1st International Conference on Advances in Science, Engineering and Robotics Technology 2019 (ICASERT 2019)

Improved Piecewise Constant Approximation Method for Compressing Data Streams

Atik Mahbub
Department of Computer Science & Engineering
East West University
Dhaka, Bangladesh
atikmahbub100@gmail.com

Farhana Haque
Department of Computer Science & Engineering
East West University
Dhaka, Bangladesh
farhanatisha23@gmail.com

Habibul Bashar
Department of Computer Science & Engineering
East West University
Dhaka, Bangladesh
basharewu@gmail.com

Mohammad Rezwanul Huq
Department of Computer Science & Engineering
East West University
Dhaka, Bangladesh
mrhuq@ewubd.edu

***Abstract*— Stream data, obtained from the various sensors and devices with high speed, used daily, are indispensable parts in our day-to-day life. There is an immense necessity to store these data comprising of large volume. Data compression is one of the effective ways of storing data efficiently. In this paper, different data compression techniques such as PCA, APCA have been improved and thus we propose a new hybrid model. In the proposed model, the window size is fixed but data can be compressed efficiently than PCA. Besides, the delay time is minimized compared to the other existing techniques since the model is developed by considering both the delay time and several data points to be compressed. Our experiment shows that the proposed method performs twice as better than PCA in terms of compression and minimizes RMSE by 10%-25% compared to APCA method.**

Keywords—Data Compression, Stream data, Window

I. INTRODUCTION

Stream data has entrusted to be one of the most valuable concepts in Modern information technology. Experts are predicting that the content of stream data will become a huge amount of data that has been collected from social media or any other online resources [5], [12]. As Stream data are ejected from cell phones, vehicles, appliances, buildings, meters, machinery, medical equipment and many other machines continuously [3], it becomes huge data and turnout cumbersome to store this large volume of data. Sometimes new data removes old data due to lack of space in storage. Compression of data activates quick processing of data by accessing queries immediately over compressed data [1], [2],
[10] using few I/O as compressed data usually stored using lessen number of disk pages [4], [6]. Our aim is to compress the continuously coming stream data to make the memory space free from over-burdened data [3] and Data should be compressed in such way that compressed value should be close to actual value [9]. The data values are divided into certain segments, which are known as windows and a threshold to support error rate is considered. An improved of Piecewise Constant Approximation (PCA) technique is being followed, where window size of data is fixed-length but it gives optimal result in consideration of delay time and total compressed data points on running process rather than PCA [7], [8].

II. RELATED wORK

Piecewise Constant Approximation (PCA) & Adaptive Piecewise Constant Approximation (APCA) are two exoteric techniques to compress data optimally in model-based compression [9], [10]. Model Based Compression depends on two main properties, which are continuity of the sampling methods or processes and interrelations of different sampling processes [11].

PCA distributes the entire dataset into piecewise fixed length segments, which are known, as windows [7]. For each window, if the difference between the maximum and minimum value of respective windows is less than twice of the maximum error rate threshold than the whole window will represent one single average value. PCA algorithm gives more data points as window size is fixed. However, the delayed time of PCA is low.

APCA is not as straightforward as PCA [8]. Here, the window sizes vary according to the condition. It scans data in a selective order and inserts data into a window until the difference between maximum and minimum value is less than twice of maximum error rate threshold. If we consider the best case where all off the data values maintaining threshold value, the device will not get any data for a long time. Though APCA gives fewer data points but the time, which can be delayed of through APCA is higher.

III. PROPOSED METHOD

Like PCA, this method also divides the entire dataset into piecewise fixed length segments known as windows, W and it is users define value. A maximum error support threshold, € is also in under consideration. The maximum value, 1 ϵ W and the minimum value, V_{min} ϵ W of each window is calculated. If the maximum and minimum value of the window is less than twice of the maximum error rate threshold then the data values will be compressed.

The equation looks like the following.

$$V_{max} - V_{min} < 2*€ \quad (1)$$

The above equation can be named as "**Compression Condition**". If the above condition is true, all the data points, $V_i \in$ W is represented by a constant. That constant can be named as "**Representative Value**".

$$\text{Representative Value, V} = \frac{V_{max} + V_{min}}{2} \quad (2)$$

A. *Phases of the Model*

The whole method is partitioned into two phases. One is Merge phase and another is a Split phase. Initially, if the data points of the window follow the compression condition, then it will go to the Merge phase. Otherwise, it will go to the split phase.

a) Merge Phase: When the values of a window follow the compression condition then the window is saved into the buffer and the next window is formed with next consecutive data points. Now, V_{max} & V_{min} are calculated from this current window and buffer elements. If the difference between V_{max} & V_{min} is also followed the compression condition, then these two windows will be merged together and temporarily store the window in the buffer and form the next window in the same manner and follow the same process. After each window is compressed, the value of the buffer will be updated by adding the current window's data points. If the compression condition is not violated for at most N consecutive windows, then the data points of these N consecutive windows will be merged and a representative value will be calculated that will be sent to the server. Here N indicates at most how many windows will be merged. It is an arbitrary number and user input. If the values of N consecutive windows do not meet the compression condition, then after each window is compressed, the window stored in the buffer will calculate its representative values and will send into the server. Then it clears the buffer.

b) Split Phase: If the compression condition breaks in PCA, then the full window is sent to the server. But in this method, instead of sending the whole window, it splits the data points of the current window and tries to compress them. Thus, data points are reduced by more than PCA. It also scans data points of a window into a selective order like APCA.
Initially, V_{max} & V_{min} of the current window and their differences are calculated. Temporarily the current window is saved in the buffer. After that, the difference violates compression condition or not, is checked. There may be two types of cases at this stage. If V_{max} & V_{min} the current window does not violate the condition, then it will go case 1 otherwise the current window will go the split phase in case 2.

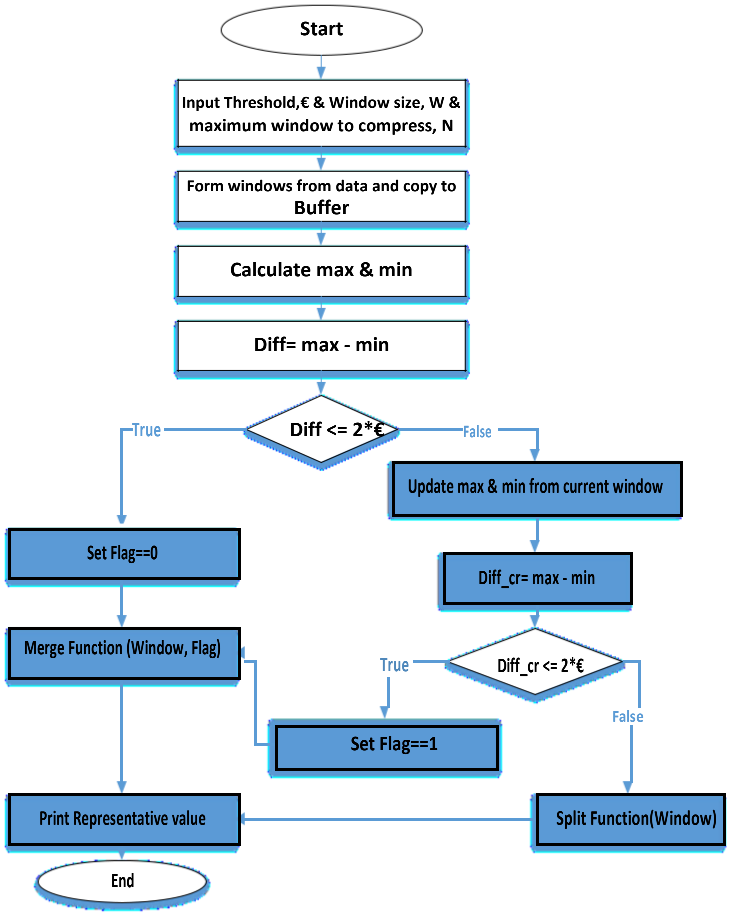
In the first case, since data points of the current window follows the condition, it forms the next window and checks the next window is eligible for merging or not in the previously described way. If the next window will not eligible for merging data points, then a representative value of the window that stored into the buffer will be calculated and sent to the server. After that, it clears the buffer.

In the second case, the first two data points of the buffer element are compared and it is seen that if those data points break the compression condition or not. If the condition breaks, the first data is sent is to the server and the second and third data are compared and again check the condition and thus, the process continues, if there are values in the above window. And if the first two data points of the buffer fulfill the compression condition, then those data points will be compared with the third data. In this way, all the values of that window will be checked sequentially until the buffer becomes empty and compare it to see if the compression condition is met or not. If the condition satisfies then data will be compressed. Finally, the buffer is cleared.

B. *Flow Chart*

As the entire process is partitioned into phases, it has three flow charts.

a) Main Procedure

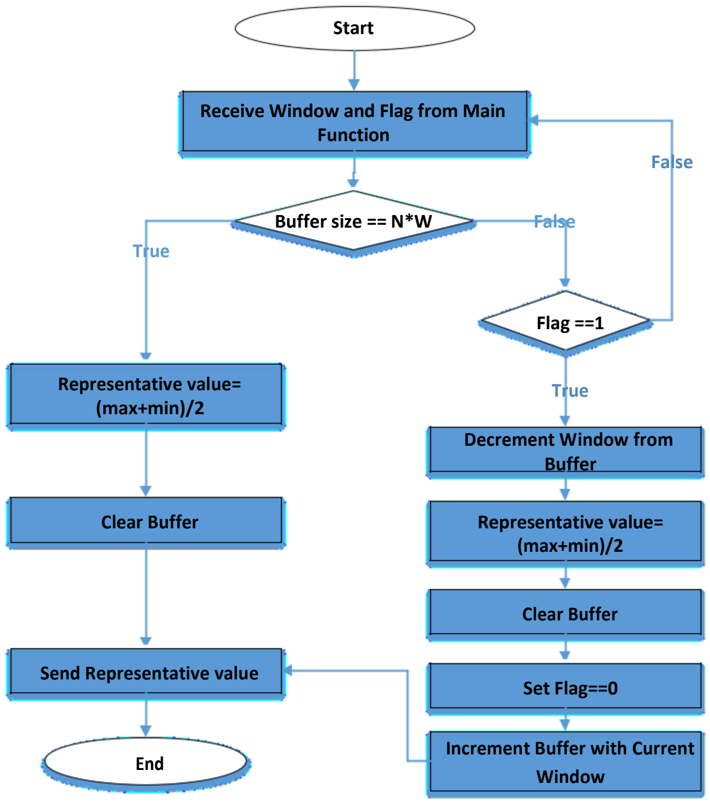


Start
Input Threshold,€ & Window size, W & maximum window to compress, N
Form windows from data and copy to Buffer
Calculate max & min
Diff= max - min
Diff <= 2*€
True
False
Update max & min from current window
Set Flag==0
Diff_cr= max - min
Merge Function (Window, Flag)
True
Diff_cr <= 2*€
False
Set Flag==1
Print Representative value
Split Function(Window)
End

Fig. 1. Flowchart of the Main Procedure

Fig. 1. explains the complete workflow of our proposed method. Initially, the program takes input window size as W; at most how many windows can be merged as N & threshold as **€** from users. Then build windows from the dataset and copy the window to the buffer. Now, the maximum and minimum value of the window and their differences are calculated and if the difference is less than twice of threshold and it initiates flag=0 and then calls the merge function with window and flag as an argument. Otherwise maximum and minimum is updated from the buffer the last window and their difference is calculated. If the updated difference is less the twice of threshold then it set flag=1 and call the merge function but if the difference is not less than 2*€ call the split function with a window as an argument. The representative value that calculated finally will send to the server or user.

b) Merge Phase



Start
Receive Window and Flag from Main Function
Buffer size == N*W
True
False
Flag ==1
False
True
Representative value= (max+min)/2
Clear Buffer
Send Representative value
End
Decrement Window from Buffer
Representative value= (max+min)/2
Clear Buffer
Set Flag==0
Increment Buffer with Current Window

Fig. 2. Flowchart of the Merge Phase

Fig. 2. shows the operations during the merge phase. In the Merge phase, receiving window and flag value it is checked that if buffer size and N*W are equal then the representative value will be calculated directly because buffer size is full. Then it will clear the buffer and sent the representative value to the main function as a return value. If buffer size and N*W are not equal, then it will check flag value. If flag=1 that means current window violates the compression condition, so it decrements the current window from the buffer, calculates the representative value and clears the buffer. After that flag will set as 0 to indicate that buffer can receive window now and increment buffer with current window value and finally representative value will send to the main function as a return value.

c) Split Phase

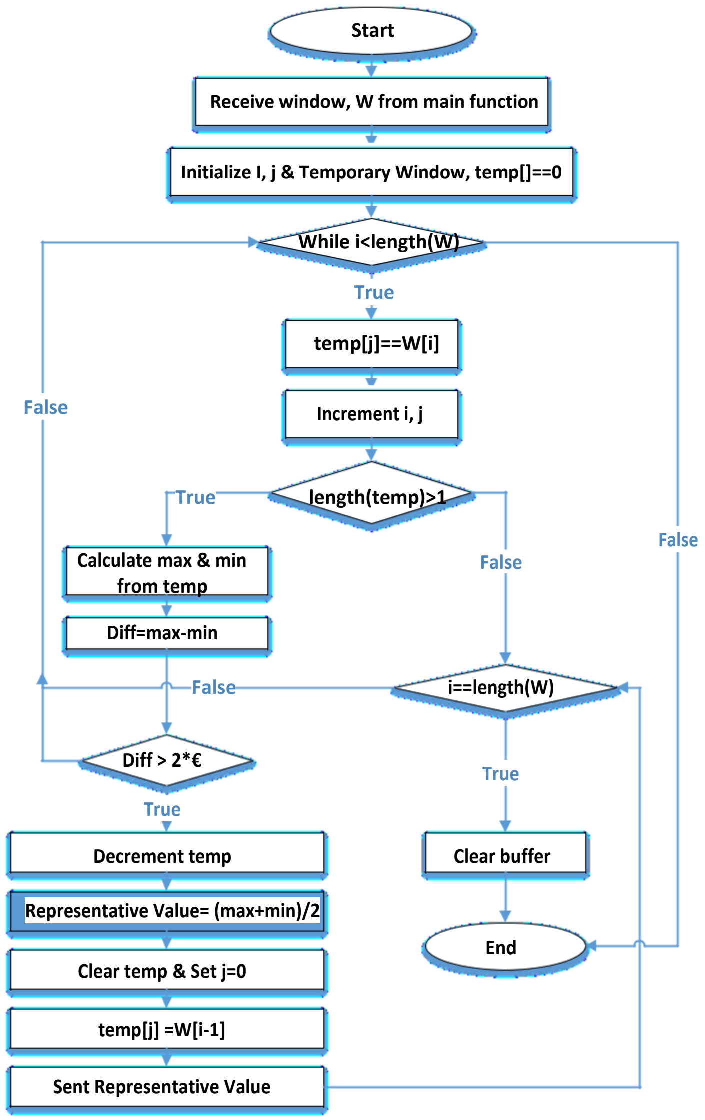


Fig. 3. Flowchart of the Split Phase

Fig. 3. shows the operations during the split phase. In the Split phase, window will receive as an argument from the main function. Then a temporary array is initiated and data of window will be copied to this temporary array. When there is more than one data in the temporary window then it will calculate the difference between the maximum & minimum value of the array. If the difference is less than 2*€ then it will go for next data otherwise pop the last value and calculate the representative value. After that clear the temporary array and current data point will be assigned into its zero indexes. This process will continue for every data point in the window. After scanning all data point's window will be cleared from the buffer.

IV. Implementation

In this method, fewer data points are sent to the server compare to PCA. In APCA, data values are sequentially searched. In the best case all the data values are followed compression condition then the algorithm will execute continuously. If the dataset is too large and a large number of data values are followed compression condition, the server will get any data for a long time so that delay time of APCA is larger. It is necessary to find results in an optimal way. This method is proposed considering both delay time and the number of data points. In PCA, data points are more but less delay time but in APCA data points are low but delay time is high.

A. Datasets

For evaluating Model-based Compression, the data values those have low spike those data will give the optimal result. Basically, we have taken three types of the environmental dataset from Kaggle. A variant number of stream data signals are contained by each and every data set. We have taken water's pH value as our first dataset. Around 10500 a number of data points in total. Their range is in between 0 to 7. We have chosen the second dataset is the one-year temperature of a country. There are roughly 11 thousand data points in total. The third dataset contains 5 years of high temporal resolution (hourly measurements) data of air pressure with more than 21000 number of data points.

B. Running Example

Data source generates data continuously. Here, window size, W is 5, error support threshold, € is 0.5 and N is 3 that indicates at most 3 windows can be merged for compression. Initially, first window forms with consecutive 5 data points from a data source which are 7.3, 7.7, 7.5, 7.2 and 7.1 as the window size are 5 and the window is saved to the buffer. The maximum value, V_{max} & the minimum value, V_{min} of the first window, is 7.7 & 7.1. While the difference between V_{max} & V_{min} is 0.6 which is less than 2*€ so this window can be compressed. Now, it checks the next window is eligible for merging or not.

Again, the second window formed with next consecutive 5 data points for example 7.4, 7.5, 7.6, 7.1, and 7. Now, V_{max} & V_{min} are calculated from this current window and buffer elements. Here, V_{max} is 7.7 & V_{min} is 7. As the difference is 0.7 so it can also be compressed. Temporarily this window also stores in the buffer and form the next window.

In a similar way, the third window is formed with next consecutive 5 data points for instance 7.4, 7.5, 7.6, 7.1, and 6.5 then again, V_{max} & V_{min} is calculated from this current window and buffer elements. Now, V_{max} is 7.7 & V_{min} is 6.5 and their difference is 1.1 so that it violates the compression condition. But it first two windows didn't violate the condition. Hence, at first, we merge the window of the buffer; calculate the representative value from windows that stored in the buffer and send a single representative value to the server instead of sending 10 data points. After that, it clears the buffer.

Now, V_{max} & V_{min} of the current window and their differences are calculated. After that, the difference violates compression condition or not, is checked. There may be two types of cases at this stage. If V_{max} & V_{min} of the current window does not violate compression condition, then it will go case 1 otherwise the current window will go the split phase in case 2.

Case 1: As V_{max} & V_{min} of current window do not violate compression condition, temporarily this window saves in the buffer, form the next window and checks the next window is eligible for merging or not in the previously described way. In this example, the data points of the current window break the compression condition.

Case 2: Since the difference of V_{max} & V_{min} violates the condition, this window is not compatible to compress. In split phase, temporarily this window saves in the buffer and it scans each data points and checks with the compression condition. Initially, the first two data points which are 7.4 & 7.5 is got and is checked whether the difference between these two data points follows the condition or not. The difference is less than twice of € and It maintains the condition. Now, it takes the next data point, 7.6 and checks if the difference between these three data points follows the condition or not. The similar process continues until the data points breaks the compression condition. Here, for the first 4 data points of the window, the compression condition is not violated.

The last data point, 6.5 violates the conditions so it makes a window from the first 4 values and sends a representative value to the server. The last data point also sends to the server. Then it clears the buffer and takes the next window. In this case, only two data points are sent to the server instead of sending 5 data points for this window. If more data points come, then the rest of windows will follow the same process of merging and splitting windows. Thus, compression occurs in the proposed method.

C. Prototype Development

We have made a desktop application with a user interface where the user can select any data source and select a window size, maximum window to compress as Max Window, error threshold value according to their wish. The application calculated total compression data, their compression ratio and RMSE of compression using this proposed technique.

V. Experimental Results

The proposed compression method is evaluated based on various performance factors like compressed data points, Compression ratio and approximation error.

A. Evaluation Measure

a) Compressed Data Points: Compressed data points indicate total how many values are compressed for all the data. Lessen compressed data points indicate the optimal result.

b) Compression ratio: The fraction of data storage values among compressed data and similar original data can be prescribed as compression ratio. The higher compression ratio indicates the higher ability to data compression.

$$\textbf{Compression ratio} = \frac{\textbf{size\_of\_actual\_data}}{\textbf{size\_of\_compressed\_data}} \qquad (3)$$

c) Approximation Error: At the point when compression methods demonstrate identical performance, approximation error can turn into a subordinate measurement to consider. To consider this viewpoint, the approximation errors of data compression are illustrated using the root mean square error (RMSE). RMSE generally used to assess the quality of data approximation.

$$\textbf{RSME} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(V_i - \widehat{V_i})}^2 \qquad (4)$$

Here, v_i means a raw data value in a given sensor data signal, and $\widehat{v_i}$ means the relating approximated value in the compressed signal. Better approximation occurs when the value of RMSE is low. As every stream data signal contains specific range where data values are limited, we standardized the RMSE esteems (from 0 to a given error tolerance €) at each and every experiment set, to have a typical view to exploring the experimental results.

B. Experimental Results

The evaluation is measured with various usages of compression techniques and error support threshold is computed. The absolute error changes over a generally unique range since every data set pass through different domain values. In this manner, it becomes difficult to compare all compression techniques in a similar way. To give the general picture, the relative error is used in the experiments. Besides, in the Merge Phase of the technique, the maximum number of consecutive windows that can be combined together depends on the machine. This varies for a different machine. For our three chosen dataset, we consider Error support threshold is 0.5, the maximum window to compress is 4 and the window size is 5. The results are presented in Table I and Figure 4, 5, 6.

As Table I shows, for dataset 1, dataset 2 and dataset 3 PCA gives the maximum compressed data points compared to the other two techniques. The proposed method gives much fewer data points compared to PCA.

Compare to the other two techniques APCA gives the highest value for each of the datasets. So APCA gives the most optimal result as we know the higher compression ratio indicates a higher ability to compress data.

Fig. 4. shows the experimental results in terms of total data points after the compression. APCA performs better than our proposed method which is not surprise, since it is optimal in terms of total data points after compression. However, our proposed method performs 100% better than PCA which is a major improvement. Fig. 5. shows the compression ratio obtained from these results.

Fig. 6. shows the experimental results in terms of RMSE of the compression algorithms. The RMSEs of data compression is computed to reflect the quality of compression. PCA gives the lowest RMSE value. The RMSE value of PCA and proposed methods is very close. As the lower RMSE value is the better approximation so PCA and proposed method the provides better result than APCA.

The proposed technique gives better result when considering all the performance parameters. It gives much less compressed data points than PCA and it gives higher compression ratio compared to PCA. Besides, RMSE value of this technique is smaller APCA. However, in a few cases, delay time and RMSE of compression is higher than PCA.

TABLE I. EXPERIMENTAL RESULTS

PCA			
	Dataset 1	Dataset 2	Dataset 3
Compressed Data Points	4765	4453	12519
Compression Ratio	2.27	2.41	1.71
RMSE of compression	0.21	0.20	0.30
APCA			
	Dataset 1	Dataset 2	Dataset 3
Compressed Data Points	1606	1387	4911
Compression Ratio	6.67	7.74	4.35
RMSE of compression	0.28	0.31	0.46
Proposed Method			
	Dataset 1	Dataset 2	Dataset 3
Compressed Data Points	2332	2244	6712
Compression Ratio	4.55	4.78	3.18
RMSE of compression	0.21	0.29	0.41

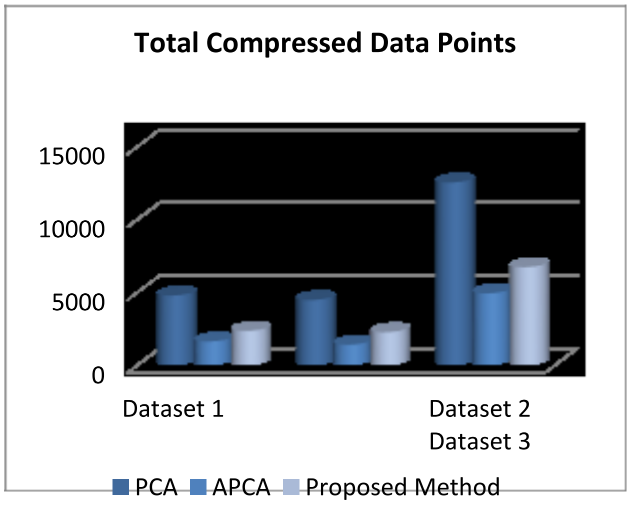


Fig. 4. Comparison of Total Compressed Data Points

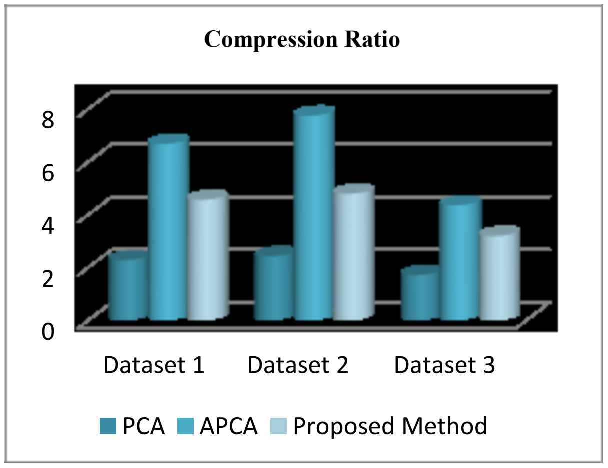


Fig. 5. Comparison of Compression Ratio

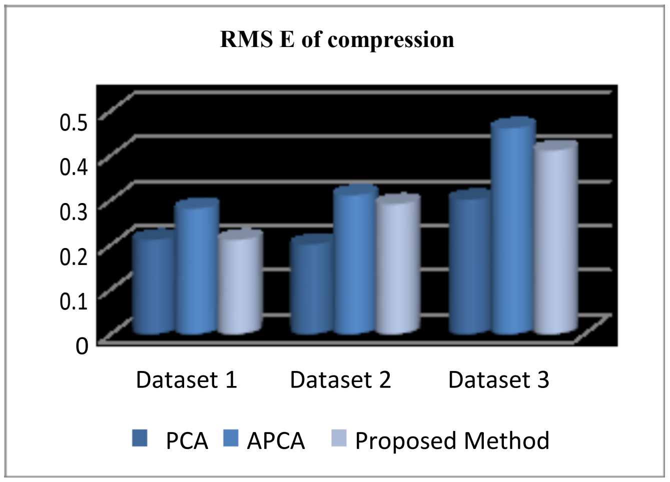


Fig. 6. Comparison of RMSE of Compression

VI. Conclusion and Future Work

This proposed technique is an improvement over PCA. Like PCA, widow size is constant here, but data point is also minimized notably. Considering two criteria such as compression ratio and RMSE value of compression this proposed technique gives most optimal result. Besides the delay time of this method is quite low so that machine or server does not wait long to get the values. Our future concentration is that the machine can automatically take decision about the window size which provides to get optimal result and that window size will be taken accordingly while algorithm will be run. Besides, we will try to apply this algorithm in diverse dataset and minimum more data points as much as can.

References

[1] X. Lian and L. Chen, "Efficient Processing of Probabilistic Reverse Nearest Neighbor Queries over Uncertain Data," in *Proc. VLDB Endowment, vol. 18, no. 3*, 2009, pp. pp. 787-808.

[2] H. Ding et al., "Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures," in *Proc. VLDB Endowment, vol. 1, no. 2*, 2008, pp. 1542-1552.

[3] Y. Kotidis, and N. Roussopoulo A. Deligiannakis, "Compressing Historical Information in Sensor Networks," *ACM SIGMOD Int'l Conf. Management of Data*, pp. 527-538, 2004.

[4] K. Sayood, *Introduction to Data Compression. Elsevier.*: 2000.

[5] D. Chu et al., "Approximate Data Collection in Sensor Networks Using Probabilistic Models," in *Proc. Int'l Conf. Data Eng. (ICDE)* , 2006, p. 48.

[6] T. Westmann et al, "The Implementation and Performance of Compressed Databases," in *SIGMOD Record*, vol. vol. 29, no. 3, pp-67, 2000, pp. 55-67.

[7] An Evaluation of Model-Based Approaches to Sensor Data Compressio, "Nguyen Quoc Viet Hung, Hoyoung Jeung, and Karl Aberer," *Ieee Transaction on Knowlegde and Data Enginering*, vol. 25, november 2013.

[8] Y. Cai and R. Ng, "Indexing Spatio-Temporal Trajectories with Chebyshev Polynomials," in *Proc. ACM SIGMOD Int'l Conf. Management of Data*, 2004, pp. 599-610.

[9] I. Lazaridis and S. Mehrotra, "Capturing Sensor-Generated Time Series with Quality Guarantees," in *Proc. Int'l Conf. Data Eng. (ICDE)*, 2003, pp. 429-440.

[10] E. Keogh et al, "Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases ," in *Proc. ACM SIGMOD Int'l Conf. Management of Data*, 2001, pp. pp. 151-162.

[11] N. Meratnia, P. Havinga Y. Zhang, "Outlier Detection Techniques for Wireless Sensor Networks: A Survey," *J. IEEE Comm. Survey & Tutorials*, vol. 12, no. 2, pp. 159-160, pp. 159-160, second quarter 2010.

[12] Maria ndrawan-Santiago, David Taniar Prajwol Sngat, "Sensor data management in the cloud:Data storage,data ingestion,and data retrieval," 2017.