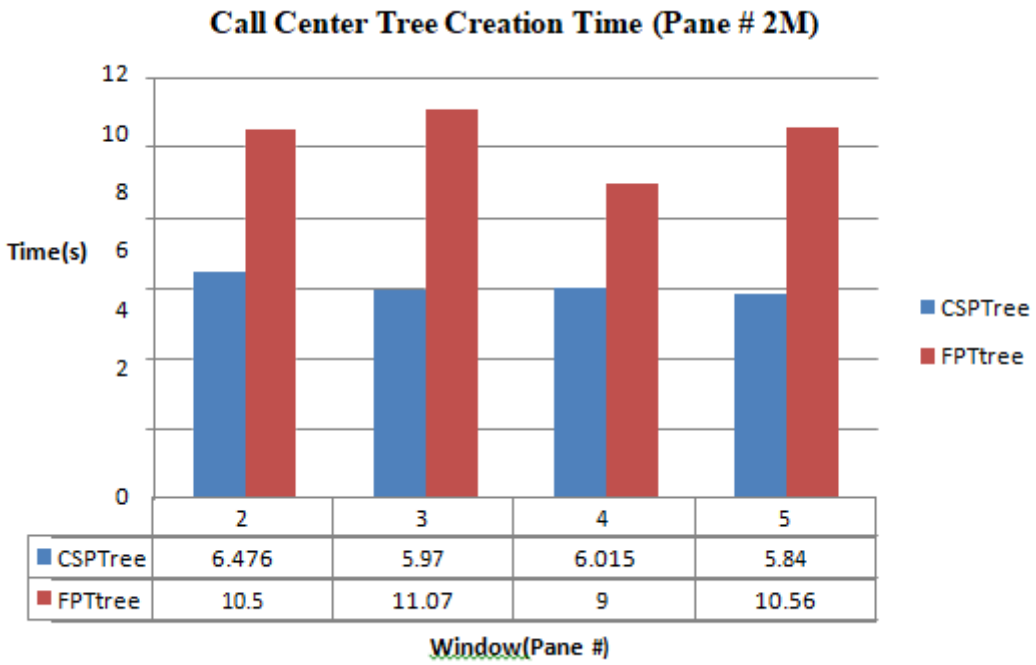


Figure 5.9: Call Center Tree Creation Time

The fig 4.13 above shows the tree creation time for the Call care dataset between CSP Tree and FP tree. The resulting outcome demonstrates and confirms that CSP Tree in the entire window has a faster rate of tree creation when compared to CSP Tree

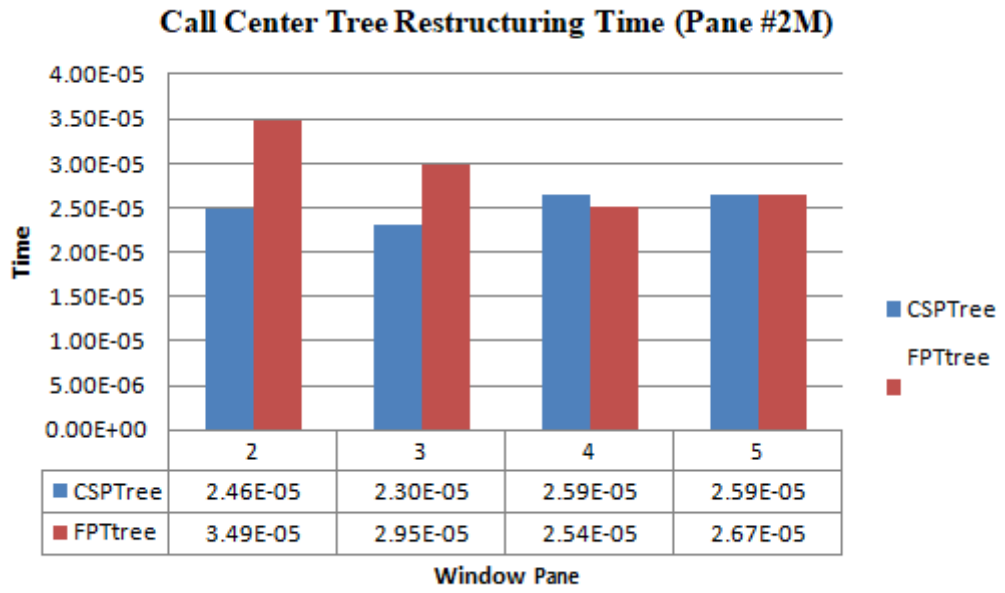


Figure 5.10: Call Center Tree Restructuring Time

The fig 5.10 above shows the time it takes for the trees created for the Call Center dataset to be restructured between CSP Tree and FP tree. The resulting outcome shows that CSP Tree has a faster rate of restructuring each tree when compared to FP Tree with the call center dataset.

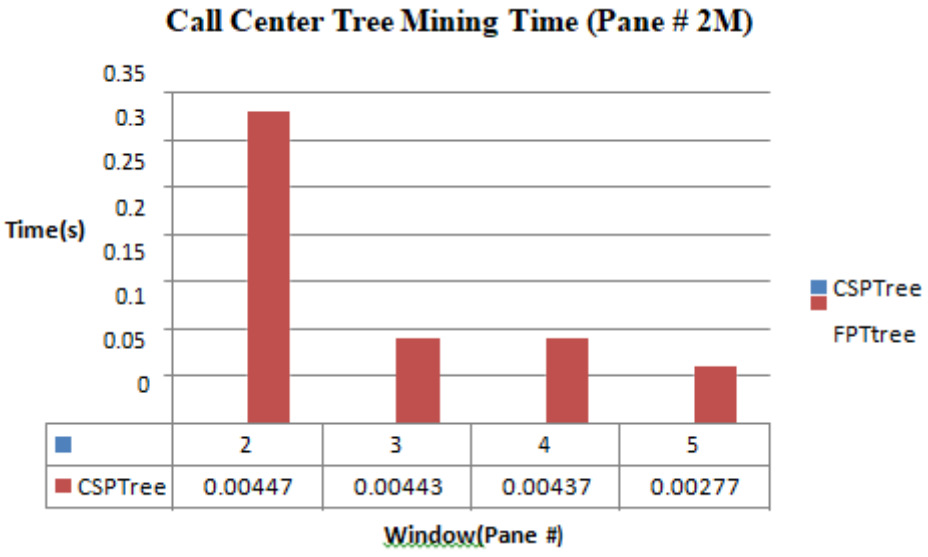


Figure 5.11: Call Center Tree Mining Time

In fig 5.11 above when all necessary conditions to the consistent arrival of data are met, CSP tree tends to mine faster as there is an influx dataset for CSP to handle. This is a positive test case which shows are requirements meet the actual results over FP tree as expected.

5.2 MEMORY USAGE

In the representation of space efficiency which measures the amount of memory needed for an algorithm to execute a given dataset. The tables below are detailed space efficiency of CSP Algorithm when compared to the FP algorithm.

CSPTree	FPTree	Window Size	Pane Size
118	121	2	10K
140	144	3	10K
160	166	4	10K
179	188	5	10K

Table 5.4: Memory Usage for BMS

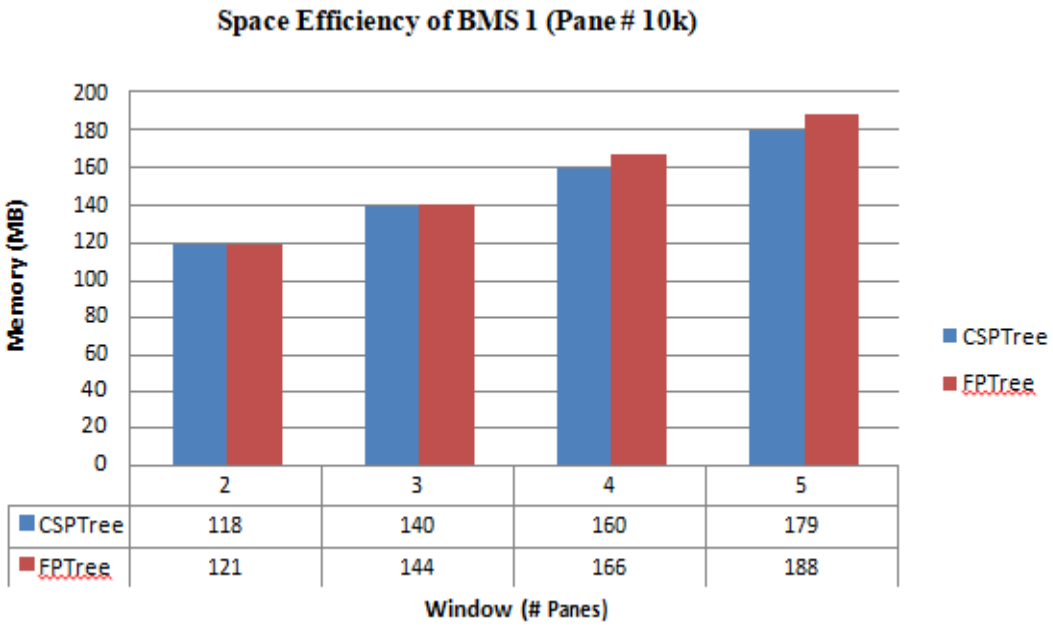


Figure 5.12 Space Efficieny of BMS

The figure 5.12 above shows how efficient the CSP Tree algorithm has been able to utilize the minimal memory assigned to it compared to FP Tree. Which means it only needs a n infinitesimal amount of memory to function effectively.

CSPTre	FPTree	Window Size	Pane Size
456	460	2	150
458	458	3	150
460	459	4	150
458	458	5	150

Table 5.5: Memory Usage for Sales Dataset

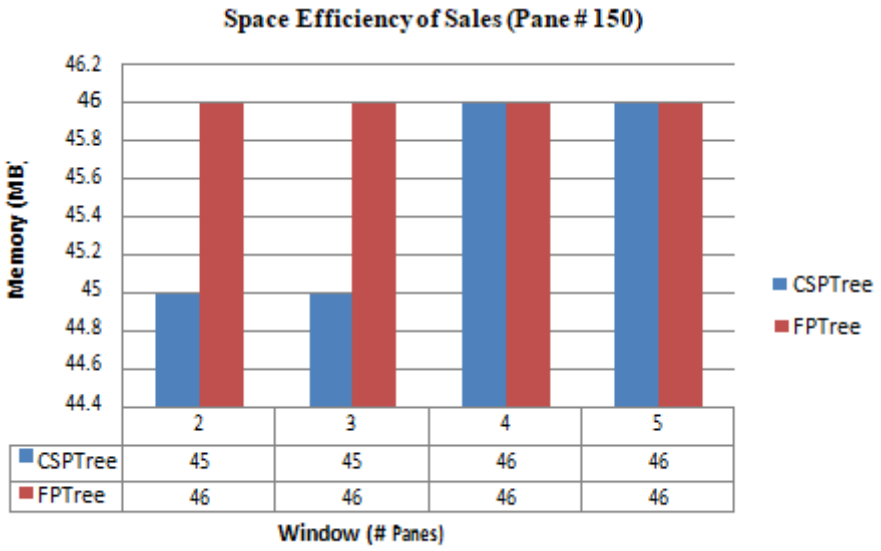


Figure 5.13: Space Efficiency of Sales

As expected, fig 5.13 above shows how efficient the CSP Tree algorithm has been able to utilize the minimal memory assigned to it compared to FP Tree. This means it has higher memory adeptness when compared with FP Tree Algorithm.

Crime Dataset

CSPTree	FPTree	Window Size	Pane Size
47853568	48009216	2	1500
47742976	48095232	3	1500
48140288	48336896	4	1500
47747072	48099328	5	1500

Table 5.6: Memory usage on Crime Dataset

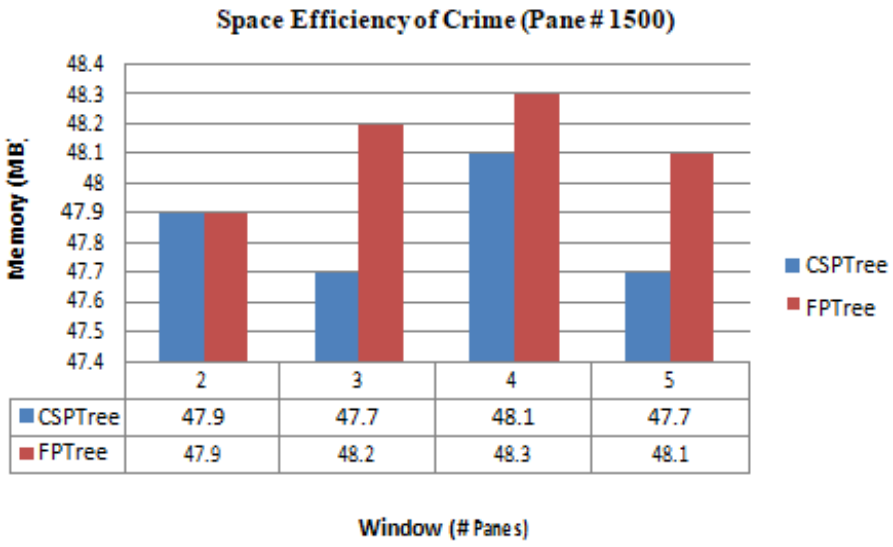
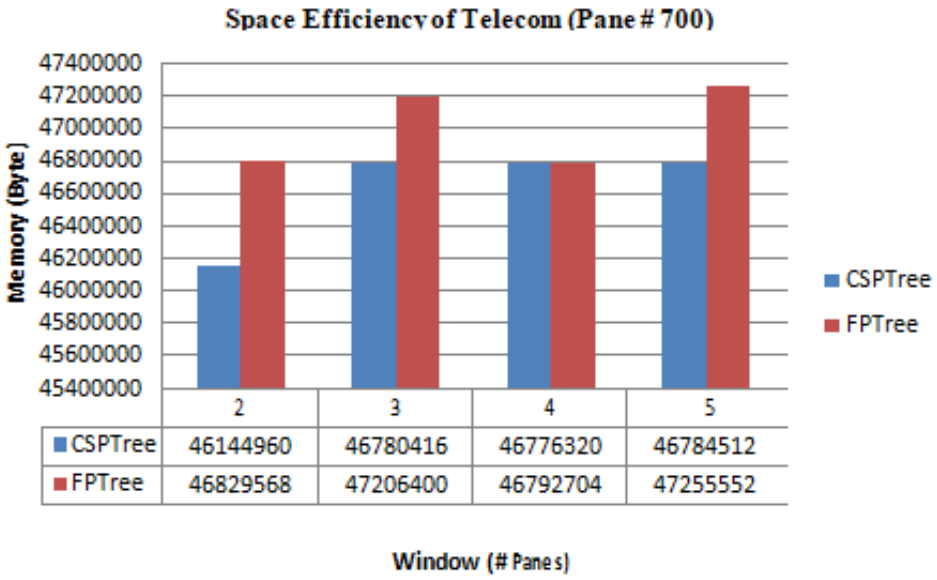


Figure 5.14: Space Efficiency on Crime dataset

The fig 5.14 above shows how efficient the CSP Tree algorithm has been able to utilize the minimal memory assigned to it compared to FP Tree. Which means it only needs a n infinitesimal amount of memory to function effectively.

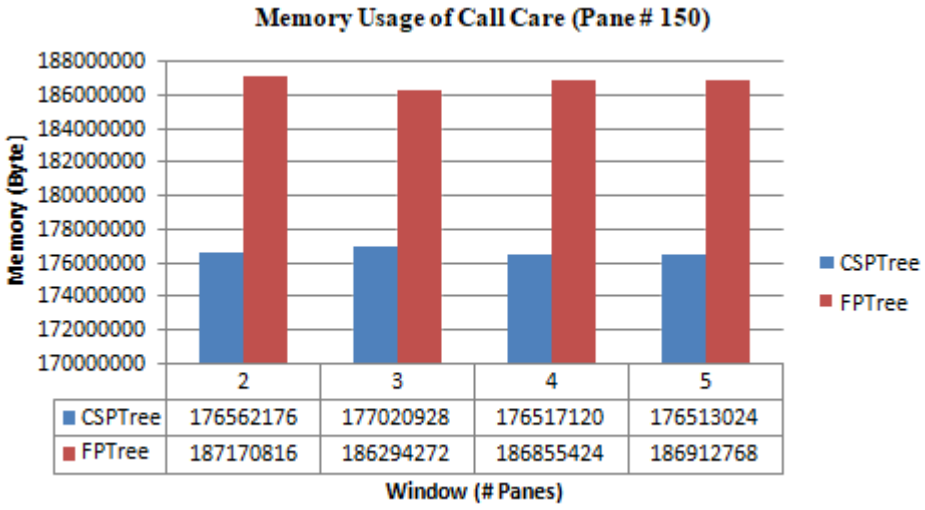
CSP Tree	Fp Tree	Window size	Pane Size
46144960	46829568	2	700
46780416	47206400	3	700
46776320	46792704	4	700
46784512	47255552	5	700

Table 5.7: Memory usage on Telecoms Dataset



	2	3	4	5
CSPTree	46144960	46780416	46776320	46784512
FPTree	46829568	47206400	46792704	47255552

Figure 5.15: Space Efficiency of Telecoms



	2	3	4	5
CSPTree	176562176	177020928	176517120	176513024
FPTree	187170816	186294272	186855424	186912768

Figure 5.16: Space Efficiency for Call care

The fig 5.16 above proves how efficient CSP Tree algorithm has been able to utilize the exploit the memory assigned to it compared to FP Tree. This means when provided a limited amount of it can still function efficiently and effectively.

5.3 ANALYTICAL EVALUATION

The analysis of any algorithm is duly focused on time complexity and space complexity. As compared to time plexity, the space complexity analysis requirement for an algorithm is pretty much straightforward, but Wherever and whenever necessary both of them need to be put to used space complexity refers to the amount of working storage that is needed by a given algorithm. The amount of memory needed by a program to run from start to completion is referred to as Space complexity while the amount ime needed by the program to run from start to completion is referred to as the Time complexity, which typically ends on the size of the input data. It is a function of size: (n) [T (n)].

§ Best Case:

It is the function defined by the maximum number of steps taken on any instance of size (n).

§ Average Case:

It is the function defined by the Average number of steps taken on any instance of size (n).

§ Worst Case:
It is the function defined by the minimum number of steps taken on any instance of size (n).

5.4 ASYMPTOTIC ANALYSIS

Asymptotic Analysis is the big idea that handles above issues in analyzing algorithms. In Asymptotic Analysis olves assessing and weighing the performance(s) of an algorithm in terms of input size (we don’t measure the ual running time). We calculate,

how does the time (or space) taken by an algorithm increases with the input. An increase in the input size to the time and space is greatly considered and calculated. Asymptotic Analysis is not perfect, but it's an ideal way to analyzing an algorithm. For example, say there are two ting algorithms that take 1000nLogn and 2nLogn time respectively on a machine. Both of these algorithms are mptotically the same (order of growth is nLogn)

5.4.1 Worst, Average and Best Cases

Aymptotic analysis overcomes the problems of naive way of analyzing algorithms.

We can have three cases to analyze an algorithm:

i) Worst Case

ii) Average Case

iii) Best Case

5.4.2 Worst Case Analysis

In the worst-case analysis, we calculate upper bound on running time of an algorithm. It is essential to know the e(s) that cause maximum number of operations to be executed. Compact Search Algorithm, the worst case happens when the element to be searched (x in the above code) is present in the array. When x is not present, the search() function compares it with all the elements of the array[] by one. Therefore, the worst-case time complexity of linear search would be Θ(n) worst-case complexity (denoted in asymptotic notation) measures the resources (e.g. running time, memory) algorithm requires given the worst-case. It gives an upper bound on the resources required by the algorithm. In case of running time, the worst-case time-complexity indicates the longest running time performed by an orithm given any input of size n, and thus this

guarantees that the algorithm finishes in a short time. Moreover, order of growth of the worst-case complexity is used to compare the efficiency of two algorithms.

5.4.3 Average Case Analysis

In the average case analysis, we take all possible inputs and calculate computing time for all of the inputs. Sum all the culated values and divide the sum by total number of inputs. We must know (or predict) distribution of cases the linear search problem, let us assume that all cases are uniformly distributed (including the case of x not ng present in array). So, we sum all the cases and divide the sum by (n+1). Following is the value of average e time complexity.

$$\text{Average Case Time} = \frac{\sum_{i=1}^{n+1} \theta(i)}{(n+1)}$$

$$= \frac{\theta((n+1)*(n+2)/2)}{(n+1)}$$

$$= \Theta(n)$$

5.4.4 Best Case Analysis

A best-case analysis, the lower bound on running time of an algorithm is being determined. It is required to be are of the case(s) that cause the minimum number of operations to be executed.

The linear search problem, the best case occurs when x is present at the first location. The number of operations he best case is constant (not dependent on n). So time complexity in the best case would be Θ (1) set of the times, we do worst case analysis to analyze algorithms. In the worst case analysis, we guarantee an er bound on the running time of an algorithm which is a valuable and acceptable information average case analysis is not easy to do in most of the practical cases and it is rarely done. In the

average case lysis, we must know (or predict) the mathematical distribution of all possible inputs. Best-Case analysis guarantees a lower bound on an algorithm and doesn't provide any information as compared he worst case which can result in an algorithm taking an awfully long time to run alternatively, for some algorithms all the cases are asymptotically the same, i.e., there are no worst and best cases. example, Merge Sort does Θ(nLogn) operations in all cases. Most of the other sorting algorithms have worst best cases. For example, in the typical implementation of Quick Sort (where pivot is chosen as a corner), the worst occurs when the input array is not sorted, and the best occur when the pivot elements always in the array in two halves is already sorted. For insertion sort, the worst case occurs when the array is reverse sorted, and the best e occurs when the array is sorted in the same order as output.

5.5 EFFICIENCY OF ALGORITHM (Asymptotic Notations)

The main idea of asymptotic analysis is to have a measure of efficiency of algorithms that doesn't depend on chine specific constants and doesn't require algorithms to be implemented and the time it takes a program to be pared. Asymptotic notations are mathematical tools to represent time complexity of algorithms for asymptotic analysis. following 3 asymptotic notations are mostly used to represent time complexity of an algorithm.

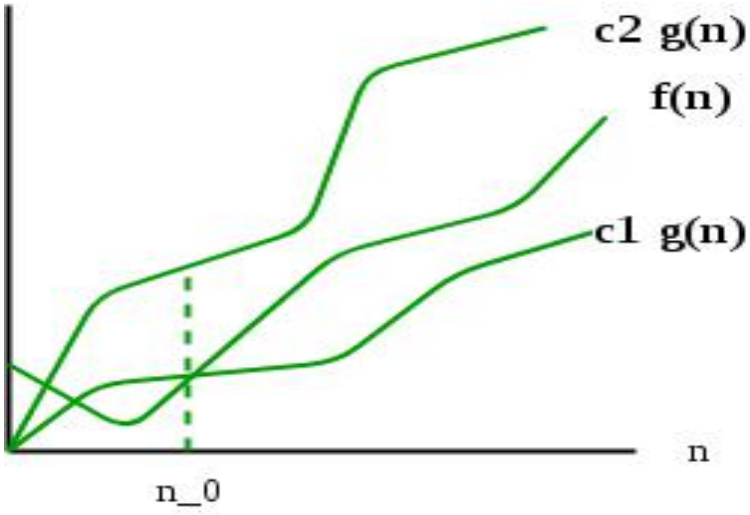


Figure 5.17: Graph complexity at Theta notation **f(n) = theta(g(n))**

5.4.1 θ Notation: The theta notation bounds a function from above and below, so it defines the exact asymptotic avior.

A simple way to get Theta notation of an expression is to drop low order terms and ignore leading constants.

For a given function g(n), we denote Θ(g(n)) in the following set of functions.

Θ(g(n)) = {f(n): there exist positive constants c1, c2 and n0 such
that 0 <= c1*g(n) <= f(n) <= c2*g(n) for all n >= n

The above definition means, if f(n) is theta of g(n), then the value f(n) is always between c1*g(n) and c2*g(n) for e values of n (n >= n0). The definition of theta also requires that f(n) must be non-negative for values of n ater than n0.

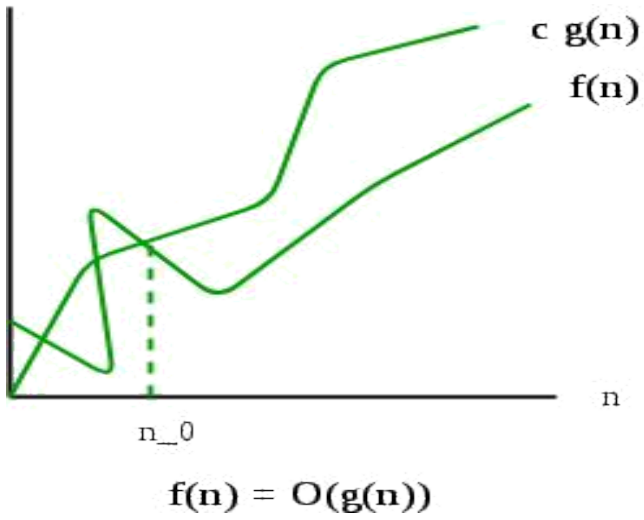


Figure 5.18: Graph complexity at 0 Notation

5.4.2 Big O Notation

Big O notation defines an upper bound of an algorithm; it bounds a function only from above. For example, sider the case of Insertion Sort. It takes linear time in best case and quadratic time in worst case. We can safely that the time complexity of Insertion

sort is O(n^2). Note that O(n^2) also covers linear time. We use Θ notation to represent time complexity of Insertion sort, we have to use two statements for best and rst cases:

the worst-case time complexity of Insertion Sort is Θ(n^2).

the best-case time complexity of Insertion Sort is Θ(n).

Big O notation is useful when we only have upper bound on time complexity of an algorithm. Many times, easily find an upper bound by simply looking at the algorithm.

(n)) = {f(n): there exist positive constants c and
n0 such that 0 <= f(n) <= c*g(n) for
all n >= n0}

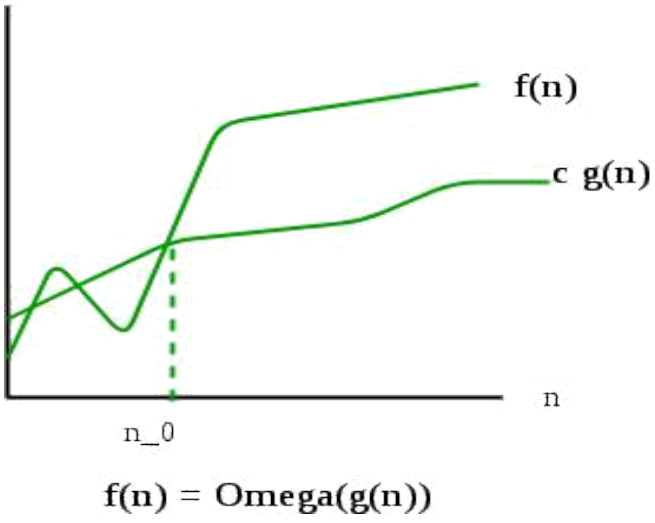


Figure 5.19:Gaph complexity at Omega Notation

5.4.3 Ω Notation

The Big O notation provides an asymptotic upper bound on a function, Ω notation provides an asymptotic lower bound. The Big O notation can be useful when we have

lower bound on time complexity of an algorithm. As discussed previously, best case performance on an algorithm is generally not useful, the Omega notation is the least used notation ong all three.

a given function g(n), we denote by Ω(g(n)) the set of functions.

g(n)) = {f(n): there exist positive constants c and
n0 such that 0 <= c*g(n) <= f(n) for all n >= n0}.

5.5 CONCLUSION

We have evaluated the CSP-tree algorithm on various application datasets to observe its performance and time complexity. We analysed the results in line with the evaluation metrics one by one. The CSP-Tree dynamically achieves a considerable amount of reduced mining time. We have shown that despite additional insignificant tree restructuring cost, CSP-tree achieves a remarkable performance gain on an overall runtime. The evaluation also shows that CSP Tree algorithm has been able to utilize the exploit the memory assigned to it compared to FP Tree. This means when provided a limited amount space, it can still function efficiently and effectively.

Moreover, the CPS algorithm presents the fastest runtime performance in all cases. The algorithm finds the latest frequent patterns from the dataset, and requires less time as compare to current algorithm in all cases. As per the threshold, we compare our algorithm with existing algorithm and in all cases, it gives better performance for runtime.

CHAPTER SIX

6.0 CONCLUSIONS AND FUTURE WORK

Data stream being a continuous and changing sequence of data that constantly arrive at a system needs to be processed in near real-time. The dissemination and challenges involved on the data stream phenomenon has necessitated the development of diverse stream mining algorithms. Several studies emphasized on different approaches proposed to overcome the challenge of storing and processing of fast continuous and uninterrupted streams of data. Traditional data mining techniques require multiple scans on a timestamp of data, but that cannot be viable or doable for stream data applications as they require a single scan of the data.

The mining of Frequent Patterns from data streams poses many challenges.

i) First, it is not feasible to keep all streaming data in the memory.

ii) Second, mining algorithms need to process the arriving data in real time with one scan of data.

Iii) Third, the distribution of data varies over time, and hence analysis results need to be updated in real time.

This study builds on existing stream mining approaches by addressing the issue of resource awareness involved in developing a compact dynamic tree for finding frequent item set. The aim of this research work is, therefore, to develop a dynamic compact prefix-tree for mining frequent patterns in stream data in various domains and implementing a stream mining algorithm.

6.1 SUMMARY OF CONTRIBUTIONS

The primary contribution of this paper is that we propose a unified approach to improving mining capability by considering data dependency extensively in data mining. We adopted the frequent pattern mining as the a model, and use a middleware propagation for efficient inference, so as to clean the data, to infer missing values, this generally improves the mining results from a model that ignores data dependency. This research may also contribute to data mining practice with our investigations on some real-life applications.

We made the following contributions:

1. The Dynamic Compact Stream Pattern tree algorithm could be used in solving the problems of mining which support single database scan;

2. It could be used and applied in different industrial fields such as sensor network analysis and network monitoring system, where real time data are needed for processing;

3. It will add knowledge to the existing knowledge of data science where data mining algorithm is a requirement.

4. We addressed the problem of memory adaptive over the mining of item-sets from data streams.

5. We proposed an algorithm, called *Dynamic Compact Stream Pattern (DCSP)*, to discover Frequent Pattern over the entire data streams. *DCSP* is based on a dynamically achieves frequency descending prefix tree structure with only a single-pass over the data by applying tree restructuring techniques such as branch sort method.

Our experiments were conducted on both synthetic and real datasets. The results show an efficiency enhanced performance without missing any sequential patterns from the data stream.

6.2 RECOMMENDATION FOR FURTHER STUDIES & CONCLUSION

DCSP had the advantages of not requiring further pruning threshold and its performance was relatively stable over a e range of low-probability-item population. In particular, it outperformed most tree mining algorithms in comparison when the dataset contained few low-probability items. We argued that the previous approaches and the CSP approach were orthogonal to each other. And there can be further improvements if two tree mining approaches could be combined leading to a generally best overall performance.

This research elaborated on research challenges associated with data streams which emanated from real world publication as discussed in previous chapters. The discussed issues were illustrated by practical applications. The study of real-world problems mentioned in this research identified shortcomings of existing methodologies demonstrated previously unaddressed research challenges.

It will be ideal for the researchers in data stream mining to take into account the following action points;

. Considering the continuous availability of information by developing models that handle incomplete, and delayed feedback;

. developing a systematic methodology for streamed preprocessing;

. creating simpler models through multi-objective optimization criteria, which consider not only accuracy, but also computational resources, diagnostics, reactivity, interpretability.

. developing online monitoring systems, ensuring reliability of any updates, and balancing the distribution of resources.

Our research showed that , there are challenges in every step of the CSP algorithm process. As of recent, modeling data streams has been viewed and approached as an extension of traditional methods. However, our discussion from the application examples show that in many cases it would be beneficial to step aside from building upon existing mining approaches, and start blank considering what is required in the stream setting.

Primarily, there exists various tools and technologies for implementing data streams and there seems to be no data mining tool and technologies that offer key features required for now. While each tool and technology may have strengths and weaknesses, the choice depends on the aim and objective of the research and data made readily available. A decision in favour of the wrong technology may result in increased overhead cost and time. The provision should take into consideration factual analysis that comes along with the system requirement. Furthermore, earch efforts should also be directed to ways through which existing data streaming tools can be further improved and enhanced with technologies to provide key features such as scalability, integration, timeliness, consistency, erogeneity and incompleteness management.

BIBLIOGRAPHY

Aggarwal C (2009): On classification and segmentation of massive audio data streams. KnowledgeInformation System 20(2):137–156

Adnan Idrisa, Muhammad Rizwana & AsifullahKhan (2012):Churn prediction in Telecom using Random Forest and PSO based data balancing in combination with various feature selection strategies; Comput Electr Eng, http://dx.doi.org/10.1016/j.compeleceng.2012.09.001

Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer (2010); MOA: Massive online analysis. J. Mach. Learn. Res., 11:1601–1604,

Anuj .S, Prabin, D., Panigrahi, K., (2011): "A Neural Network based Approach for Predicting Customer Churn in Cellular Network Services", *International Journal of Computer Applications*; 27– No.11, 0975 – 8887.

Ascarza, .E, Ebbes, .P, Netzer, .O and Danielson, .M (2016): Beyond the Target Customer:
Social Effects of CRM Campaigns.

Arasu, .A, Babcock, .B, Babu, .S, Datar, .D, Ito, .K, Nishizawa, .I, Rosenstein, J and J. Widom (2003). STREAM: The stanford stream data manager. ACM SIGMOD, Demo

Babcock B, Babu S, Datar M, Motwani R,Widom J (2002) Models and issues in data stream systems.*In:Proceedings of 21st ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems*(PODS' 02), pp 1–16.

Bifet A, Holmes G, Pfahhringer B, Kirkby R, Gavalda R (2009). New Ensemble Methods for Evolving Data Streams. *In KDD '09: Proceedings of the 15th ACM SIGKDD International conference on Knowledge Discovery and Data Mining*, New York, NY, USA, ACM Press; pp. 139-148.

Bifet, Holmes. G, Kirkby, R., and Pfahringer, B (2010); MOA: Massive online analysis. J.
Machine.Learning. Res., 11:1601–1604.

Bifet,A.,Gavaldà,R (2008).:Miningadaptivelyfrequentclosedunlabeled rooted trees in data streams. In: Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining, pp. 34–42. Las Vegas, USA

Bingquan Huang, Mohand Tahar Kechadi & Brian Buckley (2012). Customers churn prediction in telecommunication; Expert Systems with Application. An International Journal Vol 39(1); Tarrytown, NY,USA,pg 1414-1425

Borja, B., Bernardino, C., Alex, Ricard Gavald`a, David Manzano-Macho (2013): The Architecture of a Churn Prediction System Based on Stream Mining.

Borgelt C., (2003)"Efficient Implementations of Apriori and Eclat," in 1st IEEE ICDM Workshop on Frequent Item Set, p. 9.

Breiman, L. (2001) Machine Learning 45: 5. https://doi.org/10.1023/A:1010933404324. *Kluwer Academic Publishers*. **ISSN:** 0885-6125.

Chakravarthy, S., and Jiang. Q (2009): Stream Data Processing: A Quality of Service Perspective: Modeling, Scheduling, Load Shedding, and Complex Event Processing *Advances in Database System* 36, DOI: 10.1007/978-0-387-71003-7 1.

Chandrasekharan, S, et al.2003 TelegraphCQ: Continuous dataflow processing for an uncertain world.
CIDR.

Charu C. Aggarwal (2007). DATA STREAMS: Models and Algorithms: *Springer IBM, T. J. Watson Research Center Yorktown Heights,* NY, USA, pp. 1-6, ISBN- 13: 978-0-387-28759- 1.

Chen Y., Guo J., Wang Y., Xiong Y., Zhu Y. (2007) Incremental Mining of Sequential Patterns Using Prefix Tree. In: Zhou ZH., Li H., Yang Q. (eds) Advances in Knowledge Discovery and Data Mining. PAKDD. Lecture Notes in Computer Science, vol 4426. Springer, Berlin, Heidelberg.

Charikar M, Chen K, Farach-ColtonM(2002) Finding frequent items in data streams. In: Proceedings of 29th international colloquium on automata, languages and programming, pp 693–70.

Chris Giannella, Jiawei Han, Jian Pei, Xifeng Yan, Philip S. Yu (2003); Mining Frequent Patterns in Data Streams at Multiple Time Granularities; Data Mining: Next Generation Challenges and Future Directions, AAAI/MIT.

Coussement Kristof, & Dirk Van den Poel (2008): Churn prediction in subscription services: An application of support vector machines while comparing two parameter-selection techniques. Expert Systems with Applications 34 313–327.

Cortes, .C, Fisher, .K, Pregibon, .D and Rogers, .A (2000). Hancock: A language for extracting signatures from data streams. KDD, 9–17

Cortec . C and Pregubon. D (2001): "SIgnature-Based Methods for Data Streams", Data Mining and Knowledge Discovery, *Kluver Academic Publishers*. (5) ; pp 167 - 182

David .H, Heikki M. & Padhraic S. (2001). “Principles of Data Mining”, ISBN: 026208290 MIT Press, Cambridge, MA.

Desamparados Blazquez, Josep Domenech (2016); Big Data sources and methods for social and economic analyses. Technological Forecasting & Social Change. Department of Economics and Social Sciences, Universitat Politècnica de València, Camí de Vera s/n., Valencia 46022, Spain.

Fang, R. and Tuladhar, S. (2006). “Teaching Data Warehousing and Data Mining in a Graduate Program in Informa-tion Technology,” *Journal of Com- puting Sciences in Colleges*, Vol. 21, Issue 5, pp. 137-144.

Gantz J, Reinsel D (2012). The digital universe in 2020: big data, bigger digital shadows, and biggest

growth in the Far East. New York: IDC iView: IDC Analyse future.

George. D and Shuqin, C (2014): A Hybrid Churn Prediction Model in Mobile Telecomunication

Industry. I*nternational journal of e-Educatio, e-Management and e-Learning,* (4) No 1, pp 55-60.

Guha. S, Adam M, Nina M, Rajeev Motwani, Member, IEEE, and Liadan O'Callaghan (2003): Clustering Data Streams: Theory and Practice; *IEEE Transactions on Knowledge and Data Engineering,* VOL. 15, NO. 3, pg 515-525.

Guha. S, Rajeev R & Kyuseok S (2000): ROCK: A ROBUST CLUSTERING ALGORITHM FOR

CATEGORICAL ATTRIBUTES. Information Systems Vol. 25, No. 5, pp. 345{366, 200}.

Haiqing Li, Lang Wang (2017): A Variable Size Sliding Window Based Frequent Itemsets Mining Algorithm in Data Stream. College of College of Computer Science and Technology, Chongqing University of Posts and Telecommunications, Chongqing 400065, China.

Haixun W, Wei F, Philip S. Yu, Jiawei Han (2003): Mining Concept-Drifting Data Streams Using Ensemble Classifiers; In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '03.* pp. 226-235. DOI:10.1145/956750.956778.

Hossein, A., Mostafa, S., and Mohammad T., (2014): "A Comparative Assessment of the Performance of Ensemble Learning in Customer Churn Prediction"; *The International Arab Journal of Information Technology*, (11)6, pp 599-606.

Howard, R. (1988) "Decision Analysis: Practice and Promise." Management Science 34, 6: 679–695.

Huang, b.Q , Kechadi, T.-M, Buckley, .B ,Kiernan .G , Keogh, .E , Rashid, T (2010):"A new feature set with new window techniques for customer churn prediction in land-line telecommunications", *Expert Systems with Applications*, 37, pp. 3657–3665.

Intel IT (2012), IT Best Practices Business Intelligence Retrieved from http://www.intel.com/it/

Karp RM, Papadimitriou CH, Shenker S (2003) A simple algorithm for finding frequent elements in streams and bags. ACM Trans Database Syst, 28:51–55.

Keramati .A, Seyed M.S, and Ardabili (2011) "Churn analysis for an Iranian mobile operator", Telecommunications Policy 35 344–356

Kfihn, O., & Abecker, A. (1977). Corporate memories for knowledge management in industrial practice: Prospects and challenges. *Journal of Universal Computer Science*, 3(8) URL: http://www.iicm.edu/jucs-3-8/corporate-memories-for-knowledge.

Khan, A.A, Jamwal, .S, and Sepehri, M.M. (2010): "Applying Data Mining to Customer Churn Prediction in an Internet Service Provider", *International Journal of Computer Applications*; 9(7). 0975– 8887.

Kifer, D., Ben-David, S., Gehrke, J (2004).: Detecting change in data streams. In: Proceedings of the International Conference on Very Large Data Bases, pp. 180–191. Morgan Kaufmann, Toronto

Kisioglu .P and Topcu .Y (2011) "Applying Bayesian Belief Network approach to customer churn analysis: A case study on the telecom industry of Turkey", Expert Systems with Applications; 38, 7151–7157.

Korn, F, Muthukrishnan, F & Wu,Y : Modeling skew in data streams. In Proc. ACM SIGMOD Int.

Conf. on Management of Data, pages 181-192, 2006.

Koutri, M., Avouris, N., & Daskalaki, S. (2004). Ch. A survey on web usage mining techniques for web-based adaptive hypermedia systems.

Kristof Coussement, Dirk Van den Poel (2008): Improving customer attrition prediction by integrating emotions from client/company interaction emails and

evaluating multiple classifiers. Ghent University, *Faculty of Economics and Business Administration, Department of Marketing*, Tweekerkenstraat 2, B-9000 Ghent, Belgium; (36) pg 6127–6134

Lavrac, N., Motoda, H., Fawcett, T., Holte, R., Langley, P. & Adriaans, P. (2004). Introduction: Lessons Learned from Data Mining Applications and Collaborative Problem Solving. *Machine Learning* 57(1-2): 13-34.

Lin C, Chiu D,Wu Y, Chen A (2005) Mining frequent itemsets from data streams with a time- sensitive sliding window. *In: Proceeding of the 2005 SIAM international conference on data mining* (SDM'05), Newport Beach, pp 68–79

Leo Breiman Statistics Department University of California Berkeley, CA 94720

Luan, J. (2002). Data mining, knowledge management in higher education, potential applications. In

Workshop associate of institutional research international conference, Toronto, 1–18.

Gurmeet Singh Manku and Rajeev Motwani; 2002; Approximate Frequency Counts over DataStreams; Int'l Conf. on Very Large Databases; 2.

Mahnoosh K, and Mohammadreza, .K (2011): An Analytical Framework for Data Stream Mining Techniques based on challenges and requirements; *International Journal of Engineering Science and Technology (IJEST)*, (3)3. pp 2508: ISSN : 0975-5462.

Martens D., Van T., and Baesens B., (2009); "Decompositional Rule Extraction from Support Vector Machines by Active Learning," IEEE Transactions on Knowledge and Data Engineering, vol.

21, no. 2, pp. 178-191. .

Martin, O. (2012). Between the Ideal and the Ordeal: The Business of Data in Nigeria's Corporate Competitiveness, *ValueFronteira Limited*,The Frontier Post, 3

Masud, M. M., Al-Khateeb, K., Khan, L., Aggarwal, C., Gao, J., Han, J., and Thuraisingham, B. (2011). Detecting recurring and novel classes in concept-drifting data streams. In ICDM, pp.1176-1181

Metwally A, Agrawal D, El Abbadi A (2005) Efficient computation of frequent and top-k elementsin data streams. In: Proceeding of the 2005 international conference on database theory (ICDT'05), Edinburgh, UK, pp 398–412.

Mozer. M.C. ; R. Wolniewicz ; D.B. Grimes ; E. Johnson ; H. Kaushansky (2000): Predicting subscriber dissatisfaction and improving retention in the wireless telecommunications industry. *IEEE Transactions on Neural Networks:*

Owczarczuk, M. (2010) Churn Models for Prepaid Customers in the Cellular Telecommunication Industry Using Large Data Marts. *Expert Systems with Applications*, 37, 4710-4712. http://dx.doi.org/10.1016/j.eswa.2009.11.083

Parveen, P., Evans, J., Thuraisingham, B., Hamlen, K., and Khan, L. (2011). Insider threat detection using stream mining and graph mining, Privacy, Security, Risk and Trust (PASSAT), *IEEE Third International Conference on and 2011 IEEE Third International Conference on Social Computing (SocialCom)*, 1102–1110.

Parveen, P., Evans, J., Thuraisingham, B., Hamlen, K., and Khan, L. (2011). Insider threat detection using stream mining and graph mining. In Proceedings of the 3rd IEEE Conference on Privacy, Security, Risk and Trust (PASSAT) MIT, Boston, USA. (acceptance rate 8%) (Nominated for Best Paper Award).

Parveen, P., McDaniel, N., Evans, J., Thuraisingham, B., Hamlen, K., and Khan, L. (2013). Evolving insider threat detection stream mining perspective. *International Journal on Artificial Intelligence Tools (World Scientific Publishing*) 22 (5), 1360013-1-1360013-24.

Qian ZP, He Y, Su CZ et al (2013). TimeStream: Reliable stream computation in the cloud. In: Proc. 8th

ACM European conference in computer system, EuroSys 2013. Prague: ACM Press;. p. 1–4

Quinlan, J. (1986). 'Induction of Decision Trees'. *Machine Learning, 1*, 81-106.

Rashid T.A. (2016) Convolutional Neural Networks based Method for Improving Facial Expression Recognition. In: Corchado Rodriguez J., Mitra S., Thampi S., El-Alfy ES. (eds) *Intelligent Systems Technologies and Applications 2016*. ISTA 2016. Advances in Intelligent Systems and Computing, vol 530. Springer, Cham

Shim .B, Choi .K and Suh .Y (2012): "CRM strategies for a small-sized online shopping mall based on association rules and sequential patterns*", Expert Systems with Applications;* 39, 7736– 7742.

Srinivasan Parthasarathy, and Wei Li Mohammed Javeed Zaki, (1997) "A Localized Algorithm for Parallel Association Mining," in In 9th ACM Symp. Parallel Algorithms & Architectures. .

Tsai, .C.H and Lu, Y.H (2009): Data Mining Techniques in Customer Churn Prediction; Recent Patents on Computer Science, *Bentham Science Publishers*, 3, 28-32.

Vivek Bhambri, (2013). Effective use of Data Mining in Banking. *International journal of engineering sciences & research*, ISSN: 2277-9655 Scientific Journal Impact Factor: 3.449 (ISRA), Impact Factor: 1.85 TECHNOLOGY

Umman Tuğba & Şimşek Gürsoy (2010): Customer churn analysis in telecommunication sector. *Istanbul University Journal of the School of Business Administration* Cilt/Vol:39, Sayı/No:135-49 ISSN: 1303-1732.

Xie Yaya; Li, Xiu, Ngai, E.W.T. Ngai & Ying Weiyun (2009): Customer Churn Prediction Using Improved

Balanced Random Forests, Expert Systems with Applications 36,: 5445–5449

www.oracle.com

Yoon Koehler & Ghobarah 2010: Prediction of advertisers Churn for Google AdWorks JSM

Proceeding, American Statistical Association.

Zhen-Yu Chen , Zhi-Ping Fan , Minghe Sun, "A hierarchical multiple kernel support vector machine for customer churn prediction using longitudinal behavioral data", European Journal of Operational Research, 223, 2012, 461–472

Zorrilla, M. E., Menasalvas, E., Marin, D., Mora, E., & Segovia, J. (2005).Web usage mining project for improving web-based learning sites. In *Web mining workshop*, Catalun.

APPENDIX

Time Complexities of Functions in the CSP Algorithm code

CSP algorithm code is made up of seven functions/classes. To calculate the entire complexity of the code, we had to calculate the individual time complexities of the functions that make up the code, and then add them up.

ndow Class

```
m Pane import Pane

ss Window:
 ef __init__(self, size, pane_size):
   self.size = size
   self.dataStartMark = 0
   self.slideSize = 3 + size // pane_size
   self.dataEndMark = size * pane_size
   self.trackTable = {}
   self.frequentItem = {}
   self.sentinel = {}
   self.paneSize = pane_size
   self.container = []
   self.sentinel["count"] = 0
```

e Complexity: **N(N +1) +2 [O(N^2)]**

```
insert_pane(self, pane):
  if len(self.container) < self.size:
      self.container.append(pane)
      for trans in pane.transactions for item in trans:
```

```
            self.trackTable[item]   =   self.trackTable.get(item,   0)   +
  pane.transactions[trans] #self.tree.
 e Complexity: N +1    [ O(N) ]
f fill(self, panes):
  for pane in panes:
     self.container.append(pane)

 e Complexity: N + 15 [ O(N) ]
f slide(self, data_set):
  self.pop_oldest_pane()
  count = 0
  pane = None
  rem_data = data_set[self.sentinel["count"]:]
#   Supplied data_set has changed totally
#   if self.sentinel["data"] != data_set[self.sentinel["count"]]:
#      return

#   The dataset remain is not slideable
  if len(rem_data) < self.paneSize * self.slideSize:
     return

  for x in rem_data:
     if count >= self.paneSize * self.slideSize:
        self.sentinel["data"] = x
        if pane is not None:
```

```
            self.insert_pane(pane)

          break

      if count % self.paneSize == 0:
      if pane is not None:

            self.insert_pane(pane)

          pane = Pane(self.paneSize, (count // self.paneSize) + 1)

      pane.insert_transaction(frozenset(x))

      count += 1

      self.sentinel["count"] += 1

  self.dataStartMark += self.slideSize * self.paneSize


 e Complexity: N + 4 [ O(N) ]

f pop_oldest_pane(self):

  #for i in range(self.slideSize):

#       pane = self.container[i].remove_transaction_from_tree(tree)

#       self.container.pop(i)

  self.dataEndMark += self.slideSize * self.paneSize

  return
```

table Class

```
m bisect import bisect

m Queue import Queue


ss Sortable(Queue):

 ef __init__(self, f=lambda x: x):

  self.A = []
```

```
  self.f = f


 e Complexity: 1 [ O(1) ]

ef append(self, item):

  bisect.insort(self.A, (self.f(item), item))
  e Complexity: 1 [ O(1) ]

ef __len__(self):

  return len(self.A)


 e Complexity: 2 [ O(2) ]

f pop(self):

  if self.order == min:

      return self.A.pop(0)[1]

  else:

      return self.A.pop()[1]


 e Complexity: 1 [ O(1) ]

ef __contains__(self, item):

  return any(item == pair[1] for pair in self.A)

 e Complexity: N + 2 [ O(N) ]

ef __getitem__(self, key):

  for _, item in self.A:

      if item == key:

          return item

 e Complexity: N + 2 [ O(N) ]
```

```
ef __delitem__(self, key):
  for i, (value, item) in enumerate(self.A):
      if item == key:
          self.A.pop(i)
```

eue Class

```
ss Queue:
 e Complexity: 1 [ O(1) ]
 ef __init__(self):raise NotImplementedError

 e Complexity: N + 1 [ O(N) ]
 ef extend(self, items):
  for item in items:
      self.append(item)
```

ne Class

```
ss Pane:

 e Complexity: 3 [ O(3) ]
 ef __init__(self, size, age):
  self.size = size
  self.age = age
  self.transactions = {}

 e Complexity: 1 [ O(1) ]
 insert_transaction(self, transaction):
  self.transactions[frozenset(transaction)] = 1
```

e Complexity: **N + 1 [O(N)]**

fill(self, transactions):

  for transaction in transactions:

    self.insert_transaction(transaction)

e Complexity: **1 [O(1)]**

f remove_transaction_from_tree(self, tree):

  pass

de Class

ss Node:
""A node in a search tree. Contains a pointer to the parent (the nodehat this is a successor of) and to the actual state for this node. """

e Complexity: **11 [O(11)]**

__init__(self, value, support=0, parent=None, tail=False, pane=None, window=None, pane_counter=None):

  """Create a search tree Node, derived from a parent by an action."""

  self.parent = parent

  self.support = support

  self.value = value

  self.is_tail = tail

  self.pane = pane

  self.age = 0

  self.depth = 0

  self.pane_counter = pane_counter

  self.children = {}

```
  if parent is not None:
      self.depth = parent.depth + 1
```

e Complexity: **1 [O(1)]**

```
f inc(self, num_occur):
  self.support += num_occur
```

e Complexity: **1 [O(1)]**

```
ef dec(self, num_dec):
  self.support -= num_dec
```

e Complexity: **N + 5 [O(N)]**

```
ef processed_according_to(self, i_list):
  for child in self.children.items():
if len(child[1].children) == 0:if i_list[child[1].value][0] > i_list[self.value][0]:
            return False
      else:
         child[1].processed_according_to(i_list)
  return True
```

e Complexity: **3 [O(3)]**

```
add_child(self, node):
  node.parent = self
  if node.value in self.children:
      self.children[node.value].support += node.support
  else:
```

```
    self.children[node.value] = node
```

e Complexity: **N + 5 [O(N)]**

```
search(self, node):
  for child in self.children:
    if len(child.children) == 0:
      if child == node:
        return True
    else:
      child.search(node)
  return False
```

e Complexity: **N + 4 [O(N)]**

```
tails(self):
  ret = []
  for child in self.children:
    if len(child.children) == 0:
      ret.append(child)
    else:
    ret.extend(child.tails())
  return ret
```

e Complexity: **8 [O(8)]**

```
swap(self, node):
  if node in self.children:
    self.children.pop(node.value)
    node.parent = self.parent
```

```
      self.parent = node
      node.children = self.children
  if node == self.parent:
      self.parent = node.parent
      node.children.pop(self.value)
      self.children[node.value] = node
      node.parent = self
```

e Complexity: **N + 2 [O(N)]**

```
disp(self, ind=1):
  print (' '*ind, self.value, ' ', self.support)
  for child in self.children.values():
      child.disp(ind+1)
```

e Complexity: **1 [O(1)]**

```
__repr__(self):
  return "<Node {}>".format(self.disp())
```

e Complexity: **1 [O(1)]**

```
__lt__(self, node):
  return self.support < node.support
```

e Complexity: **1 [O(1)]**

```
__gt__(self, node):
  return self.support > node.support
```

e Complexity: **1 [O(1)]**

```
solution(self):
   """Return the sequence of actions to go from the root to this node."""
   return [node.action for node in self.path()[1:]]
```

e Complexity: **5 [O(5)]**

path(self):

"""Return a list of nodes forming the path from the root to this node."""

node, path_back = self, []

while node:

path_back.append(node)

node = node.parent

return list(reversed(path_back))

We went for a queue of nodes in breadth_first_search or a star_search to have no duplicated states, so we treat nodes with the same state as equal.

e Complexity: **1 [O(1)]**

__eq__(self, other):

return isinstance(other, Node) and self.value == other.value

e Complexity: **1 [O(1)]**

__hash__(self):

return hash(self.value)

S Tree Class

m Node import Node

m Sortable import Sortable

ss CPSTree:
e Complexity: **10 [O(10)]**

__init__(self, window_size, pane_size, slide_size):

self.nodeList = Sortable()

self.slideSize = slide_size

```
  self.paneSize = pane_size
  self.fp_tree = None
  self.transactions = {}
  self.trackTable = {}
  self.frequentItem = {}
  self.dataSet = {}
  self.treeSet = []
  self.windowSize = window_size
```

e Complexity: **(N + 3)(N + 2) + 3 [O(N^2)]**

```
create_cps_tree(self, data_set, freq_item, track_table):
  self.trackTable = track_table
  self.frequentItem = freq_item
  tree = Node("Null Node", 1)
  for tranSet, count in data_set.items():
     local_d = {}
     for item in tranSet:
        if item in freq_item:
           local_d[item] = track_table[item][0]
     if len(local_d) > 0:
        ordered_items = [v[0] for v in sorted(local_d.items(), key=lambda p:
        p[1], reverse=True)] self.add_to_tree(ordered_items, tree, count) #
        populate tree with ordered freq itemset
  self.fp_tree = tree
  return self.restructure_tree()
```

e Complexity: **N(N + 3) + 1 [O(N^2)]**

```
f restructure_tree(self):
  i_sort = {v[0]: v[1] for v in sorted(self.trackTable.items(), key=lambda p:
  p[1], reverse=True)} for child in self.fp_tree.children.items():
      for branch in child[1].children.items():
          print(branch[1].processed_according_to(i_sort))
          if not branch[1].processed_according_to(i_sort):
              self.process_branch(branch, i_sort)
          else:
              self.sort_path(branch)
  return self.fp_tree
```

e Complexity: **N + 8 [O(N)]**

```
process_branch(self, branch: Node, i_sort: dict):


for   tail
in
branch.t
ails():
      parent = tail.parent
      if i_sort[tail.value] > i_sort[parent.value]:
          if tail.support < parent.support:
#               Attaching new node to the parent parent joining_node = Node(parent.value,
tail.support) parent.parent.add_child(joining_node)
#               Reset support counts
              parent.support -= tail.support
```

```
#           Attaching tail to the new node joining_node.add_child(tail) del
(parent.children[tail.name])
#           Swap the nodes
            joining_node.swap(tail)
        else:
            parent.swap(tail)
```
e Complexity: **1 [O(1)]**

```
f sort_path(self, branch):
  pass
```

e Complexity: **3 [O(3)]**

```
add_to_tree(self, items, in_tree, count):
  if items[0] in in_tree.children:
    in_tree.children[items[0]].inc(count)
  else:
    in_tree.children[items[0]] = Node(items[0], count, in_tree)
  if len(items) > 1:
    self.add_to_tree(items[1::], in_tree.children[items[0]], count)
```

e Complexity: **3 [O(3)]**

```
remove_from_tree(self, items, count):
  if items[0] in self.fp_tree.children:
    self.fp_tree.children[items[0]].dec(count)
  else:
    self.fp_tree.children[items[0]] = Node(items[0], count, self.fp_tree)
  if len(items) > 1:
    self.add_to_tree(items[1::], self.fp_tree.children[items[0]], count)
```

e Complexity: **2 [O(2)]**

updateTrackTable(self, node_to_update, target_node):

while (node_to_update.nodeLink != None):

node_to_update = node_to_update.nodeLink

node_to_update.nodeLink = target_node

pMain2 CLass

import csv
import sys

from CPSTree import CPSTree

from Sortable import Sortable

from Window import Window

import time

ss AppMain2: #

e Complexity: **17 [O(17)]**

__init__(self, window_size, pane_size, slide_size, data_set, min_sup=1): #Definatio

n of class AppMain

ibutes

self.nodeList = Sortable()

self.dataSet = data_set

self.localData = {} # Dictionary localData

self.slideSize = slide_size

```
  self.paneSize = pane_size
  self.min_sup = min_sup
  self.transactions = {} #Dictionary transactions
  self.trackTable = {}       #Dictionary trackTable
  self.frequentItem = {} #Dictionary frequentItem
  run_time2 = []
  run_time = []
  self.dataSet = data_set
  self.windowSize = window_size
  self.window = Window(self.windowSize, self.paneSize)
  self.prepare_dataset()
  self.cps_tree = CPSTree(window_size, pane_size, slide_size)
  self.tree              =              self.cps_tree.create_cps_tree(self.localData,
  self.frequentItem, self.trackTable) return
```

e Complexity: N(**N + 1) + (N + 11) + 3 [O(N^2)]**

```
prepare_dataset(self): #Definition of method prepare_dataset
  self.localData                                                              =
  self.create_init_set(self.dataSet[self.window.dataStartMark:self.window.dataEn
  dMark]) for trans in self.localData: # first pass counts frequency of occurance
     for item in trans:
        self.trackTable[item] = self.trackTable.get(item, 0) + self.localData[trans] #Adds
        up all similar items in
aset
  start = time.time()
```

```
  for k in list(self.trackTable): #Converts self.trackTable
      from a dictionary to list run_time = []
      run_time2 = []
      if self.trackTable[k] < self.min_sup: #allows only items who is equal or
greater than the minimum port to be in the list.
          del    (self.trackTable[k])    #Deletes
      items less than min support else:
          self.trackTable[k] = [self.trackTable[k], None]


          run_time2.append(k)
          end = time.time()
          run_time2 = end - start
          run_time.append(run_time2)
          run_time2 = []


  self.frequentItem = set(self.trackTable.keys()) #Creates a set of frequent item key
  wihout their values
#   ranger = -len(self.dataSet) if len(self.dataSet) < self.paneSize * self.windowSize
else
#   -self.paneSize * self.windowSize
  print(self.trackTable)
  print(self.frequentItem)
  print(run_time)
  @staticmethod
 e Complexity: N + 4 [ O(N) ]
 ef create_init_set(data_set):
```

```
  ret_dict = {}
  for trans in data_set:
      if frozenset(trans) in ret_dict.keys():
          ret_dict[frozenset(trans)] += 1
      else:
          ret_dict[frozenset(trans)] = 1
  return ret_dict
```

e Complexity: **5 [O(5)]**

```
reload_dataset(self, data_set):
#    If dataset has not changed since the last loading if len(self.dataSet) == len(data_set):
       return
#    new_data = data_set[len(self.dataSet):]
#    If new data is not up to what the algorithm can slide through
  if self.slideSize * self.paneSize > len(data_set[len(self.dataSet):]):
      return
  self.dataSet = data_set
  self.refresh()
```

e Complexity: **5 [O(5)]**

```
refresh(self):
  # data are no more available in the dataset to slide into
  if self.slideSize * self.paneSize > len(self.dataSet[self.window.dataEndMark + 1:]):
      return
  else:
      self.window.slide(self.dataSet)
      self.trackTable = {}
      self.frequentItem = {}
```

```
    self.prepare_dataset()

    self.tree = self.cps_tree.create_cps_tree(self.localData, self.frequentItem,
    self.trackTable)
```

e Complexity: **5 [O(3)]**

```
ef walk_tree(self, node, prefix_path): # ascends from leaf node to root if node.parent
is not None:
    prefix_path.append(node.value)

    self.walk_tree(node.parent, prefix_path)
```

e Complexity: **6 [O(6)]**

```
ef prefix_path(self, node): # treeNode comes

 from header table condPats = {}

 while node is not None:

    prefix_path = []

    self.walk_tree(node, prefix_path)

    if len(prefix_path) > 1:

       condPats[frozenset(prefix_path[

    1:])] = node.support node =

    node.nodeLink

 return condPats
```

```
_name__ == '__main__':

ata_one = [['r', 'z', 'h', 'j', 'p'],

  ['z', 'y', 'x', 'w', 'v', 'u', 't', 's'],

  ['y', 'r', 'x', 'z', 'q', 't', 'p'],

  ['z', 'y', 'x', 'w', 'v', 'u', 't', 's'],
```

```
    ['y', 'r', 'x', 'z', 'q', 't', 'p'],
    ['y', 'r', 'x', 'z', 'q', 't', 'p'],
    ['z', 'y', 'x', 'w', 'v', 'u', 't', 's'],
    ['y', 'r', 'x', 'z', 'q', 't', 'p'],
    ['z'],
    ['r', 'x', 'n', 'o', 's'],
    ['y', 'r', 'x', 'z', 'q', 't', 'p'],
    ['z'],
    ['r', 'x', 'n', 'o', 's'],
    ['y', 'r', 'x', 'z', 'q', 't', 'p'],
    ['z'],
    ['r', 'x', 'n', 'o', 's'],
    ['y', 'r', 'x', 'z', 'q', 't', 'p'],
    ['r', 'x', 'n', 'o', 's'],
    ['y', 'r', 'x', 'z', 'q', 't', 'p'],
    ['z'],
    ['r', 'x', 'n', 'o', 's'],
    ['y', 'r', 'x', 'z', 'q', 't', 'p'],
    ['y', 'z', 'x', 'e', 'q', 's', 't', 'm']]
ata_two = [['r', 'z', 'h', 'j', 'p'],
    ['z', 'y', 'x', 'w', 'v', 'u', 't', 's'],
    ['y', 'r', 'x', 'z', 'q', 't', 'p'],
    ['z', 'y', 'x', 'w', 'v', 'u', 't', 's'],
    ['y', 'r', 'x', 'z', 'q', 't', 'p'],
    ['y', 'r', 'x', 'z', 'q', 't', 'p'],
```

['z', 'y', 'x', 'w', 'v', 'u', 't', 's'],

['y', 'r', 'x', 'z', 'q', 't', 'p'],

['z'],

['r', 'x', 'n', 'o', 's'],

['y', 'r', 'x', 'z', 'q', 't', 'p'],

['z'],

['r', 'x', 'n', 'o', 's'],

['y', 'r', 'x', 'z', 'q', 't', 'p'],

['z'],

['r', 'x', 'n', 'o', 's'],

['y', 'r', 'x', 'z', 'q', 't', 'p'],

['r', 'x', 'n', 'o', 's'],

['y', 'r', 'x', 'z', 'q', 't', 'p'],

['z'],

['r', 'x', 'n', 'o', 's'],

['y', 'r', 'x', 'z', 'q', 't', 'p'],

['y', 'z', 'x', 'e', 'q', 's', 't', 'm'],

['r', 'x', 'n', 'o', 's'],

['y', 'r', 'x', 'z', 'q', 't', 'p'],

['r', 'x', 'n', 'o', 's'],

['y', 'r', 'x', 'z', 'q', 't', 'p'],

['z'],

['r', 'x', 'n', 'o', 's'],

['y', 'r', 'x', 'z', 'q', 't', 'p'],

['y', 'z', 'x', 'e', 'q', 's', 't', 'm']]

```
pp_main = AppMain2(3, 10, 10, data_one, 1)
print(app_main.tree)
app_main.refresh()
print(app_main.tree)
app_main.refresh()
print(app_main.tree)
app_main.refresh()
print(app_main.tree)
app_main.reload_dataset(data_two)
app_main.refresh()
print(app_main.tree)
 input = pandas.read_csv(sys.argv[1])
 column = sys.argv[2]
 duram = input.column
```

e Complexity: **N + 48 [O(N)]**

in Program

```
 ort pandas as pd
m AppMain import AppMain
 ort random as random
 ort time
 ort dis
 ort math, sys
m sys import getsizeof
m memory_profiler import profile
```

```
t = time.time()
profile

dow_size = int(input('Input the size of your window: '))
e_size = int(input('Input the size of your pane: '))
 _sup = int(input('Input the minimum support: '))

=  pd.read_csv("C:/Users/Silas/Desktop/phd project/moviedata.csv")

.dropna(axis=0)  #  Drop  rows
with missing values a_set = []
a_set2 = []

 s_length = []
 _of_pane = []
 _of_trans = []
 s_size = []

nt = 0;
 of_trans = 0;

 of_items_in_a_transaction = random.randrange(100, 200, 2)
```

```
his Part of the program produces the

transactions from the he dataset'''


line in file['Movie']:

ount = count + 1

ata_set2.append(line)

f count % no_of_items_in_a_transaction == 0:

  no_of_items_in_a_transaction = random.randrange(100, 200, 2)

  data_set.append(data_set2)

  data_set.append(',')

  trans_length.append(len(data_set2))

  trans_size.append(sys.getsizeof(data_set2))

  data_set2 = []

  no_of_trans = no_of_trans + 1


of_trans_in_pane = 0;


of_trans_in_pane = len(data_set)//(pane_size * window_size)

e_size = 0;

e_size = len(data_set)//(window_size)

nt ('The number of items are :' , count)

nt ('The number of items in a transaction are :' , no_of_items_in_a_transaction)

nt ('The average number of transactions in a pane are :' , no_of_trans_in_pane)

nt ('The number of transactions are :' , no_of_trans)

nt (trans_length)

nt (size_of_trans)
```

```
x_trans_length = max(trans_length)

nt('The maximum transactions length is:' , max_trans_length)

 _trans_length = min(trans_length)

nt('The minimum transactions length is:' , min_trans_length)

 rage_translength = (count//no_of_trans)

nt('The average transaction length is :', average_translength)
nt('ATL/I x 100 = ', (average_translength/count)*100)
nt('The datasize of panes is :', sys.getsizeof(data_set[:no_of_trans_in_pane]))
nt('The datasize of data_set is :', sys.getsizeof(data_set))
int('The datasize of transactions is :', sys.getsizeof(data_set))

 _main = AppMain(window_size, pane_size, 3 , data_set, min_sup)
 nt(app_main.tree)
=   time.time() nt(end - start)
s.dis(AppMain)
 p_main.refresh()
int(app_main.tree)
 p_main.refresh()
int(app_main.tree)
```

p_main.refresh()

int(app_main.tree)

I A) Answers to some questions on the result

Question 1: Reason for using a particular min_sup?

ng minimum criteria of 1000 means that the only items that are counted as frequent item must have a frequency ater than or equal to the minimum support. More of the items on the database are having a frequency above 0.

Question 2: Reason for converting them to transactions?

CSP algorithm reads patterns that are present in transactions. Therefore, the dataset are converted from mere of items to a list of transactions.

Question 3: Why random selection?

effectiveness of the algorithm is independent of whether it is time-based or random selection.

Question 4: How effective is the algorithm?

can determine the effectiveness of the CPS algorithm by running it over [different sets of databases].

Question 5: How the CPS algorithm works?

, 'z', 'h', 'j', 'p'], ['z', 'y', 'x', 'w', 'v', 'u', 't', 's'], ['y', 'r', 'x', 'z', 'q', 't', 'p'], ['z', 'y', 'x', 'w', 'v', 'u', 't', 's'], ['y', 'r', 'x', 'z', 't', 'p'], ['y', 'r', 'x', 'z', 'q', 't', 'p'], ['z', 'y', 'x', 'w', 'v', 'u', 't', 's'], ['y', 'r', 'x', 'z', 'q', 't', 'p'], ['z'], ['r', 'x', 'n', 'o', 's'], ['y', x', 'z', 'q', 't', 'p'], ['z'], ['r', 'x', 'n', 'o', 's'], ['y', 'r', 'x', 'z', 'q', 't', 'p'], ['z'], ['r', 'x', 'n', 'o', 's'], ['y', 'r', 'x', 'z', 'q', 't', 'p'], 'x', 'n', 'o', 's'],

['y', 'r', 'x', 'z', 'q', 't', 'p'], ['z'], ['r', 'x', 'n', 'o', 's'], ['y', 'r', 'x', 'z', 'q', 't', 'p'], ['y', 'z', 'x', 'e', 'q', 's', 't',].

dataset above consists of 23 transactions. The main CPS code is ‘app_main = AppMain(2, 12, 1, data_one, 1)’. s line of code uses a window containing 2 panes, with each pane containing 12 transactions. This covers theire 23 transactions present in data_one. Using a minimum support of 1, it captures all the elements present in a_one i.e {'y', 'm', 'v', 'n', 'h', 'o', 's', 'p', 'w', 'z', 't', 'q', 'x', 'u', 'r', 'e', 'j'}as the ‘most frequent item’.

Question 6: Give a simple analysis on the movie dataset?

table A: Further analysis on movie datasets

Dataset	#Trans. (T)	#Items (I)	MaxTL (MTL)	AvgTL (ATL)	(ATL/I)/ 100
Movie Data	170 66	36645 3	38	2 1	0.0057061 2

From the table A above, since the value of (ATL/I x 100) is less than 10 for movie dataset, then the dataset is rse.

The movie dataset has 637 different locations. It has 13 genres with comedy with the highest frequency of 101701 animation with the lowest frequency of 87. It is rated from 0.0 to 6.0 with 3.5 having the highest frequency of 45and 6.0 having the least frequency of 5.

The Explanation of Movie Data Analysis Result

e 1:

window Size: 2

number of Pane: 10

minimum support: 1000

number of items in movie dataset: 366453

number of items in a transaction: 36

number of transactions in a pane: 1703

number of transactions: 17033

maximum transactions length is: 38

minimum transactions length is: 2

average transaction length is : 2L/I x 100 = 0.005730612111239368 (Since the value of (ATL/I x 100) is less than 10 for movie dataset, then dataset is sparse)

a size of panes is : 6848Kb

asize of data_set is : 142696Kb

a minimum support of 1000, here are the movies whose frequency is more than 1000 and that form a pattern(s) ny of the transaction set. The number after the movie title is the frequency of the movie.

ere Comes the Boom': [1014, None], 'Seven Psychopaths': [1251, None], 'Hobbit; The An Unexpected Journey':

47, None], 'Silver Linings Playbook': [2340, None], 'Jack Reacher': [3793, None], 'Impossible; The': [2774,

ne], 'Pitch Perfect': [2817, None], 'A Good Day to Die Hard': [1382, None], 'I Give It A Year': [1515, None],

2 Dalmatians (Disney)': [1738, None], ',': [17033, None], 'Up (2009) (Disney)': [1438, None], 'Despicable Me

ctureBox)': [5131, None], 'Brother Bear (Disney)': [1702, None], 'Your Highness (PictureBox)': [1447, None],

st And The Furious; The (PictureBox)': [1213, None], 'Snatch (2000) (Sony Pictures)_BBFC': [1937, None],

antis: The Lost Empire (Disney)': [2068, None], 'Wreck it Ralph': [1341, None], 'Cars 2 (Disney)': [1940,

ne], 'Emperors New Groove; The (Disney)': [1558, None], 'Eagle; The (PictureBox)': [4506, None], 'Seed Of

ucky (PictureBox)': [1306, None], 'Life Of Pi': [3000, None], 'Brother Bear 2 (Disney)': [1272, None], 'Quartet':

56, None], 'Argo': [1865, None], '13 Ghosts (2001) (Sony Pictures)_BBFC': [2322, None], 'Rise of the

ardians': [1193, None], 'Django Unchained': [2372, None], 'Flight (Before DVD!)':[1110, None],

ventureland (Miramax)_BBFC': [1074, None], 'Stand By Me (Sony Pictures)_BBFC': [1001, None], 'Bone

lector; The (Sony Pictures)_BBFC': [1677, None], '2 Fast 2 Furious (PictureBox)': [1503, None], 'Bourne

imatum; The (PictureBox)': [1494, None], 'Few Good Men; A (Sony Pictures)_BBFC': [1164, None],

nfessions Of A Shopaholic (Disney)': [2327, None], 'Aristocats; The (Disney)': [1974, None], 'Surrogates

sney)': [1152, None], 'Hollow Man (Sony Pictures)_BBFC': [1454, None], 'Far And AwayctureBox)_BBFC': [1711, None], 'Don Quixote (PictureBox)': [1104, None], 'Accepted (PictureBox)[#]BBFC':

37, None], 'Saint Sinner (PictureBox)': [1507, None], 'Bad Boys (1995) (Sony Pictures)_BBFC': [1427, None],

bsters (PictureBox)': [1922, None], 'G-Force (Disney)': [1486, None], '_test:Twins Effect; The (PictureBox)':

36, None], 'Desperado (1995) (Sony Pictures)_BBFC': [1297, None], 'SabrinaThe TeenageWitch

ctureBox)_BBFC': [1272, None], 'Lady And The Tramp (Disney)': [1069, None], 'U-Turn (Sony

tures)_BBFC': [1687, None], 'Hook (Sony Pictures)_BBFC': [2105, None], 'Matilda(1996) (Sony

tures)_BBFC': [3395, None], 'Dumbo (Disney)': [1144, None], 'Snow White: The Fairest Of Them All
ctureBox)': [1177, None], 'Wall-E(Disney)': [1282, None], 'Gambit': [1078, None], 'Scary Movie 2

ramax)_BBFC': [1033, None], 'Charlie St. Cloud (PictureBox)_BBFC': [2774, None], 'The Story Of Us

arnerFilm)_BBFC': [1580, None], 'Ali G Indahouse (PictureBox)_BBFC': [1062, None], 'Lego: The Adventures

Clutch Powers (PictureBox)_BBFC': [1064, None], 'Baby Mama (PictureBox)': [1089, None], 'Borrowers; The

ctureBox)': [1177, None], 'Swan Princess; The (Sony Pictures)_BBFC': [1710, None], 'Chronicles Of Narnia;

: *Prince Caspian (Disney)': [1104, None], 'Kids Are All Right; The (PictureBox)': [1445, None], 'Adventures Rocky And Bullwinkle (PictureBox)[#]BBFC': [1894, None], 'Proposal; The (2009) (Disney)': [1441, None], sketeer; The (Miramax)_BBFC': [1117, None], 'Flashbacks Of A Fool (Disney)': [1209, None], 'Principles Of t; The (Film4oD)': [1081, None], 'Memoirs Of A Geisha (Disney)': [1499, None], 'Gone Baby Gone ramax)_BBFC': [1040, None], 'Wild Hogs (Disney)': [1068, None], 'Hot Fuzz (PictureBox)[#]BBFC': [1611, ne], 'Skeleton Key; The (05) (PictureBox)_BBFC': [1431, None], 'High School Musical 3: Senior Year (Disney)': 28, None], 'Johnny English (PictureBox)[#]BBFC': [1346, None], 'Spy Kids (Miramax)_BBFC': [1535, None]}*

reck it Ralph', 'Flashbacks Of A Fool (Disney)', 'Saint Sinner (PictureBox)', 'Cars 2 (Disney)', 'Few Good Men; Sony Pictures)_BBFC', 'Brother Bear (Disney)', 'Hobbit; The An Unexpected Journey', 'Lego: The Adventures Clutch Powers (PictureBox)_BBFC', 'Gambit', 'Brother Bear 2 (Disney)', 'Bad Boys (1995) (Sony

tures)_BBFC', 'Your Highness (PictureBox)', 'Don Quixote (PictureBox)', 'Dumbo (Disney)', ',', 'Baby Mama ctureBox)', 'Scary Movie 2 (Miramax)_BBFC', 'Gone Baby Gone (Miramax)_BBFC', 'Mobsters (PictureBox)', ventureland (Miramax)_BBFC', 'Kids Are All Right; The (PictureBox)', '13 Ghosts (2001) (Sony tures)_BBFC', 'Atlantis: The Lost Empire (Disney)', 'Up (2009) (Disney)', 'Accepted (PictureBox)[#]BBFC', ok (Sony Pictures)_BBFC', 'Lady And The Tramp (Disney)', 'Life Of Pi', 'Confessions Of A Shopaholic (Disney)', istocats; The (Disney)', 'Rise of the Guardians', 'Borrowers; The (PictureBox)', 'Ali G Indahouse ctureBox)_BBFC', 'Sabrina The Teenage Witch (PictureBox)_BBFC', 'Principles Of Lust; The (Film4oD)', tilda (1996) (Sony Pictures)_BBFC', 'Spy Kids (Miramax)_BBFC', 'Hot Fuzz (PictureBox)[#]BBFC', oposal; The (2009) (Disney)', 'Johnny English (PictureBox)[#]BBFC', 'Adventures Of Rocky And Bullwinkle ctureBox)[#]BBFC', 'Bourne Ultimatum; The (PictureBox)', 'Snatch (2000) (Sony Pictures)_BBFC', 'Quartet', gh School Musical 3: Senior Year (Disney)', 'Here Comes the Boom', 'Charlie St. Cloud (PictureBox)_BBFC', Fast 2 Furious (PictureBox)', 'Musketeer; The (Miramax)_BBFC', 'Despicable Me (PictureBox)', 'Seve' chopaths', 'Emperors New Groove; The (Disney)', 'Surrogates (Disney)', 'Eagle; The (PictureBox)', 'Memoirs A Geisha (Disney)', 'Silver Linings Playbook', 'Far And Away (PictureBox)_BBFC', '_test:Twins Effect; The ctureBox)', 'Jack Reacher', 'Hollow Man (Sony Pictures)_BBFC', 'Fast And The Furious; The (PictureBox)', ne Collector; The (Sony Pictures)_BBFC', 'Desperado (1995) (Sony Pictures)_BBFC', 'G-Force (Disney)', ch Perfect', 'Argo', 'U-Turn (Sony Pictures)_BBFC', '102 Dalmatians (Disney)', 'Snow White: The Fairest Of m All (PictureBox)', 'Stand By Me (Sony Pictures)_BBFC', 'Wild Hogs (Disney)', 'Skeleton Key; The (05) ctureBox)_BBFC', 'Impossible; The', 'Django

Unchained', 'Wall-E (Disney)', 'Flight (Before DVD!)', 'A Good y to Die Hard', 'Chronicles Of Narnia; The: Prince Caspian (Disney)', 'Swan Princess; The (Sony tures)_BBFC', 'The Story Of Us (WarnerFilm)_BBFC', 'I Give It A Year', 'Seed Of Chucky (PictureBox)'}

1 The frequent data in the dataset (Insight into frequent pattern)

ck Reacher 1816, Hobbit; The An Unexpected Journey 647, Pitch Perfect 89, Impossible; The 28, Silver

ings Playbook 7, 102 Dalmatians (Disney) 1, I Give It A Year 1, A Good Day to Die Hard 1, Seven

chopaths 1, Here Comes the Boom 1}

The explanation to this is amongst the list of transactions, the most frequent sequential order of movie watched is ollowed. *Jack Reacher* appears 1st 1816 times but it has a total frequency of 4355 in the entire movie dataset. *bbit; The An Unexpected Journey* appears 2nd after *Jack Reacher* 647 times but it has a total frequency of 4309 he entire movie dataset. *Pitch Perfect* appears in 3rd position after *Hobbit; The An Unexpected Journey* appears times but has a total frequency of 5399 in the entire movie dataset. *Impossible; The* appears 4th position after *ch Perfect* 28 times but has a total frequency of 3169 in the entire movie dataset. *Silver Linings Playbook* appears position after *Impossible; The* 7 times but has a total frequency of 54 in the entire movie dataset. *102 Dalmatians sney)* appears 6th position after *Silver Linings Playbook* 1 time but has a total frequency of 1790 in the entire vie dataset. *I Give It A Year* appears 7th position after *Silver Linings Playbook* 1 time but it has a total frequency 2603 in the entire movie dataset. *A Good Day to Die Hard* appear 8th position after *I Give It A Year* 1 time but a total frequency of 1511 in the entire movie dataset. *Seven Psychopaths* appears 9th position after *A Good Day Die Hard* 1 time but has a total frequency of 1349 in the entire movie dataset. *Here Comes the Boom* appears

position after *Seven Psychopaths* 1 time but has a total frequency of 1801 in the entire movie dataset.