Analyzing Data Streams Using a Dynamic Compact Stream Pattern Algorithm

Ayodeji Oyewale
School of Computing, Science& Engineering
University of Salford
Salford, Manchester, United Kingdom
E-mail: a.oyewale@edu.salford.ac.uk

Chris Hughes
School of Computer, Science & Engineering
University of Salford
Salford, Manchester, United Kingdom
E-mail: c.j.hughes@salford.ac.uk

Mohammed Saraee
School of Computing, Science & Engineering
University of Salford
Salford, Manchester, United Kingdom
Email: m.saraee@salford.ac.uk

Abstract— **In order to succeed in the global competition, organizations need to understand and monitor the rate of data influx. The acquisition of continuous data has been extremely outstretched as a concern in many fields. Recently, frequent patterns in data streams have been a challenging task in the field of data mining and knowledge discovery. Most of these datasets generated are in the form of a stream (stream data), thereby posing a challenge of being continuous. Therefore, the process of extracting knowledge structures from continuous rapid data records is termed as stream mining. This study conceptualizes the process of detecting outliers and responding to stream data. This is done by proposing a Compressed Stream Pattern algorithm, which dynamically generates a frequency descending prefix tree structure with only a single-pass over the data. We show that applying tree restructuring techniques can considerably minimize the mining time on various datasets.**

Keywords- Data Mining; Frequent Pattern (FP); Stream data; Compact Pattern Stream (CPS) & Interactive Mining, Path Adjustment Method (PAM), Branch Sort, Merge Sort Algorithm.

I. INTRODUCTION

Within the global business world, understanding data is crucial to success. A lot of research has been dedicated to the computation on data where most companies are able to collect enormous amounts of it with relative ease. Without doubt, many companies now have more data than they can handle, a vital portion of this data entails large unstructured data sets that amount up to 90 percent of an organization's data. The ability to mine and analyse data, in any form, from many sources, gives us deeper and richer insight into business patterns and trends, helping drive operational efficiencies and competitive advantage in manufacturing, marketing, security and IT at large [4]

In an ideal corporate world, competitiveness should be successfully and sustainably built on data; however, the reality is that obtaining good quality data for decision-making in this milieu is a big ordeal [1]. Detecting meaningful patterns in streaming applications is particularly challenging. The detector must process data and output a decision in real-time, rather than making many passes through batches of files. In most scenarios the number of streams is large and there is little opportunity for human, let alone expert intervention. As such, operating in an unsupervised, automated fashion (e.g., without manual parameter tweaking) is often a necessity.

Data streams are continuous, changing sequence of data that constantly arrive at a system and needs to be processed in near real-time. The dissemination of data stream phenomenon has necessitated the development of diverse range of stream mining algorithms. Several studies [2] describe the approaches currently being used to overcome the challenge of storing and processing fast, continuous and uninterrupted streams of data.

Subsequently, latter sections of this paper enumerates on the data models which is found in section two, the approach to exploit the CSP algorithm is introduced in section three. Section four explains the criteria through which the algorithm is evaluated. The procedures in analyzing a stream of data are explained in section five.

II. DATA MODELS

When weighed against data in traditional databases, data streams are unbounded and the number of transactions increases over time. As a result of these, different data models for effective processing have been suggested in different mining algorithms.

The adapted model is based on a sliding window. That is, despite the infinite arrival of the stream data, the frequent itemsets 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 array and patterns from the stream data(2).

Data streams differ from the conventional stored relation model in several ways:

i.) The data elements in the stream arrive online.

ii) The system has no control over the order in which data elements arrive to be processed, either within a data stream or across data streams.

iii) Data streams are potentially unbounded in size.

iv) the instance an element from a data stream has been processed it is discarded or archived — unless specifically stored in an external storage point it cannot be retrieved easily, which ideally is small relative to the size of the data streams. Frequently, information stream inquiries may perform joins between information streams and put away social information.

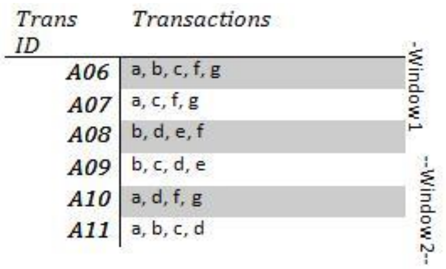
For the motivations behind this paper, we will accept that if put away relations are utilized, their substance stay static.

Therefore, we block any potential exchange preparing issues that may emerge from the nearness of updates to put away relations that happen simultaneously with information stream handling.

III. APPROACH TO CSP ALGORITHM

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 the CSP algorithm allows for direct development of frequency-descending item order list from data. This will save the algorithm from iterating over tree nodes several times during the creation and modification of tree. Sliding window is used in this work where data are captured in panes which are housed in windows. When new data is inserted, the window slides thereby removing some old pane and inserting new ones; depending on sliding window size. It constantly updates itself by extracting the expired transaction after each window slides. Ensuring that the tree does not contain unwanted data points .



Trans ID	Transactions	
A06	a, b, c, f, g	-Window1
A07	a, c, f, g	
A08	b, d, e, f	
A09	b, c, d, e	-Window2--
A10	a, d, f, g	
A11	a, b, c, d	

Figure 1 Transaction with Window

Each transaction is processed with the definition of window size and pane size as show the transaction table above. 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.

A. Formatting of Data

The algorithm does not 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

B. Creation of Window

In order to study the characteristics of dynamic flow, it is eminent to configure window size since the appropriate window size is a determinant in effectively carrying out an analysis on the datasets.

Creation of new window requires;

a) Size of window: maximum number of pane the window will contain
b) Size of pane, and
c) Sliding size

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

The code snippet above defines the new window that needs to be created after pan slide.

IV. ALGORITHM EVALUATION

Through theoretical and experimental analysis, frequent-item identifying algorithms are often evaluated based on three aspects:

- accuracy,
- runtime, and
- space usage

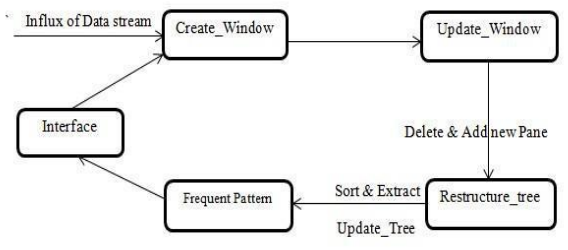


Figure 2: Data-Flow Architecture

A successful connection should be established which makes easy passage of the data. A given transaction is introduced into the current sliding window with its respective support value. The sliding window is used in the next update window module which takes the input from window creation module and deletes the old panes thereby adding new panes so as to mine the latest frequent patterns. Thereafter, the updated window is passed as input to the restructure module which in turn performs the extraction, sorting and reinsertion operation. The sorted, extracted data is also passed as input to the final module which mines all the transactions over dynamic data streams and finds the latest frequent patterns. The discovered patterns which are greater than threshold value are finally displayed as the set of latest frequent patterns over the dynamic data stream.

In order to restructure the CPS-tree, we use our proposed efficient tree restructuring mechanism, called Branch sorting method (BSM) [6], and the Path adjusting method proposed in [5]. As the CPS-tree is developed within the current window, a base up FP tree mining procedure is used to create exact set of most recent frequent patterns. The mining task is very proficient due to the recurrence plummeting tree structure.

V. PROCEDURES IN ANALYZING STREAM DATA

In order to develop and analyse an incessant influx of data, it is required to take into account vital information with respect to stream data.

1) *Data preprocessing:*

Information preprocessing in this examination work includes the utilization of a middleware which fills in as a working shrouded interpretation layer which empowers a correspondence and collaboration, and information administration between the working framework and the application running on it.

2) *The middleware:*

The middleware class is made to assemble skeleton of communication between the system and the information it is intended to be processed. The constructor of this class requires record name of the document that ought to be prepared, the thing column(s) that the calculation should process and the value-based section the thing ought to be coordinated against.

```
_ _def init (self, file, transaction_field, item_field):
```

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

3) *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.

$$(Di) = \underline{\{So, S1 \ldots Si\} i \leq n - 1} \quad \{Si, Si+1 \ldots Sn + i - 1\} \geq n - 1 \qquad (1)$$

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

VI. RESULT ANALYSIS

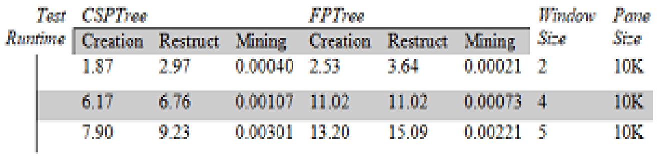
An empirical evaluation of the performance of CSP implementation is made. A comparative analysis of CSP tree with FP tree algorithm is done using. All programs are done using Oython 3.0 and executed in WInsows 7 on a 2.66 GHz CPU with 1GB memory usage. And this is done using the BSM, Sales and Telecom datasets.

TABLE I: DATASET CHARACTERISTICS

Datasets	No of Transactions	Size(M)
BSM	515,597	2GB
Sales	88, 475	9.7M
Telecoms	9,683,900	10.6GB

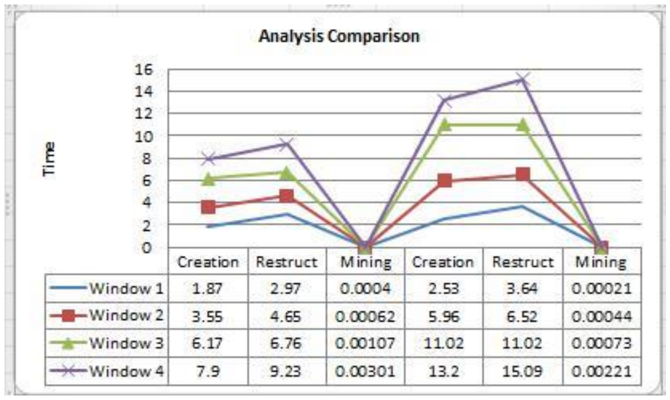
The aforementioned Table I above shows the characteristics of the datasets used for the analysis. The BSM dataset contains several entries from an electronics retailer. The Sales dataset consists of retail Watson Analytics Sample Data of Sales Products and the telecoms datasets consists of the call detail record of customers. In this experiment, the average runtime of all active windows are computed for the two algorithms on each data. At every window, mining is perfromed on the recent window. The pane size is dependent on the size of the dataset introduced.

TABLE II: RUNTIME ANALYSIS



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
	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

A runtime analysis for the three dataset is shown in the Table II 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.
It 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.

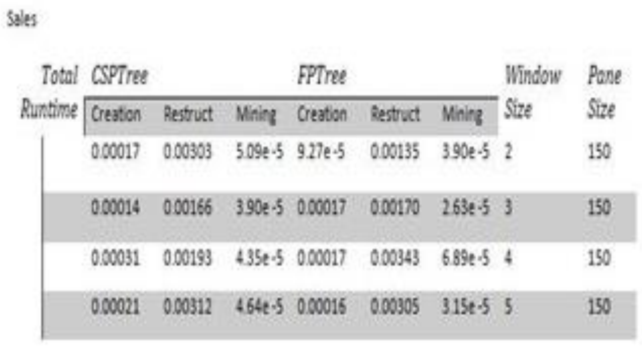


	Creation	Restruct	Mining	Creation	Restruct	Mining
Window 1	1.87	2.97	0.0004	2.53	3.64	0.00021
Window 2	3.55	4.65	0.00062	5.96	6.52	0.00044
Window 3	6.17	6.76	0.00107	11.02	11.02	0.00073
Window 4	7.9	9.23	0.00301	13.2	15.09	0.00221

Figure 3: CSP AND FP Comparison on BSM Data

A comparative analysis for BMS dataset between CSP Tree and FP tree is shown in the Figure 3 above. The graph indicates precisely that at each level of data creation, restructuring and mining, CSP tends to outpace FP Tree.

TABLE III: RUNTIME ANALYSIS ON SALES DATA

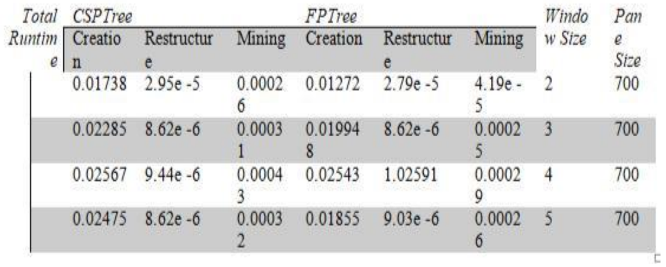


Sales

Total Runtime	CSPTree			FPTree			Window Size	Pane Size
	Creation	Restruct	Mining	Creation	Restruct	Mining		
	0.00017	0.00303	5.09e-5	9.27e-5	0.00135	3.90e-5	2	150
	0.00014	0.00166	3.90e-5	0.00017	0.00170	2.63e-5	3	150
	0.00031	0.00193	4.35e-5	0.00017	0.00343	6.89e-5	4	150
	0.00021	0.00312	4.64e-5	0.00016	0.00305	3.15e-5	5	150

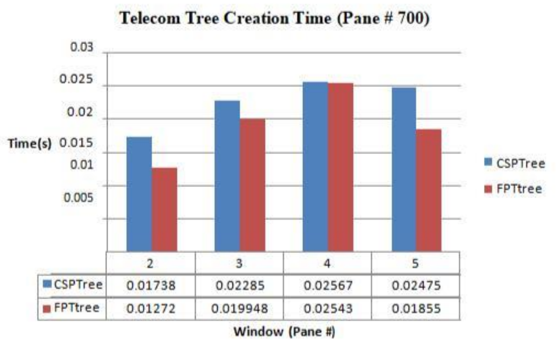
The runtime analysis on Sales dataset shown in a Table III above comprises of the time it takes for a tree to be created before restructuring can take place. Thereafter, the tree restructured based on the new set of incoming dataset. It also shows how long it takes to mine a specific dataset present in the pane of a predefined window.

TABLE IV: RUNTIME ANALYSIS ON TELECOMS DATASET



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

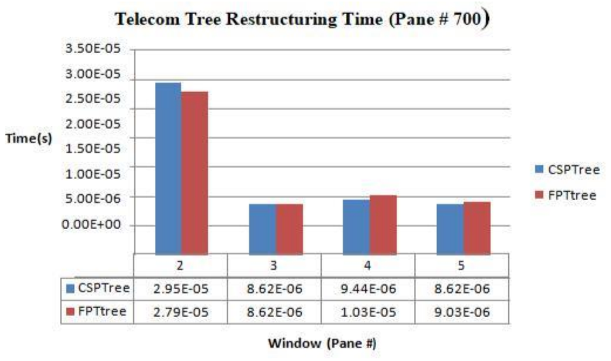
The runtime analysis in Table IV is the Test runtime for the Telecoms dataset which comprises of the tree creation, tree restructuring and how long it takes to mine a given instance of the telecoms data present in the pane of a window. As can be seen, the mining rate of CSP is quite faster when compared to that of Fp tree.



	2	3	4	5
CSPTree	0.01738	0.02285	0.02567	0.02475
FPTtree	0.01272	0.019948	0.02543	0.01855

Figure 4: Tree Creation

The time it takes for the algorithm to create a tree for the telecoms between CSP Tree and FP tree is drawn up in the Figure 4 above. 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.



	2	3	4	5
CSPTree	2.95E-05	8.62E-06	9.44E-06	8.62E-06
FPTtree	2.79E-05	8.62E-06	1.03E-05	9.03E-06

Figure 5: Tree Restructuring

Also shown above is 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. The following datasets are being analysed with respect to the time of tree creation, the time rate for restructuring and how long it takes to mine the data in conjunction with the varying time window. This juxtapose has been made to make a comparative analysis between CSP and FP algorithm. The time it takes to create a tree using CSP algorithm is shorter when compared to FP algorithm. It takes CSP 1.8sec to build a tree while it takes FP algorithm over 3 sec. The figure 5 below shows a graphical advantage of CSP over FP.

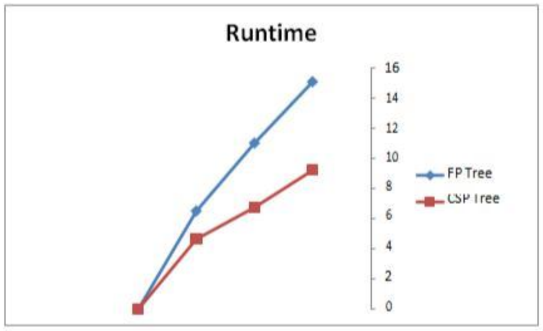


Figure 6: Graphical Advantage of CSP over FP

The above Figure 6 depicts the Tree restructuring phase of the all dataset. While restructuring the tree, we use the Path Adjustment method (PAM) which depends on the Degree of Displacement of two items and it swaps two nodes with Bubble sort method. And also, the branch sort algorithm makes use of the merge sort approach. Due to the high rate of displacement while using PAM, it is unsuitable for the algorithm hence BSM performs better during tree restructuring approach with merge algorithm. The merge sort method unlike quicksort makes use of Divide and Conquer algorithm. The most recent input array is intermittently divided in two halves; it sorts the two halves and then merges the two sorted halves.

VII. CONCLUSION

We have proposed CSP-tree that dynamically achieves frequency-descending prefix tree structure with a single-pass by applying tree restructuring technique and considerably reduces the mining time. We also adopted Branch sorting method using merge sort which is a new tree restructuring technique and presented guideline in choosing the values for tree restructuring parameters. It shows that despite additional insignificant tree restructuring cost, CSP-tree achieves a remarkable performance gain on overall runtime. The easy-to-maintain feature and property of constantly summarizing full data stream information in a highly compact fashion facilitate its efficient applicability in interactive, incremental and stream data.

REFERENCES

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

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

[3] B. Borja, C. Bernardino, C. Alex, R. Gavald`a, M. David Manzano-Macho, The Architecture of a Churn Prediction System Based on Stream Mining. 2013.

[4] Intel IT, IT Best Practices Business IntelligenceRetrieved from http://www.intel.com/it/.Feb 2012.

[5] J.-L. Koh and S.-F. Shieh. An efficient approach for maintaining association rules based on adjusting FP-tree structures. In Lee Y-J, Li J, Whang K-Y, Lee D (eds) Proc. of DASFAA 2004. Springer-Verlag, Berlin Heidelberg New York, 417–424. , 2004

[6] S. K. Tanbeer, C. F. Ahmed, B. S. Jeong,, and Y.-K. Lee. CP-tree: A tree structure for single-pass frequent pattern mining. In Proc. of PAKDD, Lecture Notes Artif Int, 1022-1027. 2008

[7] T.A. Rashid, 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. Oct 2016.