THESIS TITLE: Design and Evaluation of fast and Efficient Clustering algorithms for high data volume using hardware/software Methodologies

Thesis submitted for the degree of Doctor of Philosophy (Science)

By

Payel Banerjee

(Index no: 147/16/Phys./25)

Under the Supervision of

Dr. Tapas Kumar Ballabh, Jadavpur University
&
Dr. Amlan Chakrabarti, Calcutta University.

Dept. of Physics

Jadavpur University

CERTIFICATE FROM THE SUPERVISOR(S)

This is to certify that the thesis entitled "Design and Evaluation of fast and Efficient Clustering algorithms for high data volume using hardware/software Methodologies" Submitted by Sri / Smt. PAYEL BANERJEE who got his / her name registered on 17 : 08 : 2016 for the award of Ph. D. (Science) Degree of Jadavpur University, is absolutely based upon her own work under the supervision of Dr. T.K. Ballabh and Dr. Amlan chakrabarti and that neither this thesis nor any part of it has been submitted for either any degree / diploma or any other academic award anywhere before.

TKBallabh 6/02/2023

Department of Physics
Jadavpur University
Kolkata - 700 032

[illegible] 6/02/2023

(Signature of the Supervisor(s) date with official seal)

Director
A. K. Choudhury
School of IT
University of Calcutta

DECLARATION

I declare that the work in this dissertation titled "Design and Evaluation of fast and Efficient Clustering algorithms for high data volume using hardware/software Methodologies" has been carried out by me under the guidance of my supervisors. The information derived from any literature has been duly acknowledged and a list of references has been provided. No part of this dissertation was previously presented for another degree at this or any other institution.

Payel Banerjee
01/03/2023
....
Signature of the Candidate

Acknowledgments:

This research work would not have been possible without the support of many people. I take this opportunity to thank all the people who were always there to help me out whenever it was required. I want to express my deepest gratitude to my supervisors Dr. Tapas Kumar Ballabh, Associate Professor, Jadavpur University and Dr. Amlan Chakrabarti, Professor, Calcutta University for showing interest on the topic selected by me and having so much patience to listen to my doubts and clear them whenever I called him up. I want to thank you sir, for helping me at every stage where my work was paused due to any sort of doubts.

Finally, I would express my thanks to my family and friends for their tremendous support and guidance throughout my work.

Dedicated to my Parents

List of Presentations and publications

1. **Banerjee, P.,** Ballabh, T.K. and Chakrabarti, A. (2020). PLEADER: A Fast and Area Efficient Hardware Implementation of Leader Algorithm. In: Saini, H.S., Singh, R.K., Tariq Beg, M., Sahambi, J.S. (eds) Innovations in Electronics and Communication Engineering. Lecture Notes in Networks and Systems, vol 107. Springer, Singapore. https://doi.org/10.1007/978-981-15-3172-9_61

2. **Banerjee, P.,** Chakrabarti, A. and Ballabh, T.K. (2021). An Efficient Algorithm for Complete Linkage Clustering with a Merging Threshold. In: Sharma, N., Chakrabarti, A., Balas, V.E., Martinovic, J. (eds) Data Management, Analytics and Innovation. Advances in Intelligent Systems and Computing, vol 1175. Springer, Singapore. https://doi.org/10.1007/978-981-15-5619-7_10

3. **Banerjee, P.,** Chakrabarti, A. and Ballabh, T.K. Accelerated Single-Linkage algorithm using triangle inequality. Sādhanā 45,66 (2020). https://doi.org/10.1007/s12046-020-1297-4

4. **Banerjee, P.,** Chakrabarti, A. and Ballabh, T.K. Accelerated Single Linkage Algorithm using the farthest neighbour principle. Sādhanā 46, 45 (2021). https://doi.org/10.1007/s12046-020-01544-6

5. **Banerjee, P.,** Chakrabarti, A. and Ballabh, T.K. An Efficient and Speedy approach for Hierarchical Clustering Using Complete Linkage method [To be presented in 5th IEEE International Conference on Electrical, Computer and Communication Technologies (IEEE ICECCT 2023)]

6. **Banerjee, P.,** Chakrabarti, A. and Ballabh, T.K. A Complete Linkage Algorithm for Clustering Dynamic Datasets [UNDER REVIEW]

Contents:

Chapter-1: Introduction **1**

1.1 Clustering and its types ………………………………………………………. 2

1.2 Application of clustering algorithms with special emphasis on hierarchical clustering ………………………………………………………………... 7

Chapter-2: Literature Review and novelty of our research work **10**

Chapter-3: Research Methodology **14**

3.1 Introduction ……………………………………………………………………… 14

3.2. Hardware vs. software programming languages ………………………….. 14

3.3. Language used for hardware design entry ………………………………… 15

3.4. Hardware Device Used for the Design Implementation …………….... 15

3.4.1. FPGA vs. Microcontroller ……………………………………………………. 16

3.4.2. FPGA vs. ASIC ………………………………………………………………….. 18

3.4.3. Importance of FPGAs ……………………………………………………….. 18

3.4.4. Internal structure of FPGAs …………………………………………………… 19

3.4.5. FPGA DESIGN FLOW ………………………………………………………………. 19

3.4.6. Software Tools for Synthesizing the Design ……………………………… 21

3.4.7. Tool for comparing clustering algorithms ………………………………….. 22

3.4.8. Data set used for Verification ………………………………………………… 22

Chapter-4: An Efficient Algorithm for Complete Linkage Clustering with a Merging Threshold **24**

4.1. Determining the threshold for the preclustering step ………………….. 25

4.2 Time and Space Complexity of the algorithm ……………………………. 29

4.3. The advantages and disadvantages of the algorithm ………………… 29

4.4. Experimental Results ……………………………………………………………. 30

Chapter-5: An Efficient and Speedy approach for Hierarchical Clustering using Complete Linkage Method 33

5.1. The necessity of the Final Merging stage ………………………………… 33
5.2. Calculation of the Maximum Inter-Cluster distance ………………… 35
5.3. Time and Space Complexity of the algorithm …………………………. 39
5.3.1. The Time and Space Complexity of the Pre-Clustering stage …… 39
5.3.2. The Time and Space Complexity of the Final Merging stage ……. 40
5.3.3. The total Time and Space Complexity of the algorithm …………… 43
5.4. Advantages and disadvantages of our algorithm over other well-known methods ……………………………………………………………………………. 43
5.5. Experimental Results …………………………………………………………. 45
5.6. Conclusion ……………………………………………………………………… 46

Chapter-6: A complete Linkage algorithm for clustering dynamic datasets 47

6.1. The final merging step ……………………………………………………… 47
6.1. a. keeping a record of all the pre-clusters being merged into a single cluster ………………………………………………………………………………… 47
6.2. Time and space complexity of the algorithm …………………………….. 48
6.2.1. The time and space Complexity of the Pre-clustering stage ……… 48
6.2.2 The time and space Complexity of the final merging stage ………… 48
6.2.3. The total time and space complexity of the algorithm …………….. 49
6.3. Clustering dynamic datasets ……………………………………………………. 49
6.4. Complexity for Clustering newly added data in dynamic datasets … 53
6.5. Data in threshold to more than one cluster ………………………………. 53
6.6. Advantages and disadvantages of the algorithm …………………………. 54
6.7. Experimental data ……………………………………………………………….. 55
6.8. Conclusion ……………………………………………………………………………… 56

Chapter-7: PLEADER: A Fast and Area Efficient Hardware Implementation of Leader Algorithm 57

7.1. PLEADER-Parallel Leader Algorithm ……………………………………………… 57

7.2. Hardware implementation of the proposed algorithm 59
7.2.1 Distance Calculation Unit 61
7.2.2. Comparing Unit 62
7.2.3 Clustering Unit 62
7.2.4 Control unit 62
7.3. Time and Space Complexity of the algorithm 63
7.4. Challenges in hardware implementation 65
7.5. Experimental results 65
7.6. Conclusion 68

Chapter-8: Accelerated single linkage algorithm using the farthest neighbor principle 70

8.1. Pre-clustering algorithm 70
8.1.1 Steps of the Single Linkage pre-clustering (SLP) algorithm 76
8.1.2 Accelerated Single Linkage pre-clustering (ASLP) algorithm 79
8.1.3 Time and space complexity of the Accelerated pre-clustering algorithm (ASLP) 80
8.2. The second stage of the algorithm 81
8.2.1 Steps of the algorithm 82
8.2.2 Algorithm used to scan a column of the distance matrix to find the nearest neighbor 83
8.2.3 Time complexity of the algorithm 84
8.2.4 Space complexity of the algorithm 85
8.3. Total time and space complexity of the two-stage algorithm 85
8.4. Effect of threshold on the quality of the clusters 85
8.5. Effect of threshold on the convergence time of the algorithm 86
8.6. Effect of the ordering of the input data on the cluster quality 87
8.7. Advantages and disadvantages of the algorithm 87
8.8. Experimental results 90
8.9. Conclusions 93

Chapter-9: Accelerated single linkage algorithm using triangle inequality 94

9.1. Accelerated SL algorithm 94
9.1. 1. Pre-clustering algorithm 94
9.1.2. The second stage of the algorithm.. 96

9.2. Time and space complexity of the proposed method …………………… 99
9.2.1. Calculation of time and space complexity of accelerating Leader algorithm ………………………………………………………………………………. 99
9.2.2. Time complexity of the second stage ……………………………………… 99
9.2.3 Total time and space complexity of the algorithm …………………… 101
9.3. Advantages of the algorithm over other well-known SL algorithms… 102
9.4. Experimental results ………………………………………………………………. 103
9.5. Conclusion …………………………………………………………………………………. 107

Chapter-10: Conclusion of the Thesis **108**

References **109**

Chapter-1
Introduction

In today's world, the main challenge faced by data experts is dealing with "data avalanche" – the term used to express high data volume generating at a very high speed. Analyzing such data volume is a tedious task due to insufficient space and memory. This lack of provision of storing the data before analyzing it, demands real-time "on the fly" data analysis. Thus, time plays a very vital role and the algorithms to be used for analyzing the data need to be fast and scalable enough to cope with the high incoming speed of the input data.

3 Vs:

Big data is characterized by 3Vs i.e., volume, variety, and velocity [1].

Volume: The word 'Volume' refers to the size of the dataset. In today's world, a large amount of data is getting generated by various platforms like social media, various smart devices, sensors, etc. With increasing data size, the architecture that is required to store these data needs to be smart enough in handling such data explosion.

Variety: The word 'variety' refers to the various forms of data getting generated. The real-world data in a database may not be of the same formats with different features and characteristics. These variations of data in a database make it very hard to analyze especially if the data is unlabeled.

Velocity:

Here 'velocity' refers to the speed of the data. These days the data getting generated are so fast that the processing has to be almost real-time to avoid loss of data. Thus, the existing algorithms have to be remodified by reducing their convergence time to cope with this speed of incoming data.

1.1. What is Clustering?

Clustering [2][3] is a tool that acts as a savior to the data experts for analyzing large data volumes with special emphasis on unsupervised learning. It not only aids in gaining a clear understanding of completely unlabeled data but also speeds up processing since instead of working with a set of data that are entirely unlabeled, groups of data are now available that share the same property, facilitating analysis.

The traditional variants of the Clustering algorithms have very high time complexity leading to the unsuitability of the algorithms in the recent scenario of "big data" analysis. Thus, our main focus is the reduction of the convergence time of the algorithms without increasing the space complexity and without hampering the cluster quality as far as possible.

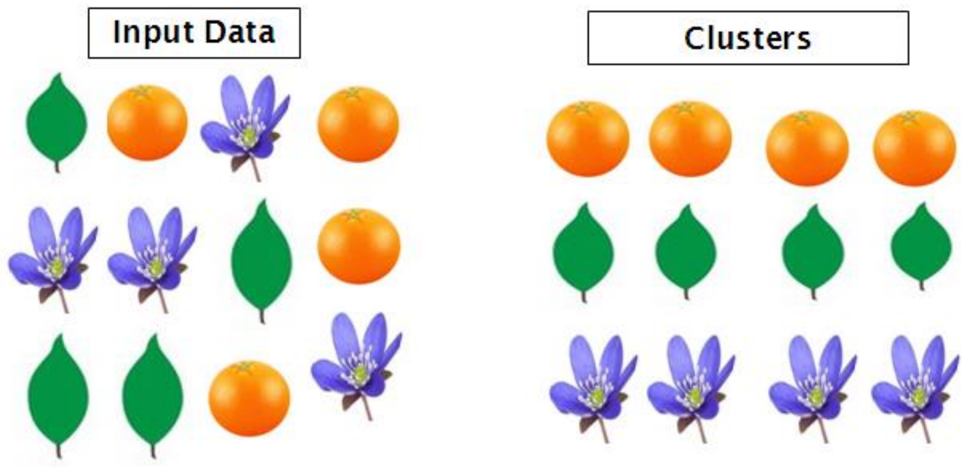
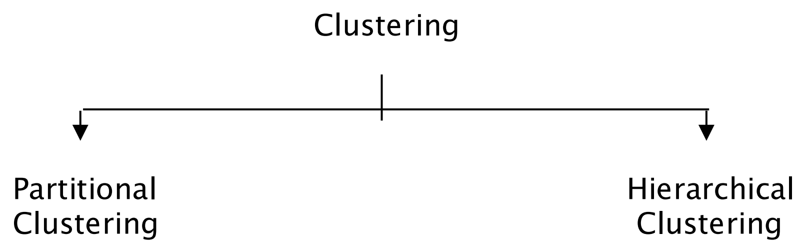


Figure 1: Segregating objects into clusters

The basic difference between Clustering and classification lies in the difference between *supervised* and *unsupervised learning*. Although both the methods deal with the categorization of data by segregating them into small groups, Classification does that based on the predefined data labels whereas clustering does not have any data labels to use. Thus, classification is used when data is labeled whereas clustering deals with purely unlabeled data.

Types of clustering algorithms:



Clustering algorithms are broadly classified into two major categories, Hierarchical and Partitional. Although the job of every clustering algorithm is to group the data based on some similar properties, the "definition of similarity" is not the same for all the algorithms and hence the output clusters may not be the same for all the algorithms even when applied on same data. Partitional Clustering algorithms [4] always require an apriori cluster number specification to provide correct clusters and an improper specification of the number of clusters may provide erroneous result. There comes Hierarchical clustering [5] to the rescue as it is capable of providing a clear insight about the clusters in a tree-like structure called Dendrogram without any advanced specification of the cluster number. Thus, Hierarchical clustering is always preferred over partitional methods where in-advance specification of the cluster number is not available. *Our research work mainly deals with Hierarchical clustering in an attempt to reduce its convergence time without increasing space and affecting cluster quality thereby making it suitable for large datasets.*

The hierarchical clustering algorithm is broadly classified into two types [6][7]:

a) **Agglomerative Clustering:** Agglomerative clustering is also commonly known as the "*bottom-up approach*" where every data is referred to as a cluster of size unity. Here at every iteration, two clusters are merged based on the distance criterion and the process repeats unless all the clusters are merged into a single cluster.

b) **Divisive Clustering:** Divisive clustering also referred to as the "*top-down approach*" is the process that starts with a single cluster containing all the data followed by splitting of the clusters into smaller ones unless all the data are separated as unit clusters.

Hierarchical Clustering is also divided into six special categories [8-12] based on the Linkage or similarity methods viz. Single Linkage, Complete Linkage, Average Linkage, Centroid Linkage, and Ward Linkage.

- **Single Linkage (SL):** Here the term "distance between two clusters" refers to the minimum distance between them. This method is also called as "nearest neighbor method".

  $\mathbf{dist\ (C_i, C_j) = min\ \{dist\ (x_i, x_j)\}\ \forall\ x_i \in C_i, \forall\ x_j \in C_j}$

- **Complete Linkage (CL):** Here "distance between two clusters" refers to the maximum distance between them. This method is also called as "farthest neighbor method".

  $\mathbf{dist\ (C_i, C_j) = max\ \{dist\ (x_i, x_j)\}\ \forall\ x_i \in C_i, \forall\ x_j \in C_j}$

- **Average Linkage (AL):** Here "distance between two clusters" refers to the average distance between all the members of one cluster to that of another cluster.

  $\mathbf{dist\ (C_i, C_j) = avg\ \{dist\ (x_i, x_j)\}\ \forall\ x_i \in C_i, \forall\ x_j \in C_j}$

- **Centroid Linkage:** Here "distance between two clusters" refers to the distance between the centroids of the two clusters.

  $\mathbf{dist\ (C_i, C_j) = \{dist\ (x_i', x_j')\}, x_i' = \Sigma\ x_i / n_i\ , x_j' = \Sigma\ x_j / n_j\ , \forall\ x_i \in C_i\ , \forall\ x_j \in C_j}$

where **x_i'** is the centroid of cluster **C_i** and **x_j'** is the centroid of cluster **C_j**.

In Centroid linkage, the centroid is mainly affected by the larger of the two clusters that are being merged. ***Median linkage*** alleviates the problem by giving equal importance to both the clusters and the position of the new centroid is always in the middle of the line joining the two previous cluster centroids.

- **Ward Linkage:** Here merging is done based on the minimum increase of the within-cluster variance also called SSE (Sum of Squared Error). This method merges two clusters where the minimum increase of the pooled within-cluster variance is possible. Here data is assigned to a cluster only if the sum of squares of the distances of all the points including this data from the mean of the cluster remains minimum.

  SSE (sum of squared error) =$\sum_{i=1}^{n}\{dist\ (x_i, x')\}^2$ **$\forall$ $x_i \in C_i$**

  Where $'n'$ is the number of data in the cluster **C_i**, x' is the mean of the cluster.

  Our research work mainly focuses on Single and Complete Linkage clustering as they are the most commonly used Hierarchical clustering methods.

  What is the threshold and why is it used?

  The threshold is a constraint or a limit that is often set by the user based on the application where clustering is used. Whenever the distance between two clusters surpasses the user-defined threshold, the clustering action stops. This helps in getting clusters based on the application and the requirement of the user. Setting a threshold also converges the algorithm faster as now instead of naively clustering all the data into a single cluster, the action can be stopped when the desired number of clusters is obtained.

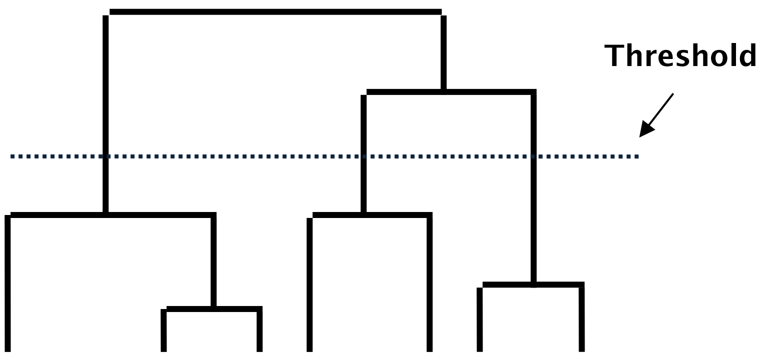


Figure 2: Dendrogram cut at the user-defined threshold

Figure2. shows that the Dendrogram is cut at a higher level with increasing value of threshold i.e. more and more clusters are merged thereby decreasing the number of clusters. Thus, the improper selection of threshold may provide undesirable clusters with a very high or low degree of cohesion. So depending on the application the user should specify a proper threshold to get desirable clusters.

The major disadvantage of Single Linkage clustering is its formation of loosely bound large clusters known as the "*chaining effect*". This is because here clusters are merged if the minimum distance between them lies within the user-defined threshold. Thus, although it makes the minimum within cluster distance in threshold, the maximum intra cluster distance still remains above the threshold.

Hence, where very compact clusters are desirable, Complete Linkage clustering is much more preferred as it produces small clusters with maximum intra-cluster distance (maximum of the distances between all the members of the cluster) within the user-defined threshold.

According to the "Lance-Williams dissimilarity update" formula, the distance between a cluster 'iUj' from any other cluster or object 'k' is given by,

$d(i \cup j, k) = \alpha_i d(i, k) + \alpha_j d(j, k) + \beta d(i, j) + \gamma |d(i, k) - d(j, k)|$ (1)

where 'α_i', 'α_j', 'β', 'γ' are the criterion of agglomeration with values of 'α_i' =0.5, 'α_j' =0.5, 'β' =0, 'γ' =-0.5 for Single Linkage and 'α_i' =0.5, 'α_j' =0.5, 'β' =0, 'γ' =0.5 for Complete Linkage algorithm.

Distance Measure:

The choice of distance metric is a vital task in clustering as a wrong choice of distance metric can assign data to a wrong cluster. Human intuition is limited to the three-dimensional world. As dimensionality increases, the computing distance between all the features becomes more complex. This is called the "Curse of dimensionality ". Choice of distance metric [13] [14] is very crucial in such scenario and depends on the dimension of the input data. The famous L_k norm states that as 'K' decrease the problem of calculating distance at a high dimension can be alleviated. Thus, L_1 norm or Manhattan metric is much more preferable than L_2 norm or Euclidean metric.

1.2. Application of clustering algorithms with special emphasis on hierarchical clustering:

Clustering finds major applications [2][3][15] in the field of data mining, image processing, pattern recognition, outlier detections, recommendation engines, bioinformatics, and biomedical applications like gene recognition, clustering DNA sequences, phylogeny reconstruction, protein structure prediction, etc.

Some major applications of Hierarchical clustering are listed below.

a. bibliographic information retrieval like Clustering Journals based on citation count using Single Linkage clustering [16],
b. Use of single-linkage clustering for analyzing growth rate of GDP [17]
c. Use of Complete Linkage algorithm for "Automatic analysis of malware behavior using machine learning" [18]
d. Using single linkage clustering in health research, such as grouping South

African death causes according to the number of deaths brought on by each ailment [19].

e. Application of hierarchical clustering to identify various SARS-CoV-2 epidemic patterns across Italian regions [20] .
f. “Hierarchical Clustering Analyses of Plasma Proteins in Subjects with Cardiovascular Risk Factors Identify Informative Subsets Based on Differential Levels of Angiogenic and Inflammatory Biomarkers” [21] etc.
g. “Speaker attribution of multiple telephone conversations using a complete-linkage clustering approach” [22].
h. Use of Complete Linkage algorithm for the study of “speaker clustering for speaker attribution in large telephone conversation datasets” [23].
i. Use of Complete Linkage algorithm for speaker diarization [24].
j. Applying Hierarchical Clustering Algorithms for the purpose of Recognition Using Hand Biometrics [25].
k. “Hierarchical cluster analysis in clinical research with heterogeneous study population: highlighting its visualization with R” [26].
l. Application of Hierarchical clustering algorithm to cluster engagement of Nigerian scholars in nanotechnology research between 2010-2020 using Euclidean distance measure [27].
m. Application of hierarchical clustering based on relative gene expression changes to compare the effect of infection between diseases [28].

Our entire research work focusses on two most important Linkage methods of Hierarchical clustering i.e. Single Linkage and Complete Linkage. Our aim is to make these algorithms suitable for large data volume as well as dynamic databases. We have proposed six algorithms in total for our purpose and have shown how each algorithm without that much compromising the Cluster quality makes Single and Complete Linkage algorithms suitable for large databases. We have proposed an hardware implemented version of

Leader algorithm which when applied at the preclustering stage of various well known clustering algorithms like K means, DBSCAN, Average Linkage speeds up the algorithms to a large extent. Chapter 2 discusses the Literature review or the related work of the study, Chapter 3 explains the entire research methodology, Chapter-4 provides a preclustering algorithm for Complete Linkage clustering which reduces the convergence time of the overall algorithm by a large extent, Chapter-5 provides another version of Complete Linkage algorithm which itself is an accelerated method and moreover using the preclustering algorithm as that described in chapter 4 converges much faster than the existing methods, Chapter 6 proposes a Complete Linkage algorithm which is not only suitable for large datasets but also for fast changing dynamic databases.Chapter-7 proposes an hardware implemented accelerated version of Leader algorithm that is well known for being used as preclustering stage of various algorithms like K-means, DBSCAN, average Linkage etc. Chapter-8 provides a Single Linkage algorithm with much reduced time complexity in compare to the existing methods but only suitable for single attributed data. Chapter-9 reduces the disadvantage of the algorithm in chapter-8 and proposes a method which is not only fast but also suitable for multivariate datasets.Chapter-10 finally draws the conclusion and discusses the future scope of the study.

Chapter-2
Literature Review

The traditional SL algorithm computes the distance between every data of the dataset. To calculate and to store these distances in a distance matrix a time complexity of O(n^3) is required where '*n*' is the number of data points [7][29] [30].

The algorithm for the naive Single Linkage (SL) algorithm [31] is as follows -

1. Every data point is considered as an individual cluster each of size unity.
2. The distance between all the cluster pairs is calculated by creating a distance matrix of size $n \times n$. Here dist ($\mathbf{C}_i$, $\mathbf{C}_j$)= min {dist ($\mathbf{C}_i$, $\mathbf{C}_j$)} where $\mathbf{C}_i$, $\mathbf{C}_j$ be any two clusters.
3. This step merges the pair of closest clusters, which reduces the cluster number by one in compare to the previous number of clusters.
4. Steps '2' and '3' are repeated unless all the items are merged into a single cluster of size '*n*' or the minimum inter-cluster distance exceeds the distance at any merging step.

R.Sibson [32] proposes an SL algorithm of O(n^2) and O(n) time and space complexity respectively which is an improvement upon the naïve Single Linkage clustering of O(n^3) and O(n^2) time and space complexity respectively [33][34]. However, the threshold criterion may decrease the overall convergence time as the stoppage criterion reduces the number of iterations.

Gower and Ross (1969) suggested a method to locate SL clusters using Minimum Spanning Tree (MST), as it is pretty comparable to the SL algorithm [33][35]. Using this method, the time complexity for obtaining Single Link clusters is O(n^2) because MST construction requires O(n^2) time.

All these algorithms are highly expensive in clustering high data volume and also for “on the fly” data analysis because of the requirement of the entire database in advance.

Based on the Tolerance Rough Set Model, Patra and Nandi [36] suggested a speedy Single Linkage clustering method that has a time complexity smaller than O(n^2) but does not produce exactly similar clusters as that of the traditional SL method.

A distributed Single Linkage Hierarchical clustering technique (DiSC) based on Map-Reduce [37] is presented by Jin, C. *et al.* which firstly breaks the original problem into a number of overlapping sub problems and then combines the solutions of all the sub problems into a single solution after solving each one of them separately. To get further speedups, parallelization strategies are used.

The algorithm presented by Patra *et al.* lowers the time complexity below O(n^2) although the convergence time may be long if the distance between the majority of the data is greater than half the user-defined threshold [38].

The Leader algorithm in the initial stage uses a threshold of half the user-defined value. If the proximity of the patterns are low relative to the applied threshold, a large number of leaders are likely to be produced with decreasing threshold (half the user-defined value). This in turn increases the time to process the leaders thereby increasing the convergence time of the algorithm.

The Complete Linkage (CL) Clustering algorithm, also referred to as the "farthest neighbour clustering" [39][40][41], measures the distance between two clusters as the maximum of the distances between all its members. To put it another way, if the user specifies a threshold level, the algorithm will only merge two clusters if the maximum distance between them is less than that given threshold. This process thus makes the algorithm highly suitable for getting small and strongly cohesive clusters as all the members within the cluster are required to be within the threshold distance from each other.

This traditional approach of the Complete Linkage algorithm repeatedly forms the $n \times n$ distance matrix at every merging step and thereby is highly time and memory expensive with a requirement of time and space complexity of O(n^3) and O(n^2) respectively. The use of a Priority queue for storing the distances between Clusters can improve the complexity with O(n^2 log n) time and O(n^2) space [30]. Inspired by the comparable technique "SLINK" [32] for Single Linkage clustering, D. Defays [42] created an optimally efficient approach known as CLINK . CLINK has a running time of O(n^2) and a memory requirement of O(n). On current commodity hardware, Althaus E. *et al.* proposes a greedy Complete Linkage Clustering algorithm [43] which can cluster one million points in a day, with a running time of roughly O(n^2) and space of O(n). This approach supports parallelism on shared memory devices. All these methods use the entire dataset to initiate clustering. A novel GPGPU implementation of one-dimensional hierarchical clustering was proposed by Rehn *et al.* [44] with a space requirement and the worst-case time complexity of O(n) and O(n^2/t), where 't' is the maximum number of parallel threads the GPU can support. This algorithm by increasing the level of parallelism and also by utilizing the special property of one-dimensional data, reduces both time and space requirements. However, the limited amount of GPU memory and its suitability for one-dimensional data are two major drawbacks of this algorithm. Davidson, Ian, and Ravi, S.S [45] presents a method which proves that the use of constraints in Hierarchical clustering can be used to improve both the runtime performance and the efficiency of the algorithm.

Jang *et al.* [46] have proposed an approach towards faster and scalable density-based clustering named DBSCAN++ which computes densities of some chosen subset of points instead of working on each sample point thereby speeding up the algorithm. Methods for Speeding up the K-means clustering algorithm to make it more efficient and suitable for big data analysis have also been proposed in [47][48].

The novelty of our research work:

a. The algorithms proposed in our research work uses hybrid clustering methods with an incremental pre-clustering stage followed by the further merging stage resulting in partial clustering of “on the fly” or streaming data analysis.
b. The convergence time of the proposed algorithms is much lower than the traditional approaches without compromising space complexity.
c. The algorithm is suitable for multivariate datasets.
d. The final clusters always satisfy the user-defined threshold and follow the properties of the corresponding Clustering algorithm.
e. An efficient accelerated approach of the Leader algorithm has also been proposed without compromising the cluster quality which when applied at the preclustering stage of various other clustering algorithms speeds up those methods to a large extent.
f. Complete Linkage Algorithm suitable for dynamic datasets have also been proposed and about which no work has been done earlier as per our knowledge.
g. All the benefits of our research work are supported by mathematical proofs and experimental verifications of real-world data.

Chapter-3
Research Methodology

3.1 Introduction: The methodology for research includes theoretical methods, experimental studies, numerical schemes, statistical approaches, and so on. A systematic approach to solving a problem is known as research methodology. It is the science of determining how research should be conducted. Research technique is essentially the procedures by which researchers describe, understand, and predict occurrences. It can also be defined as the study of strategies for gaining knowledge. The methodology of research includes theoretical concepts, experiments, numerical methods, statistical analysis etc. A systematic approach to problem solving is known as research methodology. It's the science of figuring out how to do research. In essence, research technique refers to the procedures that researchers use to describe, explain, and anticipate occurrences. It's also known as the study of knowledge acquisition methods. The goal is to provide a research work plan.

3.2. Hardware vs. software programming languages: The design and operation of digital logic circuits are described using a computer language called hardware description language (HDL) [49] [50].

A hardware description language is a written description consisting of expressions, statements, and control structures, similar to a programming language such as C. The fact that HDLs explicitly include the concept of time distinguishes them from most programming languages. HDLs are crucial components of electronic design automation (EDA) systems, particularly for complicated circuits like ASICS, microprocessors, and PLDs. Despite the apparent similarity between an HDL and a software programming language, there are important distinctions. As a rule, procedural (singlethreaded) programming languages lack syntactic or semantic support for concurrency. HDLs can help processes to run in parallel and can work independently of one another like concurrent programming languages. The simulator's process stack automatically updates whenever the input to the process changes. A compiler (in the case of HDLs, a synthesizer) converts an HDL code listing into a physically realizable gate netlist.

3.3. Language used for hardware design entry: VHDL, Verilog.

The main difference between Hardware description language and software programming language lies in the fact that unlike software programming languages, Hardware description language includes propagation time and signal strengths.

- **VHDL** (VHSIC Hardware Description Language)

VHDL [51] is a hardware description language and a general-purpose parallel programming language that is used to represent the dataflow, behavioral, and structural modelling techniques of digital systems. The Department of Defense (DoD) initially used this language in 1981 as part of the VHSIC programme. This language was first developed in 1983 by IBM, Texas Instruments, and Intermetrics. IEEE standardized the language in 1987.

- **Verilog**

Verilog [52] is a hardware description language (HDL) for modelling electronic systems that is standardized as IEEE 1364. Verilog majorly finds its application for the design and test of digital circuits at the register-transfer level of abstraction.

3.4. Hardware Device Used for the Design Implementation: Field Programmable Gate Array.

An integrated circuit (IC) that can be programmed after fabrication is known as a field-programmable gate array (FPGA) [53]. FPGAs provide a hierarchy of reconfigurable interconnects that allow programmable logic blocks to be "connected together," similar to how multiple logic gates can be inter-wired in different configurations. Complex combinational operations or simple logic gates like AND and XOR can be performed using logic blocks. Memory components, which might be simple flip-flops or larger memory blocks, are included in most FPGA logic blocks. The usage of bidirectional data buses and highly rapid I/Os in FPGA designs makes it difficult to confirm the precise timing of valid data during setup and hold time. To meet these time restrictions, floor planning allows resource allocation within FPGAs. Any logical function that an ASIC may accomplish can be implemented using FPGAs.

Important FPGA components include high-speed serializer-deserializer (SERDES), on-chip resistance-capacitance oscillator, phase-locked loops (PLLs), voltage-controlled oscillators and quartz-crystal oscillator.

3.4.1. FPGA vs. Microcontroller

The most important thing to remember about FPGAs and microcontrollers is that they are two completely distinct devices. A microcontroller, such as an Arduino, already has the chip created for us. Users just develop software in C or C++ and compile it into a hex file, which they then put onto the microcontroller. The programme is stored in flash memory by the microcontroller and will remain there until it is wiped or replaced.

FPGAs, on the other hand, are unique. The circuit is designed by the user. An FPGA can be configured to be as simple as an AND gate or as complicated as a multi-core CPU by the user. A user's design is created by writing HDL code (Hardware Description Language). The user then converts the HDL into a bit file that may be used to programme the FPGA. FPGAs have a minor drawback in that they save their configuration in RAM rather than flash, which means that if they lose power, they lose their settings. Every time the power is turned on, they must be configured.

That isn't as problematic as it appears, because there are flash chips available that will automatically configure the stored bit file when the computer is turned on. Some development boards don't even need a programmer because they automatically configure the FPGA when they boot up.

Microcontroller	**FPGA**
Software programming	Hardware programming
C, C++, C#	VHDL, Verilog
Serial Execution of Instructions	Parallel Execution of Instructions
Slower than FPGA	Faster

Users have designated pins for particular functionality on a typical CPU. Some microprocessors, for example, have only two pins that are used as a serial interface. If a customer wants more than one serial port or wishes to use different pins, the only option is to use software to imitate a serial port rather than purchasing a new chip. That works perfectly, except that the user is wasting valuable CPU time by sending out bits. If a user wants to simulate more than one port, all of the user's processor time is consumed. Because an FPGA allows users to design their own circuits, the user can choose which pins the serial port connects to. This also implies that a user can create as many serial ports as the user wants. The number of physical I/O pins and the size of the FPGA are the sole constraints for users. One of the most intriguing aspects of FPGAs is that the hardware may be designed to function as a CPU. FPGAs are frequently used by companies that create digital circuits, such as Intel and Nvidia, to prototype their chips before manufacturing them.

3.4.2. FPGA vs. ASIC:

The Application Specific Integrated Circuit (ASIC) is a special sort of integrated circuit that is created with a pre-specified function in mind. An FPGA is a type of IC that does not have programming built in during the manufacturing process. After it leaves the production line, an ASIC can no longer be changed. That is why, especially when producing huge quantities of the same ASIC, designers must be certain of their design. Because an FPGA is programmable, users can remodify the functionality whenever required.

3.4.3. Importance of FPGAs.

The majority of the logic circuits in big systems were built using large-scale integrated circuits (LSI) by the early 1980s. The big integrated circuit still needed random "glue logic" or interconnects to help connect them to:

1. Produce control signals
2. Transmission of data signals between subsystems.

The earliest attempt to address this issue resulted in the creation of Custom ICs, which were intended to replace a substantial number of interconnects. Performance was enhanced while system complexity and manufacturing costs were lowered. Custom ICs can have certain drawback. Because of the longer design times, they are relatively highly expensive to develop and cause delays in the introduction of products to the market (time to market).

Here are two types of expenses related to developing unique ICS.

1. Development and design costs
2. Manufacturing costs

Thus, in order to build the entire system on one chip and give the user programmability, FPGAs were created as an alternative to custom ICs.

3.4.4. Internal structure of FPGAs [54]:

FPGAs are consisted of logic blocks which can be configured to do tasks as simple as those performed by transistors or as complex as those performed by microprocessors. It can be used to carry out a number of combinations of sequential and combinational logic operations.

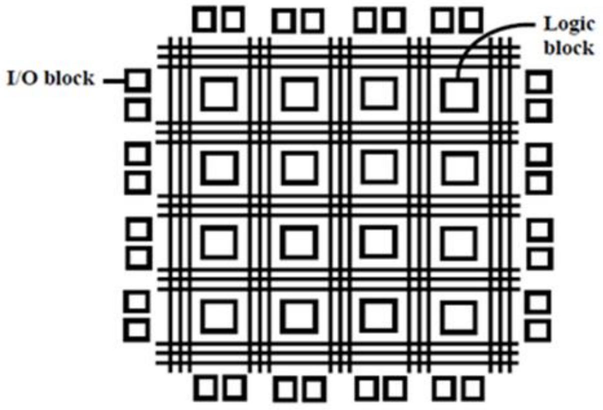


Figure 3: Simplified internal structure of FPGA (Source : Jackson, Chris. (2022). Investigating Serial Microprocessors for FPGAs).

Routing in FPGAs can be achieved by interconnecting various wires via electrically programmable switches. The density of the used logic blocks and the amount of space required for routing are often traded off when determining the number of segments needed for interconnection. Fig. 3 depicts the simplified internal architecture of an FPGA with routing.

3.4.5. FPGA DESIGN FLOW

The steps of an FPGA's general design flow are as follows:

System design:

The designer must now decide which of the operations must be implemented on an FPGA and to figure out how to integrate those functionalities with the remaining system.

Integration of I/O with the system as a whole

The output streams of the FPGA are then merged/fed to the printed circuit board (PCB).

Design Specifications

The functionality of the design can be designed using Hardware Description Languages like VHDL OR Verilog.

Synthesis

After the design has been defined, the design is implemented on a specific FPGA using CAD tools. Synthesis includes Placement and routing preceded by generic optimization, slack optimizations, and power optimizations. Partition, Place, and route are three aspects of implementation. The bit-stream file is the result of the design implementation process.

Verification of the Design

The simulator then uses the bit stream file, which simulates the functionality of the design and reports if there is any fault in the desired behavior of the design. The maximum clock frequency of the design is found using timing tools. Finally the design is now loaded into the FPGA chip, and the real-world testing is conducted.

Hardware design and development

The method of developing digital logic is quite similar to that of developing embedded software. High-level hardware description languages (often VHDL or Verilog) are used to write descriptions of the structure and behavior of hardware, which are then translated into executable code, compiled and executed. Fig 4. Depicts the general process of the hardware development.

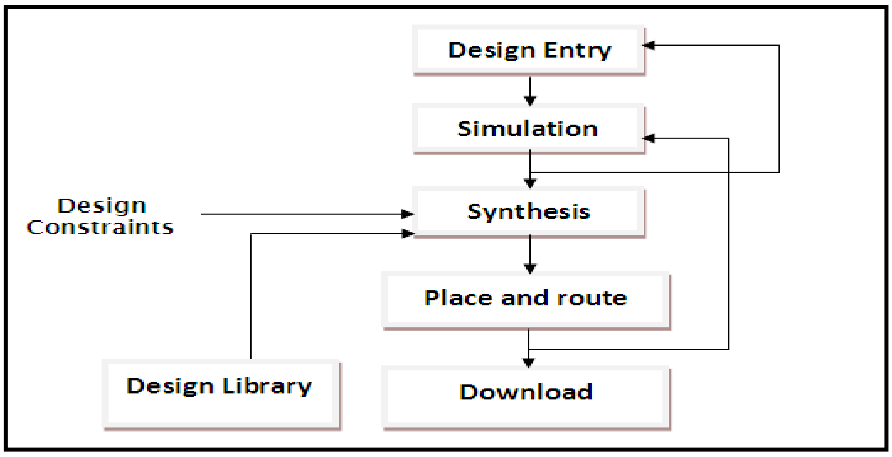


Figure 4: Steps for the hardware development for programmable logic. (Source: https://barrgroup.com/images/glossary/progLogic-figure3.gif)

Place and route translates the logical structures in the netlist into the real macrocells, connections, and input and output pins. A bitstream is the outcome of the place and route procedure. The bitstream is the binary data fed to the FPGA to implement a specific hardware design.

3.4.6. Software Tools For Synthesizing The Design: Xilinx ISE

Xilinx ISE (Integrated Synthesis Environment) [55] is a software programme developed by Xilinx for synthesizing designs, performing timing analysis, simulating the design's output to various stimuli. Xilinx ISE is incompatible with FPGA products from other manufacturers since it is strongly connected to the architecture of FPGA from Xilinx. Xilinx ISE is typically used for circuit synthesis and design. Since 2012, Vivado Design Suite has replaced Xilinx ISE as the preferred option. It performs the same functions as ISE while offering further functionality for system on chip development. The final version of ISE, version 14.7, was released by Xilinx in October 2013.
In our research, the design has been checked and verified using Xilinx ISE design suite 14.4 and implemented in Kintex 7 (KC705) using the Xilinx Vivado platform and VHDL (VHSIC Hardware Description Language).

Software Implementation of our algorithms [56]:

The "C" language is a middle-level, procedural, general-purpose programming language used to create text editors, operating systems, gaming apps, assemblers, compilers, and other tools. The key benefits of the C programming language over other languages are its user-friendly platform, speed of execution, expanded library, and variety of built-in functions, among other things. As hardware implementation in FPGA is never practicable due to resource limitations, creating quick and effective software implementations of clustering algorithms makes its use widespread in a variety of applications. The development of hardware-software co-designed algorithms is currently a common research trend, therefore the software and hardware-implemented methodologies employed in our research may have possible future applications. In our research work, the algorithms have been implemented on (ChIDE compiler) and have been executed on a PC with Intel(R) Core (TM) i3-6006U CPU @ 2.00Ghz processor, 8 GB RAM, and 64bit OS.

3.4.7. Tool for comparing clustering algorithms:

Rand Index or Rand score [57] is a popular statistical tool to measure the similarity between two clustering results. In other words, it measures the % of correct decisions. If 'N' data points exist, the total no of decisions is N(N-1)/2 as each decision involves a pair of data points.

Rand Index = $\frac{TP+TN}{TP+TN+FP+FN}$

TP= Relevant data in clusters.
TN= Irrelevant data omitted
FP=Irrelevant data retrieved
FN=Relevant data omitted

	Same cluster	**Different cluster**
Same class	**TP**	**FN**
Different class	**FP**	**TN**

The value of the Rand Index always ranges between (0,1) with '1' implying total similarity and '0' implying total dissimilarity. Thus, a rand Index very close to '1' is always preferred if the similarity between two clustering algorithms is desired.

3.4.8. Data set used for Verification:

To experimentally verify the advantages and benefits of our algorithms we have applied them on real-world datasets as obtained from the UCI Machine learning repository. We have also created data using random number generators based on our applications.

Table 1: Experimental dataset used in our research work

Name of the data Set		No of instances	No of attributes	Attribute type
"Letter" [58]		20,000	16	Integer
"Shuttle" [59]		58,000	9	Integer
"Pendigits" [60]	Testing	3,498	16	Integer
	Training	7,494	16	Integer
"Gaussian dataset"- 1 [61]		10,000	1	Real
"Gaussian dataset"- 2 [61]		40,000	1	Real
"Distillate flowrate" [62]		44,640	1	Real
"Random" [63]		100	1	Integer

Chapter-4

An Efficient Algorithm for Complete Linkage Clustering with a Merging Threshold

The complete Linkage algorithm also known as the farthest neighbor principle is an efficient Hierarchical clustering algorithm that finds special applications when compact spherical clusters are required. The main disadvantage of the Complete Linkage algorithm is its high time complexity which makes the algorithm unsuitable for clustering large datasets. This paper presents an efficient preclustering algorithm for Complete Linkage clustering that speeds up the convergence of the CL algorithm to a large extent. This algorithm takes a clustering decision without comparing the distance between the incoming data from all the members of the clusters. This preclustering algorithm can be used before any Complete Linkage algorithm to speed up the method. This work uses triangle inequality to derive a suitable threshold to find clusters having maximum intra-cluster distance within the threshold. These preclusters when subjected to any Complete Linkage algorithm, produces clusters following all the properties of the CL algorithm. The preclustering stage employs a leader algorithm which is an incremental partitional clustering algorithm that scans every data point only once and assigns data either as a leader or as a follower to that leader. The algorithm chooses data as a follower to a leader if it comes in threshold to that leader, if it does not come in threshold to any of the leaders, then it will be selected as a new leader.

The time complexity of the Leader algorithm is O(*mn*) with '*m*' being the number of leaders and '*n*' being the total number of data points. The space complexity is O(*n*). Patra et.al [64] designed an accelerated version of the Leader algorithm using triangle inequality to avoid redundant distance computations while taking a clustering decision without changing the final cluster quality.

Triangle inequality is a highly efficient tool that provides an upper bound to the distance between a pair of data points 'p' and 'q' with respect to another point 'r' such that,

$$\forall p, q, r \in D: |\text{dist}(p, q)| <= |\text{dist}(p, r)| + |\text{dist}(q, r)| \qquad (2)$$

Here ‘dist’ is a suitable metric over the domain D.
On the other hand, the reverse triangular inequality provides a lower bound to the distance between the points ‘p’ and ‘q’ such that,

$\text{dist}(p, q) >= |\text{dist}(p, r) - d(q, r)|$ (3)

If $\text{dist}(q, r) >= 2\times\text{dist}(p, r)$,

$\text{dist}(p, q) >= \text{dist}(p, r)$. (4)

Replacing p, q, r with x_i, l_j, l_1 we get,

$\text{dist}(xi, l_j) >= \text{dist}(x_i, l_1)$, if $\text{dist}(l_j, l_1) >= 2\times\text{dist}(x_i, l_1)$ (5)

Thus, if $\text{dist}(x_i, l_1)$ is $> \zeta$, then eqn. 5 shows $\text{dist}(x_i, l_j) > \zeta$, which removes the necessity to compute $\text{dist}(x_i, l_j)$ directly. This process requires storing the distances between all the leaders which can be done hand in hand while assigning a new leader while performing the algorithm.

Algorithm 1: Leader algorithm (D, h)

```
D = {x_1,x_2,.....x_n}  /D – Input dataset, n- size of the dataset , h- Used threshold/
L={ } / Leader Set /

Loop1:For any unscanned pattern $x_i \in D$
{
  Loop2:For any unchecked Leader $l_j \in L$
  {
    If dist($x_i$,$l_j$) <= h
      Assign $x_i$ as follower to $l_j$
      Break
    Else
      Repeat Loop2 with next unchecked leader
    }
If ($x_i$ cannot be assigned as a follower)
  {
    L=LU$x_i$  /Assign $x_i$ as a new leader /
  }
Repeat Loop1 with the next unscanned pattern
}
Output L with followers
```

4.1. Determining the threshold for the preclustering step

Complete Linkage (CL) Clusters follow the two most important properties –

a) The maximum intra-cluster (within cluster) distance must be within the threshold.

b) The maximum inter-cluster (cluster to cluster) distance must be more than the threshold.

Let us consider two followers f_1, f_2 of a pre-cluster with leader ‘l’ as shown in Fig.5,

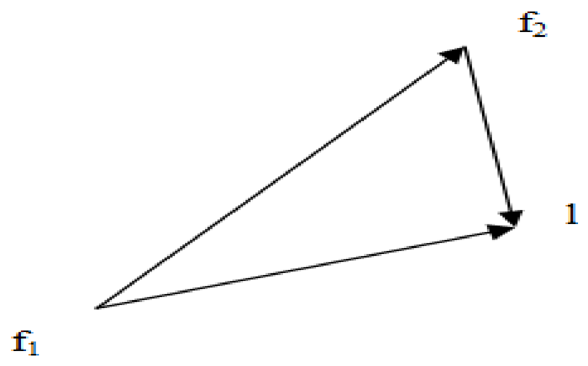


Figure 5: Maximum intra-cluster distance computation at the preclustering output

$d(f_2, f_1) <= |d(f_1, l)| + |d(f_2, l)|$ (6)

The preclustering stage being the Leader algorithm always assigns data as a follower to a Leader, if it comes within the threshold to the Leader. Therefore, if the threshold used at the preclustering stage is considered as half the user-defined value, then

$\therefore d(f_2, f_1) <= \frac{\zeta}{2} + \frac{\zeta}{2}$ (7)

$d(f_2, f_1) <= \zeta$ (8)

Eqn. ‘8’ is true for any two followers of a pre-cluster and hence the maximum intra-cluster distance is always within the threshold. Thus, by considering the threshold for the preclustering step as half the user-defined value, the maximum intra-cluster distance of the preclusters can be maintained within the threshold. This satisfies the first property of the CL clusters.

The Leader algorithm compares a pattern to all the existing leaders unless and until it is assigned as a follower or as a new Leader. The pattern may be assigned as a follower to more than one leader of the Leader set. In other words, a follower of a Cluster Cj may be in the threshold to Leader of another Cluster C_i. If ‘f_i’ be the follower of cluster ‘C_i’ with leader ‘l_i’ and ‘f_j’ be the follower of cluster ‘C_j’ then,

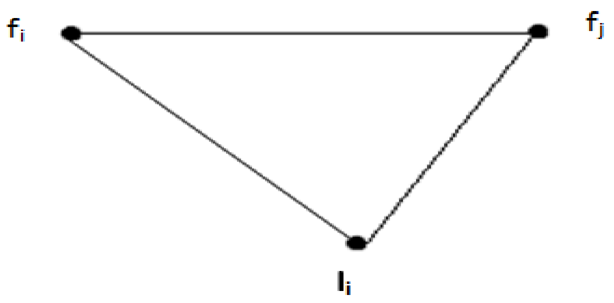


Figure 6: Inter-cluster distance computation at the output of the preclustering stage

$d(f_i, f_j) <= |d(f_j, l_i)|+|d(l_i, f_i)|$ (9)

If $d(f_j, l_i)$ is also within $\frac{\zeta}{2}$,

$d(f_i, f_j) <= \zeta$ (10)

When eqn. 10 holds for all followers of C_j and C_i, the maximum inter-cluster distance between two clusters remains within threshold. The Complete Linkage algorithm at the second stage merges such clusters (having maximum inter-cluster distance within threshold) to meet the second criteria of CL clusters.

Algorithm 2: Accelerated Leader Algorithm (D, $\frac{\zeta}{2}$)

```
D = {$x_1$,$x_2$.....$x_n$} /D being the input dataset, ζ being the user-defined threshold/
L = {}  /Leader set is initially NULL/
For any unscanned pattern $X_i \epsilon D$
 {
  If d($x_i$,$l_1$) <= $\frac{\zeta}{2}$ then
  {
   Assign $x_i$ as a follower to $l_1$;
  }
  Else
    {
      For any unchecked leader $l_j \epsilon L$
       {
        If(d($l_j$, $l_1$) >= 2 d($x_i$, $l_1$)) then
        {
         Repeat loop with next unchecked Leader;
         }
         Else
           {
           Compute d($x_i$, $l_j$)
             If d($x_i$, $l_j$) <= $\frac{\zeta}{2}$
             {
              Assign $x_i$ as a follower of $l_j$;
               Break; /Exit loop/
             }
            Else
              {
              Repeat loop with next unchecked Leader;
                }
              }
            }

        If(the pattern is not clustered)
         {
          L = LU$x_i$; /Assign $x_i$ as a new leader/
          Store d($x_i$, $l_1$)
          }
        }
   i = i + 1; /Fetch the next unscanned pattern/
  }
   Output L with followers.
```

4.2 Time and Space Complexity of the algorithm

The time complexity of the preclustering stage is $O(mn)$ for 'm' being the number of preclusters and 'n' being the size of the dataset. The space complexity is $O(n)$ for storing the leaders with the followers.

4.3. The advantages and disadvantages of the algorithm

a. Suitability for streaming data clustering:

The preclustering stage completes a part of the Clustering job before subjecting the patterns to the complete Linkage algorithm making the process easier and faster. The incremental approach does not require the entire dataset in advance unlike the well-known variants and is suitable for partial clustering of streaming data because of its high speed.

b. The total number of distance Computations saved:

The time complexity of the algorithm is although $O(n^2)$ is similar to the CLINK algorithm, the convergence time will be lower unless all the pre-clusters are singletons. The accelerated approach of the Leader algorithm as shown in Table 2 of Section 4.4 saves a lot of distance Computations by avoiding the comparison of a pattern with all the Leaders to take a Clustering decision. The Leader algorithm only computes the distance between an input pattern and the Leaders to take a Clustering decision and the dissimilarity matrix at the second stage only computes the inter-cluster distances, therefore the follower-to-follower distance computations within a cluster is saved in both the stages of the algorithm. This is irrespective of the type of data and the user-defined threshold, confirming the usefulness of our algorithm in all cases.
Therefore, the total number of distance computations saved by the proposed method in comparison to those algorithms where pair-wise distance computations between all the input patterns are required is given by,
Total no. of distance Computations Saved= *Computations saved by triangle inequality at the preclustering Stage* **+** *within-cluster follower to follower distance Computations at the second stage.*

c. Dependence of Convergence time of the algorithm on the user-defined threshold:

The decrease in the number of preclusters with threshold confirms the increase in their size with the increase in the threshold. It should be noted that with the increase in the size of Preclusters, the convergence time of the entire algorithm should decrease as more and more within-cluster follower to follower distance Computations will be saved at the second stage of the algorithm. However, the choice of threshold is application-specific and it is the sole responsibility of the user to choose the correct threshold for good clusters.

d. Suitability for Clustering Large datasets:

Increasing the size of the input dataset will increase the number of distance Computations for the Clustering action. Hence the Convergence time of the algorithm increases with an increase in the size of the input data. The proposed algorithm saves a lot of redundant distance computations, thereby making it suitable for Clustering large datasets which is a major challenge in these years in the field of data mining.

e. Dependency on the quality of the Clusters on the ordering of the Patterns:

The Leader algorithm compares a pattern with the Leaders unless and until a Clustering decision is taken. Hence, whether an algorithm will be assigned as a Leader or a follower depends on the existing Leaders of the Leader set which in turn depends on the order of the incoming patterns. A Leader formed earlier usually gets more priority of being compared to an input pattern in comparison to the other Leaders. This may cause the algorithm to miss a chance of assigning an input pattern to a nearer Leader when the pattern comes in threshold to more than one Leader. Thus, the ordering of the input data may vary the membership of a pattern from one cluster to another at the preclustering stage. This affects the final quality of the clusters and hence the final set of Clusters although formed following the user-defined threshold and satisfies the properties of the Complete Linkage algorithm, may not be identical to that produced by the Classical Complete Linkage algorithm.

4.4. Experimental Results:

1) Table 2 shows that the accelerated Leader algorithm saves a lot of distance computations in comparison to the ordinary Leader algorithm making the overall process faster. It should be noted that the threshold used for the algorithm is half the user-defined value to generate Complete Link clusters.
2) Fig.7 shows that the increase in threshold decreases Cluster number at the output of the Preclustering stage. This, in turn, decreases the compactness of the clusters, and hence it is the sole responsibility of the user to choose the correct threshold for getting desirable clusters.

Table 2: Speedup achieved by the Preclustering step when applied on large datasets (Threshold used=half the user-defined value)

Data Set	Method	User defined Threshold	Distance Computations(million)	Time (mins)
Letter	Leader	1	180.48	851.126
	Acc Leader		19.38	132.740
	Leader	3	134.87	380.314
	Acc Leader		42.74	171.568
	Leader	5	53.60	148.210
	Acc Leader		26.55	94.27
	Leader	7	17.22	46.80
	Acc Leader		10.91	35.32
Shuttle	Leader	3.5	491.49	1219.56
	Acc Leader		38.87	255.07
	Leader	5.5	220.64	372.80
	Acc Leader		24.57	121.173
	Leader	7.5	103.51	181.51
	Acc Leader		13.98	57.94
	Leader	9.5	55.19	92.86
	Acc Leader		8.70	33.14

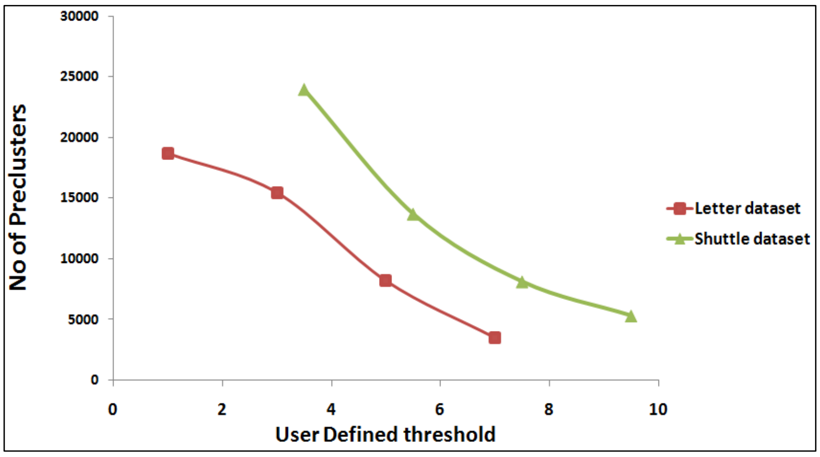


Figure 7: Variation of Cluster number with Threshold at the Preclustering output.

4.5. Conclusion

The paper presents the use of the accelerated Leader algorithm as an efficient preclustering technique which when applied before any Complete Linkage algorithm, speeds up the convergence to a very large extent. The accelerated approach of the preclustering step uses triangle inequality to avoid a lot of redundant distance computations and produces partial complete Linkage clusters in a very short time. These preclusters, when applied to the Complete Linkage algorithm at the second stage, give the final set of clusters obeying the user-defined threshold. The incremental approach of the preclustering step makes it highly suitable for the partial clustering of streaming data along with its collection. Moreover, it should be noted that the preclustering step only computes the distance between an input pattern and the leader and the second stage computes only the inter-cluster distances. Therefore, the algorithm saves within-cluster follower-follower distance computations in both the stages of the algorithm thereby speeding up the entire technique in all cases. The accelerated approach makes it suitable for clustering large datasets as distance computations tend to increase with the increase in data and the preclustering algorithm is capable of taking a clustering decision even without comparing a pattern with all the members of the clusters.

Chapter-5

An Efficient and Speedy approach for Hierarchical Clustering using Complete Linkage Method

The proposed method is a continuation of the method proposed in chapter-4 where we have proposed a second stage that further removes the redundant distance computations by reducing the convergence time of the Complete Linkage clustering algorithm to a large extent as confirmed by the experimental results.

5.1. The necessity of the Final Merging stage

Although the pre-clustering stage of the algorithm satisfies the first property of the Complete Linkage algorithm, it may be possible that the maximum inter-cluster distance between any two clusters still be within the threshold, thereby disobeying the second property of the Complete Linkage algorithm. Let C_j and C_i are pre-clusters with any follower f_j and f_i corresponding to Leaders l_j and l_i respectively.

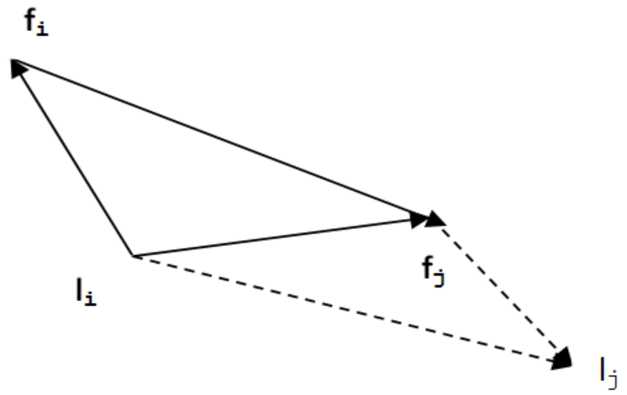


Figure 8: Maximum Inter-cluster distance calculation using Triangle Inequality

By triangle inequality,
dist (f_j, f_i) <= dist (f_i, l_i) +dist (l_i, f_j) (11)

$\text{dist}(l_j, l_i) \leq \text{dist}(l_i, f_j) + \text{dist}(l_j, f_j)$ (12)

Case-1: C_j has been formed before C_i

Assumption 1: 'f_j' has not been compared with 'l_i' following the priority rule of Leader comparison, hence there is a possibility that $\text{dist}(l_i, f_j) \leq \zeta/2$.

Assumption 2: Similarly, although $\text{dist}(f_i, l_j) > \zeta/2$ (else f_i should be a follower of l_j), it can be $\leq \zeta$.

If assumption 1 is true, the RHS of eqn. 11 and 12 will be less than ζ as $\text{dist}(f_i, l_i) \leq \zeta/2$ as the leader algorithm uses threshold of half the user defined value.

Thus $\forall f_i \in C_i$, $\forall f_j \in C_j$, if assumptions 1 and 2 are true, the inter-cluster distance between C_i and C_j will be $\leq \zeta$.

Case 2: C_i has been formed before C_j

Assumption 1:

'f_i' has not been compared with 'l_j' following the priority rule of Leader comparison, hence there is a possibility that $\text{dist}(f_i, l_j) <= \zeta$.

$\text{dist}(l_i, f_j) > \zeta/2$, else f_j would be a follower of Leader l_i.

Say, $\text{dist}(l_i, f_j) = \zeta/2 + \varepsilon_2$, $\varepsilon_2 > 0$

By Leader algorithm,

$\text{dist}(f_i, l_i) = \zeta/2 - \varepsilon_1$, $0 \leq \varepsilon_1 \leq \frac{\zeta}{2}$

$\text{dist}(l_j, f_j) = \zeta/2 - \varepsilon_3$, $0 \leq \varepsilon_3 \leq \frac{\zeta}{2}$

Thus,

$\text{dist}(f_i, l_i) + \text{dist}(l_i, f_j) = \zeta/2 - \varepsilon_1 + \zeta/2 + \varepsilon_2 = \zeta - (\varepsilon_1 - \varepsilon_2)$ (13)

Again,

$\text{dist}(l_i, f_j) + \text{dist}(l_j, f_j) <= \zeta/2 + \varepsilon_2 + \zeta/2 - \varepsilon_3 = \zeta - (\varepsilon_3 - \varepsilon_2)$ (14)

Assumption 2:

If, $\varepsilon_1 > \varepsilon_2$,

Using eqn. 11 and 13,

$\text{dist}(f_j, f_i) \leq \zeta$

Assumption 3:

If, $\varepsilon_3 > \varepsilon_2$,

Using eqn. 12 and 14,

$\text{dist}(l_j, l_i) \leq \zeta$

Assumption 4:

Similarly, although $\text{dist}(l_i, f_j) > \zeta/2$ (else f_j should be follower of l_i), it can be $\leq \zeta$.

If the above assumptions are true for all leaders and followers of C_i and C_j, the maximum inter-cluster distance between them should be within the threshold, thus disobeying the 2nd property of the Complete Linkage algorithm as given in Section 4.1.

The second stage of the algorithm can be applied to these pre-clusters to produce the final set of complete Linkage clusters with both within-cluster and inter-cluster distance in and above the threshold respectively.

5.2. Calculation of the Maximum Inter-Cluster distance

The maximum inter-cluster distance between any two clusters C_j and C_i depends on four distances:

1. $(dist(l_j, l_i))$
2. $\max(dist(l_j, f_i))\ \forall f_i \in C_i$
3. $\max(dist(f_i, f_j))\ \forall f_i \in C_i, \forall f_j \in C_j$
4. $\max(dist(l_i, f_j))\ \forall f_j \in C_j$

This step of the algorithm initially computes the maximum distance between any member of C_i from the leader of C_j directly and if found in the threshold, it employs triangle inequality to predict the distance between C_i from all the followers of C_j without computing the distances directly at every case. This method speeds up the process to a large extent as it avoids a lot of redundant distance computations before taking a clustering decision. If 'm' denotes any member of cluster C_i,

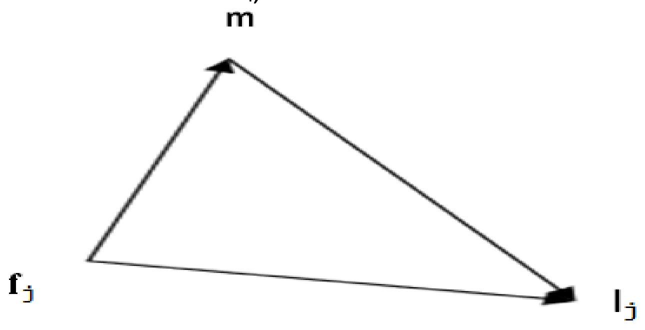


Figure 9: Using triangle inequality to find max_dist between C_j from all followers of C_i

$dist(m, f_j) <= dist(m, l_j) + dist(l_j, f_j)$ (15)

Now, $\forall f_i \in C_i, \forall f_j \in C_j$,

$\max\{dist(m, l_j)\} = \max\{dist(l_i, l_j), \max(f_i, l_j)\}$

By Leader algorithm, irrespective of the order of formation of pre-clusters, $dist(l_i, l_j) > \zeta/2$

$\therefore \max\{dist(m, l_j)\} > \zeta/2$

Thus, by eqn. 15,

$\max\{dist(m, f_j)\} <= \max\{dist(m, l_j)\} + \max\{dist(l_j, f_j)\}$

$\therefore \max\{dist(m, f_j)\} <= \zeta/2 + \boldsymbol{\varepsilon}_{max} + \zeta/2 - \boldsymbol{\varepsilon}'_{min}$, $\boldsymbol{\varepsilon} > 0$, $0 < \boldsymbol{\varepsilon}' < \zeta/2$

$=> \max\{dist(m, f_j)\} <= \zeta + \boldsymbol{\varepsilon}_{max} - \boldsymbol{\varepsilon}'_{min}$

$\therefore \max\{dist(m, f_j)\} <= \zeta - (\boldsymbol{\varepsilon}'_{min} - \boldsymbol{\varepsilon}_{max})$

If $\boldsymbol{\varepsilon}'_{min} >= \boldsymbol{\varepsilon}_{max}$ then,

$\max\{dist(m, f_j)\} <= \zeta$ (16)

If $\varepsilon'_{min} < \varepsilon_{max}$, then it is required to compute max {dist (m, f_j)} directly to determine whether max(dist (m, f_j)) is within the user-defined threshold or not.

$\varepsilon'_{min} = \zeta/2 -$ max {dist (l_j, f_j)}
$\varepsilon_{max} =$ max {dist (m, l_j)}- $\zeta/2$

Thus, calculating ε'_{min} requires the maximum distance of all the followers from their corresponding leader which can be stored hand in hand while performing the accelerated Leader algorithm. On the other hand, ε_{max} requires the maximum distance of C_i from the leader of C_j which is computed directly at the very first step of this final merging stage.
To calculate the distance between all the Clusters a "*p*-1×1" distance matrix has been formed, with '*p*' being the number of Pre-Clusters provided by the first half of the proposed method. Each row and column of the matrix corresponds to a pre-cluster and every cell is assigned with a flag that goes high or low depending on the proximity between the clusters at the corresponding row and column. The method starts with any pre-cluster corresponding to the column and scans the other "*p*-1" rows to check which of the pre-clusters can be assigned as a neighbor to that cluster. Any two clusters which are found in threshold distance are treated as neighbors and it should be noted that here distance between any two clusters refers to the maximum inter-cluster distance, which is always computed using the method as explained above in this section. In every case, whenever the distance between any two clusters is found within the threshold, the corresponding flag becomes high else goes low. If any cluster corresponding to a row is denoted by C_j and that corresponding to the column is denoted by C_i, then after scanning the entire Column, any C_j with flag high will be treated as a neighbor to C_i and will be merged to it. The newly merged C_j at every step if denoted by C_{merged}, then the next step involves checking the neighborhood of the rest of the C_j^s with the high flag from the C_{merged}. Thus, here we are trying to produce the final cluster by merging the clusters which are all neighbor to themselves. Here the term 'neighbor of a cluster' refers to a cluster in threshold to it. This is because Complete Linkage Clusters always have their maximum intra-cluster distance within the threshold i.e., all the members must be within the threshold distance from each other. As all the C_j^s which are in threshold to C_i may not be in threshold to C_{merged}, hence at every step, it is required to check the neighborhood of C_{merged} with all other C_j^s having high flag before merging any C_j to $C_i U C_{merged}$. While scanning C_j^s if a C_j is not found in threshold to C_{merged}, the flag of the corresponding cell will go low. At the next step, the last merged C_j will be the new C_{merged}, and again the distance of the rest of the C_j ^{s} with the high flag will be checked from this C_{merged}. This process continues unless no more C_j^s with high flags is left to be

compared with C_{merged}. The entire process then again restarts with a C_j with flag low as a new C_i (denoted as C_{column}) .
Whenever a merging is done or a C_{column} is assigned, the corresponding row is removed from the matrix. Thus, with every Cluster assignment, a row decreases, and the algorithm converges when all the rows are deleted.
The whole procedure used in this final stage is depicted in Algorithms 3 and 4 respectively.

Algorithm 3: Final Merging stage (ζ)

```
Ci = Pj for j=1; /P1 be the first precluster/
C*= {Ci} ;/C* be the final set of clusters/
Cj=Pi; for all j's for which Ci≠Cj
Cmerged= Ci ;

i=1;
k=i+1;

set flag of all Cj high; /so that Ci compares itself with all Cj at the first step/
while (all rows are not removed)
  {
   while (all Cjs with flag high are not compared with Current Cmerged)
    {
      If max_dist (Cmerged, Cj_leader) <= ζ then
       {
        Find εj_max / εmax corresponding to Cj/
        Find ε'j_min /ε'min corresponding to Cj/
          If (ε'j_min ≥ εj_max)
            {
             Set flag_Cj high
             no_of_Cj_with_flag_high ++
             }
         Else
            {
           Compute max_dist (Cmerged, Cj_follower)
             If max_dist (Cmerged, Cj_follower) ≤ ζ
              {
               set flag_Cj high
                no_of_Cj_with_flag_high ++
              }
             Else
               {
                set flag_Cj low
```

```
                      no_of_$C_j$_with_flag_high - -
                      }
                    }
                 }
     Else /if max_dist ($C_{merged}$, $C_{j\_leader}$) > $\zeta$ then/
       {
       Set flag_$C_j$ low;
       no_of_$C_j$_with_flag_high - -
       }
      }
  If (no_of_$C_j$_with_flag_high > 0)
    {
      for (first $C_j$ with flag high in the column) do
       {
        $C_i$=$C_i$U$C_j$
        $C_{merged}$= $C_j$
        Remove $C_{merged}$ row from matrix;
        no_of_$C_j$_with_flag_high = no_of_$C_j$_with_flag_high- -;
        }
      }
   Else
      {
     For (first $C_j$ with flag low in the column) do
        {
          i=k;
          k=k+1;
          Assign $C_i$=$C_j$;
          $C_{merged}$=$C_i$ ;
          C*=C*U$C_i$;
          Remove $C_{merged}$ row from matrix;
          Set flag_of_all_$C_j$ high;/ so that $C_i$ compares itself with all $C_j$ at the first step/
         }
       }
   }
Output C*;
```

Algorithm 4: To find max {dist (m, l_j)}

```
$l_i$- leader of any cluster corresponding to a column
$f_{i'}$- A follower of $l_i$ for any value of i'
$l_j$- Leader of any cluster corresponding to a row
Max_distance = dist ($l_i$, $l_j$)
i'=1
While all $f_{i'}$ are not compared to $l_j$
   {
     If ( dist ($f_{i'}$, $l_j$) > Max_distance)
     Max_distance = dist ($f_i$', $l_j$)
      i'=i'+1
      else
      i'=i'+1
     }
Output Max_distance;
```

5.3. Time and Space Complexity of the algorithm:

5.3.1. The Time and Space Complexity of the Pre-Clustering stage:

If '*p*' is the number of Pre-clusters provided by the algorithm and '*n*' is the size of the dataset, then the worst-case time Complexity of the Accelerated Leader algorithm [64] will be the same as the Leader algorithm i.e., O(*pn*). The space Complexity is O(*n*) for storing all Leaders along with followers.
The Pre-clustering stage requires storing the maximum distance of all followers from their corresponding Leader to calculate the maximum Inter-cluster distance using triangle inequality as described in Section 5.2. The algorithm for finding the maximum distance of a follower from its Leader is depicted in Algorithm 5.

Algorithm 5: To find the farthest follower of a Leader

```
$f_1$- First follower assigned to any Leader $l_k$
Max_dist = dist($f_1$,$l_k$)
k'=2
**While** all followers are not assigned to $l_k$
  {
     For any new follower $f_{k'}$ ,
     **if** (dist ($f_{k'}$,$l_k$) > Max_dist)
        {
         Max_dist = dist($f_{k'}$,$l_k$)
          k'=k'+1
         }
     **Else**
          (
          k'=k'+1
          }
   }
Output Max_dist ;
```

If '*f* ' be the average number of followers of each pre-cluster then for each pre-cluster O(*f*) comparisons are done to find the maximum distance of a follower from the Leader. For '*p*' pre-clusters, the time Complexity is O(*pf*). Thus, the total Time complexity of the Pre-clustering stage is O(*pf*) + O(*pn*) ≈ O(*pn*) as $f << n$.
The space Complexity for storing 'Max_dist' for each Pre-cluster is O(1), making the total Space Complexity for '*p*' pre-clusters is O(*p*)+O(*n*) ≈ O(*n*).

5.3.2. The Time and Space Complexity of the Final Merging stage:

To calculate the Time Complexity, here we have made 3 assumptions to get the maximum number of distance computations:

a. The average size of each pre-cluster is **Q**, to make the calculation easy.
b. At every iteration whenever a column is scanned by C_{column}, '*s*' neighbors are found.
c. Whenever a C_{merged} scans all the C_{rows} with a high flag, all the flags are turned low.
d. For all the cluster to cluster comparisons, $\boldsymbol{\varepsilon}'_{min} < \boldsymbol{\varepsilon}_{max}$ i.e. max{dist (m, f_j)} is to be computed directly.

The third assumption increases the number of distance computations as scanning a column with C_{merged} requires maximum '*s*-1' inter-cluster comparisons while scanning with $C_{coloumn}$ requires comparing all the remaining C_{rows} with flag low. So if a C_{merged} can make all flags low, then the next scan will be with C_{column} (any C_{row} with flag low) instead of C_{merged}, thereby increasing the chance of more distance Computations.
The process starts with a C_{column} that scans '*p*-1' Clusters which requires $Q^2(p-1)$ computations. When the entire column is scanned, any C_{row} with a high flag will be new C_{merged} and that will scan only the neighbors of C_{column} obtained in the previous scan. This will require $Q^2(s-1)$ computations as per 2nd assumption. Now, if this step makes all the high flags low as per the 3rd assumption, then the next step is to assign a new C_{column} which will again scan the entire column with '*p*-3' rows as whenever a C_{merged} or C_{column} is assigned, the corresponding rows are removed. This step requires $Q^2(p-3)$ Computations and again finds 's-1' neighbors. The step will be again followed by scanning all C_{rows} with high flag by a C_{merged} and the cycle will continue unless all rows are removed. It should be noted that while finding the {*max* {dist (m, l_j)}} as discussed in section 5.2, only 'Q-1'distance computations are required. As the leader to leader computations of all preclusters was previously stored in a matrix to employ triangle inequality in the preclustering algorithm so here only the distance of the followers of a C_{column} or C_{merged} is computed from the leader of a C_{row} and hence instead of 'Q' it only requires 'Q-1' computations in every case.
Thus, the maximum number of distance computations required in the final stage before the algorithm converges will be-

$$=[Q^2(p-1)+Q^2(s-1)+Q^2(p-3)+Q^2(s-1)+\ldots\ldots\ldots+Q^2(p-(p-s-2))+Q^2(s-1)+Q^2(p-(p-s))] + [Q^2(s-1)+Q^2(s-2)+Q^2(s-3)+\ldots\ldots+Q^2(s-(s-1))]$$
$$-$$
$$[(p-1)+(s-1)+(p-3)+(s-1)+\ldots\ldots+(p-(p-s-2))+(s-1)+(p-(p-s))] + [(s-1)+(s-2)+(s-3)+\ldots\ldots+(s-(s-1))] \qquad (17)$$

The first two series in eqn. 17 provide the maximum number of distance computations that can be required to compute the inter-cluster distances and the last two series give the maximum number of computations that can be saved by avoiding leader to leader comparisons between a C_{column} or C_{merged} from all C_{rows} with low flag and high flag respectively. The method of calculating this maximum distance is depicted in Algorithm 4. This distance is required to find $\boldsymbol{\varepsilon}_{max}$ as described in section 5.2. for applying triangle inequality at this second stage.
If, $Q^2(s-1)$ and $(s-1)$ of the first and third series in eqn. 17 repeats 'Z' times then eqn. 17 becomes,

=**[**$Q^2\{(p\text{-}1)+(p\text{-}3)+(p\text{-}5)+\ldots\ldots\ldots\ldots+\ (p\text{-}(p\text{-}s))\}$**]**+ $Z\times Q^2(s\text{-}1)$ + **[**$Q^2\{(s\text{-}1)+(s\text{-}2)+(s\text{-}3)+\ldots\ldots+\ (s\text{-}(s\text{-}1))\}$**]-** $[\{(p\text{-}1)+(p\text{-}3)+(p\text{-}5)+\ldots\ldots\ldots\ldots+(p\text{-}(p\text{-}s))\}]+Z\times(s\text{-}1)+$**[**$\{(s\text{-}1)+(s\text{-}2)+(s\text{-}3)+\ldots\ldots\ldots\ldots+(s\text{-}(s\text{-}1))\}$] (18)

i. The first and third series in eqn. 18 can continue unless,
The C_{rows} left to be compared with any $C_{column} >= s$
The terms with Z in eqn. 18 can exist (i.e Z will be non zero) unless,
The C_{rows} left to be compared with any $C_{merged} >= s$

ii. The second and fourth series in eqn. 18 start when,
The C_{rows} left to be compared by a C_{column} or a $C_{merged} < s$

p-1 *(no of pre-clusters-1)*	No of terms up to which the first and third Series of eqn.18 will continue	Z
s	1	0
s+1	1	1
s+2	2	1
s+3	2	2
s+4	3	2
s+5	3	3
s+6	4	3
s+7	4	4
.	.	.
.	.	.
.	.	.

Thus, $Z=[\frac{1}{2}(p\text{-}s\text{-}1)]$, as derived from the above chart.

[Taking ceiling function i.e. rounded upto next higher integer if (*p-s*) is even]

$= Q^2[(p\text{-}1)+(p\text{-}3)+(p\text{-}5)+\ldots\ldots\ldots\ldots+\ (p\text{-}(p\text{-}s))] + \frac{1}{2}(p\text{-}s\text{-}1)Q^2(s\text{-}1) + Q^2[(s\text{-}1)+(s\text{-}2)+(s\text{-}3)+\ldots\ldots(s\text{-}(s\text{-}1))]$

-

$[(p\text{-}1)+(p\text{-}3)+(p\text{-}5)+\ldots\ldots\ldots\ldots+\ (p\text{-}(p\text{-}s))] + \frac{1}{2}(p\text{-}s\text{-}1)\ (s\text{-}1) + [(s\text{-}1)+(s\text{-}2)+\ (s\text{-}3)+\ldots\ldots(s\text{-}(s\text{-}1))]$ (19)

Eqn. 19 if simplified using A.P Series, the time complexity will be $O(Q^2p^2)$.
The total number of data points (n)= the number of preclusters (p) × the size of each preclusters (Q).
$\therefore O(Q^2p^2) = O(n^2)$.

Thus, the time complexity of the second stage is O(n^2).
The space Complexity is O(p) for storing the status of 'p-1' flags in the "p-1×1" distance matrix. The space required to store the maximum distance of a cluster from the leader of another cluster for calculating ε_{max} is O(1) as there is no need to store this maximum distance for all clusters and this space can be reused for every pair of clusters.

5.3.3. The Total Time and Space Complexity of the algorithm:

The total time complexity of the algorithm including the pre-clustering and the second stage is O(pn)+ O(Q^2p^2) = O(Q^2p^2) = O(n^2) and the space complexity is O(n)+O(p)+O(1)= O(n) .

5.4. Advantages and disadvantages of our algorithm over other well-known methods:

- The pre-clustering algorithm merges data to a cluster only if it comes in threshold to the leader of a cluster. In other words, merging data to a cluster does not require the computation of its distance from all the members of that cluster. Moreover, the pre-clustering algorithm being an accelerated approach of the Leader algorithm also saves a lot of unnecessary distance computations by using triangle inequality.
- The second stage of the algorithm only computes the inter-cluster distances of the provided pre-clusters and hence the proposed algorithm saves all the intra-cluster follower to follower distance computations under all circumstances and thereby converging faster in comparison to the other well-known variants which computes the distance between all the data points to initiate clustering.
  Let us consider two cases to find the number of distance computations that can be saved by the algorithm for 'n' number of data points.
  There are 'n-2' pre-clusters with 'n-3' being singletons and the remaining one of size three with two followers.
  There are 'm' pre-clusters with 'm-1' being singletons and the remaining one has 'n-m+1'members or 'n-m' followers.

For case (i), the number of intra-cluster (follower to follower) distance computations is 1. This is because the largest pre-cluster has only two followers and all other pre-clusters are singletons. However, for Case (ii), $\frac{(n-m)(n-m-1)}{2}$ follower to follower distance computations is saved. Thus depending on the type of data and also on the threshold, the minimum number of distance computations that can be saved ranges between 1 to $\frac{(n-m)(n-m-1)}{2}$ .This is even in the worst case when $\boldsymbol{\varepsilon}'_{\min} < \boldsymbol{\varepsilon}_{\max}$. It should also be noted that as the threshold increases the number of pre-clusters decreases, i.e., size of preclusters increases, and hence more and more follower-to-follower distance computations can be saved as within-cluster followers are increasing.

Moreover, if $\boldsymbol{\varepsilon}'_{\min} >= \boldsymbol{\varepsilon}_{\max}$, the number of distance computations that can be saved at the second stage is $dist(m, f_j)\ \forall\ f_j$ belonging to C_j for which the above condition is satisfied, where '*m*' is any member of Cluster C_i as discussed in section 5.2. Thus, although the worst-case time complexity of the algorithm is still $O(n^2)$, the convergence time can be decreased by a large extent as shown experimentally in section 5.5.

- The algorithm is beneficial over all other well-known methods where the entire database is required in advance to initiate the clustering process and is also suitable for partial clustering of streaming data during collection because of the fast and incremental preclustering stage.
- The algorithm is suitable for multivariate datasets and does not employ parallelism techniques to enhance speed up and hence can be used in cases where the limited amount of GPU memory poses a serious drawback like the algorithm proposed by Rehn *et al.* in [44].
- In the popular Clink algorithm [42] proposed by Defays, the quality of the clusters is exactly not identical to the original Complete Linkage algorithm but satisfies the properties of the CL clusters. In our proposed algorithm too, the final quality of the cluster may not be exactly similar to the original Complete Linkage algorithm because of the incremental preclustering stage which depends on the ordering of the data points. Hence if an incoming data or cluster comes in threshold to more than one cluster, it will be merged to the cluster compared first. Nevertheless, the final clusters always satisfy the user-defined threshold and the criterion of the complete Linkage clustering i.e. the maximum inter and intra-cluster distance above and within the threshold respectively.

5.5. Experimental Results:

1. The clustering output and the convergence time are compared with the distance matrix methods for standard and large real datasets and are presented in TABLE 3 and TABLE 4 respectively. The convergence time of the algorithms is calculated by taking an average of 1000 computations for each case to increase the accuracy.
2. Fig-10 shows the % decrease in the number of distance computations with increasing data size in comparison to the distance matrix methods. It shows that the improvement is more prominent as the size of data increases for a given threshold and hence our method is suitable for large datasets. This is because for a large dataset the size of the matrix increases irrespective of the threshold value which results in a large number of distance computations. On the contrary, the proposed algorithm if gets a proper threshold can eliminate a lot of redundant distance computations by applying the triangle inequality resulting in speeding up the algorithm.

Table 3: Comparative Result for standard datasets:

Dataset	Threshold	No of distance Computations (approx. In millions)		Time (approx. in mins)	
		Distance matrix computation only	Proposed Method	Distance matrix computation only	Proposed Method
Pendigits Testing	30	6.1	1.30	10.8	8.25
	40		1.09		7.17
	50		0.81		6.37
	60		0.56		4.28
Pendigits Training	30	28	5.29	52.2	36.31
	40		4.45		30.16
	50		3.06		18.79
	60		2.03		8.77

Table 4: Comparative Result for large real datasets:

Dataset	Threshold	No of distance Computations (approx. In millions)		Time (approx. in mins)	
		Distance matrix computation only	Proposed Method	Distance matrix computation only	Proposed Method
Letter	5	199.99	26.56	372.4	143.02
	10		2.92		33.54
	15		1.55		12.49
	20		0.92		7.90
Shuttle	10	1681	7.67	3172.56	108.65
	15		2.77		91.32

	20		1.36		87.69
	25		0.84		71.04

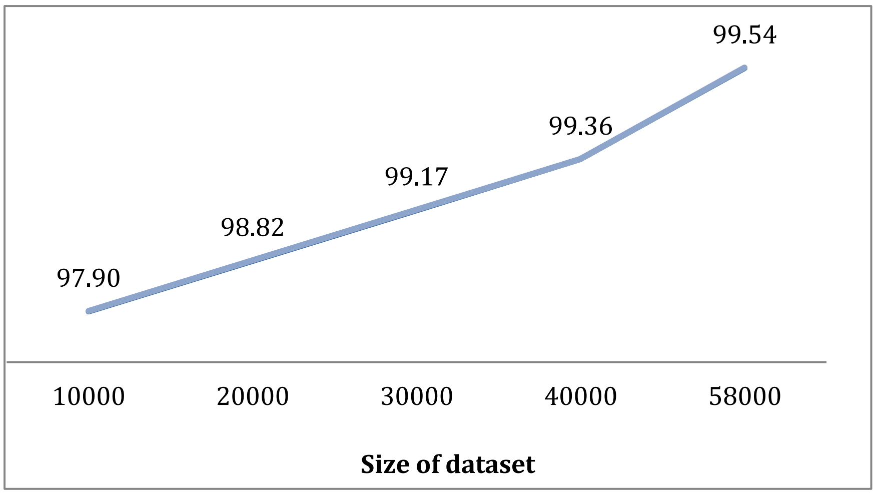


Figure 10: % decrease in the number of distance computations for shuttle dataset with threshold= 10

5.6. Conclusion

The paper presents a Complete Linkage Clustering algorithm that supports the fast clustering of large datasets using triangle inequality. The algorithm does not necessarily require computing the distance between all the data points to take a clustering decision, unlike the traditional variants where distance matrix computation with all the data points is mandatory to start a clustering action. This method because of the incremental and fast preclustering stage is also suitable for partial clustering of streaming data along with the collection which is an added advantage of the algorithm over the existing variants. The clusters produced at the output of the final merging stage always satisfy the properties of the Complete Linkage clustering and the user-defined threshold.

Chapter-6

A complete Linkage algorithm for clustering dynamic datasets

Apart from dealing with the data avalanche, researchers face problems in processing databases that are often updated. The complete Linkage algorithm being well known for deriving small and compact clusters is unsuitable for dynamic clustering as the traditional methods require the entire dataset in advance to take a clustering decision. This is because, for every addition of data, the entire dataset will be processed again and again for taking a clustering decision. The above two complete Linkage methods proposed in chapter 4 and 5 are unsuitable for handling dynamic datasets, so our paper presents a two-staged partially incremental Complete Linkage Clustering algorithm that is suitable for clustering both large and dynamic databases. Experiments are conducted with various standard datasets and the result confirms its effectiveness for both static and dynamic databases.

6.1. The Proposed algorithm

This algorithm uses the same preclustering algorithm as used in Chapter 5. The extra step used here is to store the maximum distance of the followers from the corresponding leader of a pre-cluster. Banerjee *et al*. [65] proposed a final merging stage after the preclustering stage that uses the idea of the SLINK [32] and CLINK [42] algorithms. It uses an $m \times m$ (m being the number of preclusters) distance matrix that calculates the distance between all the clusters and two arrays namely A_v and A_d, where the A_v array stores the nearest neighbour of every cluster and the A_d array stores the corresponding distances. At every iteration the A_d array is scanned, to find the minimum distance (clusters which are closest to each other) and the corresponding clusters are merged. The distance matrix and the arrays are then updated for this newly formed cluster. This process is continued unless and until all the clusters are merged or no more merging is possible, whichever is earlier. In this work, we have added an extra operation of storing the record of all the pre-clusters that are merged into a cluster. This record is used to speed up clustering dynamic databases, details of which are explained in section 6.3.

6.1.1. Keeping a record of all the pre-clusters being merged into a single cluster

This step stores the information about which pre-clusters are present within a cluster. This can be done using the steps depicted in algorithm 6. Here the word ‘cluster’ refers to the group of two or more pre-clusters in merged form.

Algorithm 6: Recording the merging of Pre-clusters in a cluster

```
1. Search $A_d$ array to find the two closest clusters to be merged
2. If (a pre-cluster is merged with another pre-cluster)
    Initiate an array for the newly formed cluster and Store the Pre-cluster indexes within it.
   Else
     If (a pre-cluster is merged with an already formed cluster)
     Insert the index of this pre-cluster within that cluster array
       Else
        If (two clusters are merged)
        Mark the array of the higher indexed cluster as merged and store this cluster Index in the lower indexed cluster array.
3. Repeat 1, 2 unless no more merging is possible.
4. Output all the clusters whose arrays are not marked as merged.
```

It should be noted that during the process, the arrays marked as merged are not freed, hence the contents (pre-cluster indexes) of that cluster can be obtained from its corresponding array.

6.2. Time and space complexity of the algorithm:

6.2.1. The time and space Complexity of the Pre-clustering stage:

The time complexity of the Leader algorithm is O(*mn*), with '*m*' and '*n*' being the number of pre-clusters and the total number of data points respectively, the space Complexity is O(*n*) for storing all the Pre-clusters with all the Leaders along with their followers [64].

Let us consider, the average size of each Pre-cluster is '*p*'. While assigning a follower to a leader, every time its distance is compared to the distance of the follower farthest from the leader. As this checking is done for all the followers of a leader, the total time taken to identify the follower farthest from the Leader is O(*p*). For '*m*' pre-clusters the total time required is O(*mp*) = O(*n*).

Each Pre-cluster stores the maximum distance of a Leader from its followers so that the comparison can be done at every follower assignment. The space required to store this maximum distance for each pre-cluster is O(1), therefore the total space required for '*m*' pre-clusters is O(*m*).

Thus, the total time complexity of this stage is O(*mn*) + O(*n*) = O(*mn*). The total Space Complexity is O(*n*) + O(*m*) = O(*n*).

6.2.2 The time and space Complexity of the final merging stage:

Banerjee *et al*. [65] showed that the time complexity of the second stage is O(n^2). The space complexity is O(*m*) for storing the arrays and O(m^2) for

storing the distance matrix. The space required for finding the nearest neighbor of a cluster (by scanning the column of a distance matrix) and the smallest element of the A_d array is O(1).
Storing the record of the indices of the pre-clusters merged into a cluster requires O(m) space for 'm' pre-clusters. Thus, the total space complexity is O(m^2).

6.2.3. The total time and space complexity of the algorithm:

The total time complexity including the Pre-clustering stage as well as the final merging stage is O(mn) +O(n^2) is O(n^2).
The total space complexity is O(n) + O(m^2) ≈ O(n), $n > m$
≈ O(m^2), $m^2>n$

6.3. Clustering dynamic datasets:

Dynamic datasets are those datasets that are updated frequently. In this work, updating refers to the addition of data to an already available database. This section shows how without using the entire database we can cluster newly added data to an existing cluster following the user-defined threshold. Let us break the arrival time of the new data into three different cases, and show how clustering can be done in all these three cases.

Case 1: Let the data arrive before the completion of the Pre-clustering phase.
This input pattern like all other data of the dataset will be subjected to the accelerated Leader algorithm with the threshold of half the user-defined value and will be either assigned as a follower to any leader or will become a new leader.

Case 2: Let the data arrive after the completion of the Pre-clustering phase but before the completion of the second phase.

The nearest neighbor of this pattern from all the existing clusters will be calculated and a new location in A_v and A_d array will be added to store the information about its nearest neighbor and the corresponding distance. The rest of the process will remain the same as that of clustering a static database, like searching the A_d array to find the two most similar clusters within the threshold and merge them. The loop of searching the nearest neighbor of a merged cluster and of clustering the two most similar clusters will continue unless no more clusters will be in the threshold or all the clusters are merged, whichever is earlier.

Case 3: Let the data arrive after clustering the entire database.

Instead of a set of totally unlabeled data, now a set of clusters are available, hence instead of comparing the new input pattern to each data, it will be compared to the available pre-clusters present within a cluster. This is why the second stage keeps a record of all the pre-clusters merged into a cluster. A newly added input pattern is merged with a cluster only if the maximum distance of the data from each of the pre-clusters (within the cluster) lies within the threshold. This maintains the maximum intra-cluster distance of the cluster within the threshold, thereby satisfying the first property of the Complete Linkage clusters. It should be noted that after the convergence of the final merging stage, all the clusters are already above the threshold so including a new pattern to a cluster will still keep the maximum inter-cluster distance above the threshold and hence no more merging will be possible. To speed up the method by avoiding the redundant distance computations, a new pattern has been added to a cluster using the method of triangle inequality. Let l_i be a leader of a pre-cluster within a Cluster C_k and f_i be any follower to that Leader (Fig.-11).

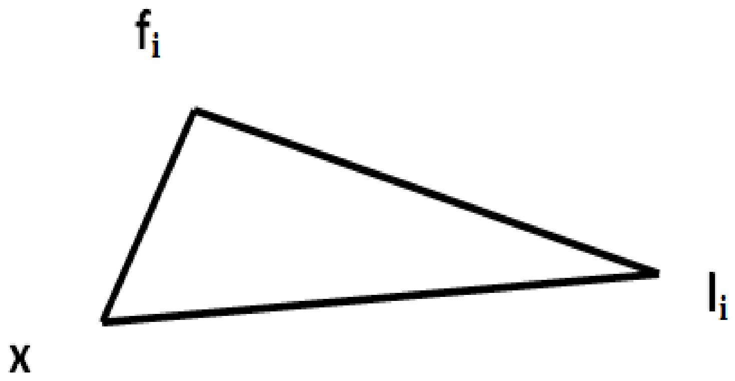


Figure 11: Computation of distance of a newly added pattern from a pre-cluster

For any incoming pattern 'x'

$\text{dist}(x, f_i) <= \text{dist}(f_i, l_i) + \text{dist}(l_i, x)$ (20)

As all the pre-clusters have been formed with a threshold of half the user-defined value and they follow the first property of the Complete Linkage algorithm i.e., the maximum intra-cluster distance is within the used threshold,

$\therefore \text{dist}(f_i, l_i) <= \zeta/2 - \varepsilon$, ε ranges from 0 to $\zeta/2$

$\varepsilon_{\min} = \zeta/2 - \max\{\text{dist}(f_i, l_i)\}$

Case-a:

Now, if the distance between the incoming pattern and a leader is within half the user-defined threshold,

$\therefore$dist (l_i, x) <= $\zeta/2-\varepsilon'$, ε' ranges from 0 to $\zeta/2$

Thus, eqn. 20 becomes,

dist (x, f_i) <= dist (f_i, l_i) +dist (l_i, x) <= ζ- (ε+ ε') <=ζ

Thus, if the distance of the incoming pattern from the leader of a pre-cluster is found within $\zeta/2$, it can be concluded that the maximum distance of the pattern from that pre-cluster is within the given threshold without comparing it with the other members of the pre-clusters.

Case-b

$\zeta/2$ < dist (l_i, x) <=ζ

Let, dist (l_i, x) =$\zeta/2$+ Q, 0 < Q <=$\zeta/2$

Then, eqn. 20 becomes,

dist (f_i, x) <= dist (f_i, l_i) + dist (l_i, x)

dist (f_i, x) <= $\zeta/2-\varepsilon+\zeta/2$+ Q

dist (f_i, x) <= ζ-(ε-Q)

It should be noted that for a particular pre-cluster and input data dist (l_i, x) is fixed i.e. 'Q' is fixed but dist (f_i, l_i) varies as a leader may have many followers.

Thus,

max{dist (f_i, x)} <= ζ-(ε_{min} -Q)

Let Z= ζ-(ε-Q) (21)

If ε >=Q, Z<= ζ

If ε<Q, Z >ζ

As, we have seen in section 6.3, ε_{min} = $\zeta/2$ – max {dist (f_i, l_i)}.

It should be noted, that the pre-clustering phase itself stores the maximum distance of every leader from its followers, hence max{dist (f_i, l_i)} will always be available for any pre-cluster.

If, max{dist (f_i, l_i)} be such that ε_{min} > Q , then by eqn. 21 , 'Z' is <= ζ for ε_{min}.

If 'Z' is <= ζ for ε_{min},

max{dist (f_i, x)} <= ζ

If using the above method, max{dist (f_i, x)} <= ζ, then check for the next pre-cluster in that cluster, else compute the maximum distance of the pattern from that pre-cluster directly.

Case c:

dist (l_i, x) >ζ
This itself implies that the maximum inter-cluster distance of a pattern from the cluster is above the threshold, hence no more checking is required, because a pattern cannot be a member of a Complete Linkage Cluster if its distance from at least a single member of that Cluster is found above the threshold.
The steps for clustering newly added data following case-3 are depicted in algorithm 7.

Algorithm 7: Checking the maximum distance of a pattern from a cluster

```
Pi= any pre-cluster belonging to a cluster Ck
li – Leader of pre-cluster Pi
fi-any follower of li
For any added pattern 'x'
Loop1: For any unchecked Pi
 If dist(x,li)<= ζ/2
  {
  Repeat Loop1 with the next unchecked Pi
  }
 Else
  {
   If dist(x,li)<= ζ
   {
    If (max{dist(fi, li)}+ dist(li, x) <= ζ)
     {
     Repeat Loop1 with the next unchecked Pi
     }
     Else
      {
      Compute the max {dist (Pi, x)} directly.
      If max{dist(Pi, x)}<= ζ
       {
       Repeat Loop1 with the next unchecked Pi
        }
        Else
         {
        Check the next cluster.
         }
        }
       }
      Else/ dist(x,$l_i$) >ζ/
       {
       Check the next cluster.
       }
      }
```

6.4. Complexity for Clustering newly added data in dynamic datasets:

The worst case appears under two conditions:

a) When ε_{min}< Q happens for a pre-cluster, the computation of the distance of the incoming point from all the followers of the pre-cluster is required.

b) If the data is found to be a member of the cluster compared at the very last or if the data can't be merged to any of the clusters.

These two conditions together lead to the computation of the distance of the incoming data with all the available clusters thereby requiring a worst-case time complexity of O(n).

The best-case arises:

a) when dist (l_i, x) <= ζ/2

or

b) ε_{min}>= Q

If any of the above two conditions become true for the very first compared pre-cluster then it leads to instant clustering leading to only O(1) computation for the clustering action.

The original algorithm already accounts for the space required for making the algorithm suitable for clustering dynamic datasets, thus the space complexity remains O(m^2) + O(n).

What if the newly added data comes in threshold with more than one cluster?

If the newly added data comes in threshold with more than one cluster, then to form more compact clusters, it's better to choose the cluster nearest to the data. For a cluster C_i, with leader l_i and followers f_i, dist (x, f_i) <= dist (f_i, l_i) +dist (l_i, x) where 'x' is the newly added data. Now, if we choose the cluster C_i for which dist (f_i, l_i)+dist (l_i, x) is lowest, then it is more probable to merge 'x' to the cluster for which dist (x, f_i) is lowest, thereby choosing a cluster for which the max intra-cluster distance after the merging of 'x' will remain low. As max{dist (f_i, l_i)}+dist (l_i, x) is always computed for a pre-cluster as described in Section 6.3, so an average of max{dist (f_i, l_i)}+dist (l_i, x) for all the pre-clusters in a cluster can be computed which provides us an idea about the average{max{dist (x, f_i)}} for that cluster. If this process is repeated for all the clusters, then 'x' can be merged with the cluster having the lowest value of {average {max {dist (x, f_i)}} if that is within the threshold (ζ). As there are total '*m*' pre-clusters available after the pre-clustering stage so this step demands some extra time of O(*m*) to locate the cluster most suitable for merging. However, the space complexity for this step will be O(1) as constant space is required to calculate the {average {max {dist (x, f_i)}} for every cluster and then to find the lowest value {average {max {dist (x, f_i)}} among all the clusters. This process should be continued with only those pre-clusters for which the dist (x, l_i) <=ζ. Although this method can't always fetch the most accurate result like the traditional CL method but are more likely to form compact clusters. The proposed algorithm already provides clusters having maximum intra-cluster and inter-cluster distance within and above the threshold thereby satisfying the condition for the CL algorithm. Hence, it is advisable to follow this part only when more compact clusters are desired.

6.6. Advantages and disadvantages of the algorithm

- The main advantage of the algorithm is the utilization of the method of triangle inequality at various stages to bring down the overall convergence time by avoiding redundant distance computations. The time complexity is O(n^2) but the use of triangle inequality may bring down the convergence time much below the traditional methods where distance matrix computation is necessary.
- The pre-clustering phase uses an accelerated leader algorithm which is a very fast incremental approach and requires scanning every data point only once. This method helps in the partial clustering of streaming data along with the collection. This was not possible with the existing variants that require all the data points at once to initiate clustering.
- The suitability of the algorithm for efficient and speedy clustering of dynamic datasets without using all the data points in every case is also an added advantage of the algorithm over the other traditional approaches and also over that proposed by Banerjee *et al.* [65].
- The only limitation of the algorithm is its dependency on the type of data used for the algorithm. This is because the triangle inequality may fail for some datasets where the conditions used for accelerating the methods both at the pre-clustering phase or while clustering the dynamic datasets may not get satisfied. This results in computing the distance between all the members of the cluster from that particular input pattern for taking a clustering decision. So, this algorithm functioning well for a dataset may not show

significant improvement for another one. So, the choice of threshold and dataset are two important criteria for the desired performance of this algorithm. It should be noted that the choice of threshold should always be application dependent.

6.7. Experimental data

To show its effectiveness on real-world datasets we conducted experiments with some datasets of TABLE 1 and the result for static datasets are provided in TABLE 5. To cluster a dynamic dataset, we have added 1000 data points (1st 1000 data points chosen from the corresponding dataset) into the already existing database after the completion of the entire clustering action to see how they are assigned to the cluster fitted for them. TABLE 6 shows the distance computations needed to cluster these 1000 data points into the already clustered database. It is to be noted that the traditional Complete Linkage algorithm does not provide any idea of effectively clustering dynamic datasets as it requires all the data points in advance to compute the distance matrix for initiating the clustering action. The convergence time of the algorithms are calculated by taking an average of 1000 computations for each case to increase the accuracy.

Table 5: Result for static standard datasets

Dataset	Threshold	No of distance Computations (approx. In millions)		Time (approx. In mins)	
		Distance matrix	Proposed Method	Distance matrix	Proposed Method
Pendigits Testing	3	6.1	3.3	10.8	9.7
	3.5		2.98		8.86
	4		2.96		8.59
Pendigits Training	3	28	11.98	52.2	49.58
	3.5		11.72		48.65
	4		9.98		46.51
Letter	3	199.99	88.20	372.4	201.21
	3.5		82.89		198.87
	4		81.78		195.67
	4.5		78.89		183.32
	5		76.92		179.32

Table 6: Result for dynamic datasets

Dataset	Threshold	No of distance Computations (approx. In millions)	
		Distance matrix	Proposed Method
Pendigits Testing (3498+ 1000 added dynamically)	3	10.11	3.12
	3.5		2.90
	4		2.88
Pendigits Training (7494+1000 added dynamically)	3	36.06	6.99
	3.5		6.87
	4		6.76
Letter (20000+1000 added dynamically)	3	220.48	19.91
	3.5		18.89
	4		18.87
	4.5		17.89

6.8. Conclusion

The paper presents a Complete Linkage Clustering algorithm that supports the fast clustering of large and dynamic databases. It uses triangle inequality to speed up the clustering method and allows the clustering of the new incoming point at any arrival stage. The clusters produced at the output of the final merging stage always satisfy the property of the Complete Linkage clusters and the user-defined threshold. The experimental result has verified that the algorithm is also suitable for large static datasets as it removes the need of computing the distance between all the data points completely unlike most of the other variants. The algorithm is suitable for the case where parallelism for speed up cannot be achieved up to a high extent due to the limited amount of GPU memory. The most well-known prevalent methods are not suitable for dynamic datasets and hence the proposed algorithm proves itself superior to the other variants. The major disadvantage of the algorithm is the dependency of the final clusters on the ordering of the incoming points because of the incremental preclustering stage, although the properties of the Complete Linkage algorithm will always be satisfied.

Chapter-7

PLEADER: A Fast and Area Efficient Hardware Implementation of Leader Algorithm

A leader algorithm is a clustering algorithm generally used as a preclustering step in various accelerated techniques of clustering algorithms like K means [66], DBSCAN [67], Average Linkage [64], and Single Linkage [38]. The algorithm while clustering large datasets can be computationally expensive because of its high convergence time. In this paper a fast and area-efficient hardware implementation of the leader algorithm is proposed which utilizes an efficient parallelism technique to speed up the algorithm without much resource consumption. It uses a master-slave architecture model where the salve processes after comparing data with the existing clusters send the result to the master process which then updates the clusters. The algorithm is capable of automatically choosing the required number of slave processes and instead of increasing its number reutilizes them repeatedly making the process an area-efficient technique. Experiments are conducted with various datasets and the result confirms that the proposed algorithm outperforms the original method in terms of speed.

7.1. PLEADER-Parallel Leader Algorithm

The proposed algorithm is a parallel version of the leader algorithm and is called PLEADER. This method uses a divide and conquer rule where it divides a single clustering process into multiple steps and applies the parallelism technique in each step to speed up the process. It uses a master-slave architecture model where the slave processes compare data to the existing leaders and send the result to the master process which then takes a clustering decision. Consider that there are 'N' slave processes and one master process. Initially, the 1st data of the dataset will be assigned as the first leader. In the next stage, the 2nd data of the dataset will be compared with this 1st leader. If the threshold criterion is satisfied, then both of them will be merged into the same cluster else will be split into two different clusters. This comparison is done only by the 1st slave process. Now if comparing 1st and 2nd data gives a new leader then the 2nd slave process gets active along with the 1st process. Now the 3rd data is compared with the 1st and 2nd leaders by the 1st and the 2nd slave processes respectively. This data will be either assigned as a follower to any of these leaders if the threshold criterion is satisfied for that leader else will be treated as a newly found leader. So, the next step is to compare the fourth data with all the existing leaders. Now if we have three leaders then three slave processes will be active to do the simultaneous comparisons. But it is not possible to have 'Z' number of slave-processes for 'Z' number of clusters for a very high value of 'Z' because of resource limitation. So, depending on the resource availability of the system we have to fix the number of slave processes.

So, when the number of clusters exceeds the number of slave processes, then the slave processes must repeat them unless and until a clustering decision is taken by the master process. Say, if, at any stage of clustering, we have four clusters and three slave processes and then the three slave processes firstly compare the data with the first three out of four leaders. If the data lies within a threshold distance of any of these three leaders, then the data is assigned as a follower to that leader. At the next stage, the comparison again starts with the next data of the dataset. In case, the data is not found in the threshold to any of these three leaders, the 1st slave process repeats itself once again to perform the comparison with the 4th leader. During this process, the other slave processes will remain inactive. If at any stage of clustering we have 'N' slave processes and 'Z' clusters, and 'Z' is greater than 'N' then firstly 'N' slave processes will get active to perform the comparison with the 'N' clusters. If the data cannot be merged with any of these 'N' clusters then at the next stage (Z-N) number of the slave- processes get active to complete the remaining comparisons provided (Z -N) is less than or equal to N. If we find (Z -N) still greater than 'N' then again 'N' process will be active to complete the comparisons with 'N' clusters out of (Z-N) number of remaining clusters. So now we are left with (Z -2N) number of clusters. If at any of these stages the data is found to lie within the threshold distance of any of these clusters then it will be assigned to that cluster. If the threshold criterion is not met even after comparing the data with all the existing clusters, then the data will be treated as a newly formed leader. After taking a clustering decision the comparison begins with the next data of the dataset. This process repeats unless and until all the data points are either assigned as followers or as leaders. Thus, here parallelism techniques are used in every iteration unless and until a clustering decision is taken.

The steps of the proposed algorithm are summarized below:

a. Activate the master process and assign the first object of the set as the first leader. Assign the number of leaders to be compared as unity.
b. Deactivate the master process and activates the required number of slave processes depending on the number of leaders to be compared and compare the next object of the set with the existing leaders.
c. Deactivate the slave processes and starts the master process. If the data comes in threshold with any of these compared leaders, then assign the data as follower to the leader and go to step ‘g’ else go to step ‘d’.
d. If no leaders are left to be compared then go to step ‘f’ else go to step ‘e’.
e. Deactivate the master process and activate the required number of slave processes depending on the number of leaders left to be compared and compare the data with the unchecked leaders. Go to step ‘c’.
f. Assign the data as a new leader. Update the number of leaders and go to step ‘g’.
g. Repeat from step ‘b’ unless and until all the objects are clustered.

Thus, here the master process, at every iteration controls the activation and deactivation of slave processes and reutilizes them again and again unless a clustering decision is reached.

7.2. Hardware implementation of the proposed algorithm:

The entire functional block of the algorithm is divided into four major Components:

a) Distance Calculation Unit
b) Comparing Unit
c) Clustering Unit
d) Control Unit

Fig.12 shows the functional block diagram of the design. Memory ‘M’ contains all the input patterns to be clustered and Memory ‘C’ contains all the output clusters, ‘C_i’ denotes each cluster block (the entire memory ‘C’ has been broken into ‘i’ blocks where each block represents a cluster, ‘i’ ranging from 1 to the maximum number of clusters that can be produced), ‘d_j’ corresponds to the data at the j^{th} location of memory ‘M’. All the signals are marked with numbers and the description of the signals have been explained with the corresponding numbers.

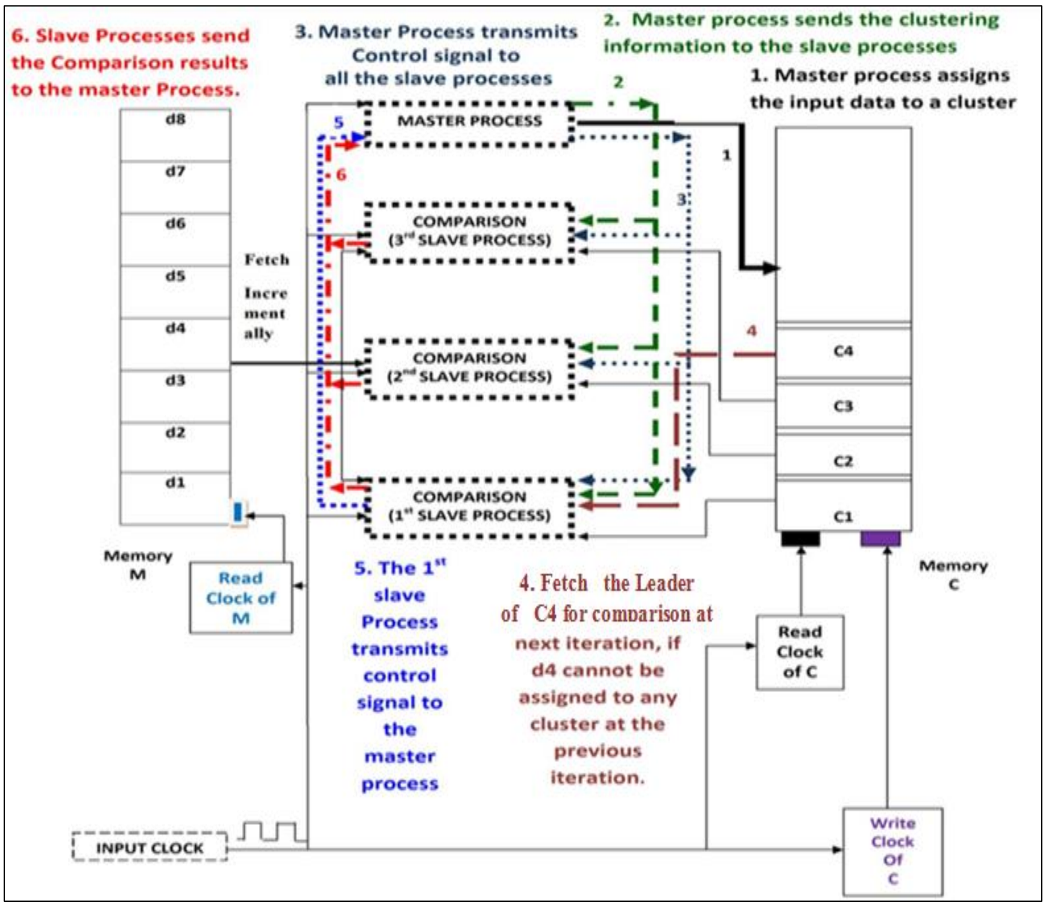

6. Slave Processes send the Comparison results to the master Process.
3. Master Process transmits Control signal to all the slave processes
2. Master process sends the clustering information to the slave processes
1. Master process assigns the input data to a cluster
MASTER PROCESS
COMPARISON (3rd SLAVE PROCESS)
COMPARISON (2nd SLAVE PROCESS)
COMPARISON (1st SLAVE PROCESS)
d8
d7
d6
d5
d4
d3
d2
d1
Fetch Incre ment ally
Memory M
Read Clock of M
C4
C3
C2
C1
Memory C
Read Clock of C
Write Clock Of C
5. The 1st slave Process transmits control signal to the master process
4. Fetch the Leader of C4 for comparison at next iteration, if d4 cannot be assigned to any cluster at the previous iteration.
INPUT CLOCK
1
2
3
4
5
6

Figure 12: Functional block diagram of the proposed algorithm Hardware Design.

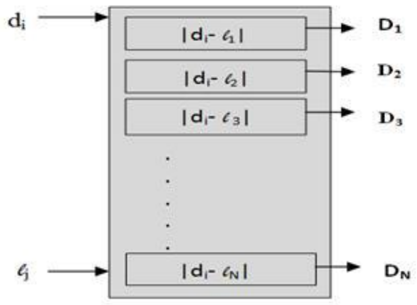


Figure 13: Distance Calculation between an input Pattern and a leader.

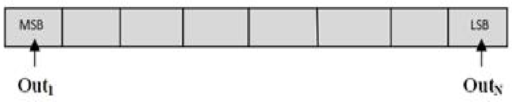


Figure 15: Vector of signal assigning priority to leaders

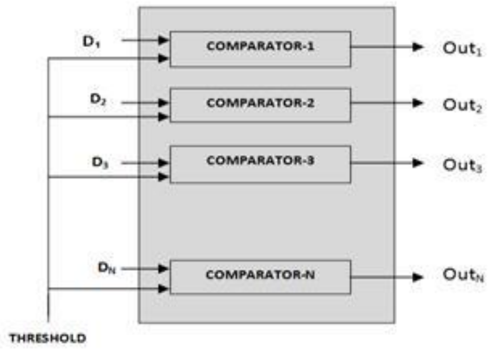


Figure 14: Comparison of the distances with threshold.

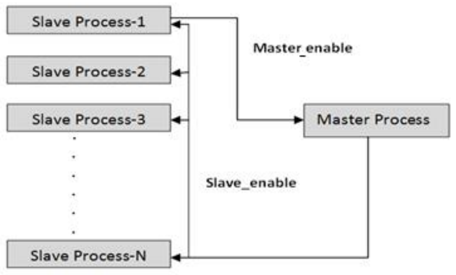


Figure 16: Control signals generated by Slave and Master Process

7.2.1 Distance Calculation Unit:

Here $l_1, l_2, l_3, \ldots l_N$ is the series of leaders according to their order of formation. In other words, 'l_1' is the first formed leader and l_N is the most newly formed leader. Choice of distance metric is a crucial task of data mining applications that highly depends on the dimensionality of the input data. Different measures give different amount of proximity to a given query point specifically at higher dimensions. Even the performance of the distance metric degrades as dimensionality increases. The behavior of L_k norm [13] shows that the choice of distance metric in high dimensionality depends on the value of k. This means that the Manhattan distance metric or L_1 norm is much more preferable than the Euclidean distance metric or L_2 norm and maximum metric or $L\infty$ norm for high dimensional data mining applications [13][14]. Implementation of the Manhattan metric is also more beneficial than the Euclidean metric in terms of resource consumption as the Euclidean distance metric involves performing square roots which is very expensive to be performed on FPGAs(Field Programmable Gate Array). Due to these advantages of the Manhattan metric over the Euclidian metric, the proposed algorithm has used the former metric for calculating the similarity between clusters. For the Manhattan metric of dimension 'd' in a plane, the Manhattan distance between the point P1 $(x_1, x_2, \ldots x_d)$ and the point P2 at $(y_1, y_2, \ldots y_d)$ is given by

$$\sum_{\mathbf{i=1}}^{\mathbf{d}} |\mathbf{X}_i - \mathbf{Y}_i|$$

This functional block as shown in Fig.13 is executed by the Slave Processes where each Slave process performs the Distance Calculation between an input pattern and the corresponding leaders simultaneously. They get a data pattern 'd_i' from the input memory and compare it with the leaders 'l_j'

fetched from the cluster memory. All the distances are then applied to the input of the Comparators which then compare the distance with the threshold.

7.2.2. Comparing Unit:

Fig.14 shows the Comparing Unit where the Comparators compare the distances provided by the Distance Calculation Unit and the user Specified threshold. The output of the comparator goes high whenever the distance exceeds the threshold else goes low. These output signals are then transferred to the Master process which then takes a Clustering decision. All the comparisons are done simultaneously by the slave processes.

7.2.3 Clustering Unit:

The clustering unit as shown in Fig.15 is performed by the Mater Process. The Master process after getting the results of the Comparison from the Comparing Unit simply makes a vector of signal with the bits at the output of the Comparing Unit. In case more than one leader are lying within the threshold distance to the input pattern, a priority rule is followed for assigning the data to a cluster according to their order of formation. So, if there are three such leaders and all are formed following the order ($1^{st} \rightarrow 2^{nd} \rightarrow 3^{rd}$) then the first leader will add the pattern as its follower. The MSB of the signal holds the bit of highest priority and is followed by the bits with decreasing order of priority as one moves towards the LSB. Here a bit of highest priority refers to the output of the comparator which compares the distance of the pattern and the first formed leader with the threshold. So, the Clustering Unit firstly checks the MSB bit of the vector and if found low, the data is assigned as the follower to that leader else the bit with the next higher priority will be checked. Thus, all the bits are checked sequentially starting from the MSB and continue unless and until a bit with a low state is reached.

7.2.4 Control unit:

The control unit shown in Fig.16 is the backbone of the entire design which controls the activation and the deactivation of the Master and the Slave processes by tracking the completion of their corresponding functions. The master and the slave processes are clocked processes. Hence at every rising edge of the clock both of them try to get activated but the control signals used in the design prevent both the master and the slave processes to work simultaneously. There are two control signals namely ‘slave_enable’ and ‘master_enable’. The slave processes get activated only after the initial cluster assignment step. The slave process works only if the slave_enable signal is high and the master_enable signal is low. Similarly, the master process works only if the slave_enable signal is low and the master_enable signal is high. At any stage of operation, all the activated slave processes work simultaneously and perform the same operations. Hence all the slave processes complete their corresponding operations at the same clock period. The slave process that gets activated for at least a single existing cluster will activate for any higher value of clusters. Therefore, the master_enable signal generated by this slave process indicates the completion of operations by all the slave processes. Hence the master_enable signal generated by this slave process can be used as a control signal for activating the master process. At the rising edge of the clock after the initial cluster assignment step, the slave processes activate depending on the number of clusters and after performing its functions makes the master_enable signal high. At the same rising edge, the master process although can't perform its operations but makes the slave_enable signal low so that at the next rising edge the slave processes can't get activated again. At the next rising edge, getting the slave_enable signal low and the master_enable signal high, the master process starts and after completing its functions makes the slave enable signal high. At the same rising edge although the slave processes can't get activated to perform their functions, makes the master enables signal low so that at the next rising edge the master process can't get activated again. It should be noted from Fig.12, that the read operation from memory ‘M’ and ‘C’ is done by the slave processes, and the write operation to memory ‘C’ is done by the Master Process. As the slave and master processes can't get activated at the same clock, hence read and write operations of the cluster memory (Memory C) is not occurring simultaneously.

7.3. Time and Space Complexity of the algorithm:

a) The first input data of the dataset is assigned as the first leader. This requires two clock pulses. The 1st clock pulse is used for the initial data read operation and the 2nd clock pulse is used for writing the data to the cluster memory.
b) At the next rising edge, the Slave-process gets activated and then compares the 2nd data in the dataset with the existing leader. At the end of the process, the master_enable signal is generated which at the next clock cycle will activate the master process.
c) The 4th clock is spent by the master process to take a clustering decision. After the end of its operations, the master process generates the slave enable signal that in the next clock activates the slave processes.
So thus, we see that starting a slave process to its restart requires two clock cycles. If after ‘X’ number of iterations (starting of a slave process to its restart), we find the number of clusters left to be compared is less than or equal to the number of slave processes, then it needs just one more iteration

to assign the data to a new cluster. Thus, in total (X+1) number of iterations are required to complete the task. So, for ‘Z’ existing clusters and ‘N’ number of slave-processes, if after ‘X’ iterations we have (Z - X N) number of clusters left to be compared, which is less than or equal to the number of slave-processes then we can find the total number of iterations for assigning a data to a new cluster.

For, $Z-XN<N$

Or, $\frac{Z}{N} < X+1$

For, $Z-XN=N$

Or, $\frac{Z}{N} = X+1$

** If (X+1) is a fraction then it will always be rounded up to the next higher integer and that will give the exact number of iterations to assign a data to a cluster. Each iteration takes two clock cycles, so 2×(X+1) number of clock cycles are required to assign data to a new cluster. Depending on the resource availability the value of ‘N’ has to be adjusted. To find the maximum number of clock pulses required to create ‘m’ leaders using these ‘N’ slave processes we have made two assumptions.

a) 1stly ‘m’ leaders are formed with no follower of each.
b) After the formation of ‘m’ leaders, the remaining ‘n-m’ number of data points will be assigned as a follower to the m^{th} leader.

Using the above two assumptions the maximum number of required clock pulses are,

$$2+2\times\left[\frac{1}{N}+\frac{2}{N}+\frac{3}{N}+\cdots+\frac{m-1}{N}\right]+2\times(n-m)\frac{m}{N} \tag{22}$$
$$=2+2\times\sum_{Z=1}^{m-1}\frac{Z}{N}+2\times(n-m)\frac{m}{N}$$

In eqn. 22, ‘2’ has been added because that includes the clocks required to assign the first data of the dataset as the initial cluster.

Here the series $\frac{1}{N}+\frac{2}{N}+\frac{3}{N}+\cdots+\frac{m-1}{N}$ $\frac{1}{N}+\frac{2}{N}+\frac{3}{N}+\cdots+\frac{m-1}{N}$ is the total number of iterations required to create ‘m’ leaders. ‘ $\frac{m}{N}$ ’ is the number of iterations required to assign each of the ‘n-m’ data points as the follower of the m^{th} leader.

Simplifying eqn. 22 gives the time complexity of the proposed algorithm as $O(\frac{mn}{N})$. Thus, the time required by the algorithm decreases with an increasing number of slave processes but increases with the increase in data elements in the set.

The space complexity of the algorithm will remain the same as $O(m)$ if only leaders are stored else $O(n)$ for storing all the patterns. After retrieving the leaders from the patterns, the followers can be assigned to the leaders at any later stage without storing the followers and thereby decreasing the space complexity to $O(m)$.

7.4. Challenges in hardware implementation

- **a.** The design utilizes LUTs (Look Up Tables) to implement the memories. In the case of large memories, many LUTs are required, which can consume a lot of resources of the FPGA (Field-programmable gate array) board, and hence in that case Block RAMs are preferable over LUTs. Block RAMs are dedicated large memories embedded in FPGA and one must check the parameters of the Block RAM from the datasheet of the target FPGA before selecting it for implementing the design.
- **b.** In this design, the slave processes are reading the leaders from the cluster memory simultaneously. It should be noted that this function requires the implementation of multiport techniques of Block RAM [68] to allow simultaneous read operations from the memory by the slave processes.
- **c.** As Manhattan Metric is supported by L_k norm for higher dimensions, our algorithm has employed the same for the design. An increase in dimensions of input data will also increase resource consumption as shown in Table 10. This is because an increase in dimensions increases the coordinates of the vectors or points in a datum which increases the load of performing addition between the corresponding coordinates while computing the distance between the two data. This is however not the drawback of the design or metric but an unavoidable problem of high dimensional distance computations.

7.5. Experimental results

The full design of the Pre-clustering algorithm has been checked and verified using Xilinx ISE design suite 14.4 and implemented in Kintex 7 (KC705) using the Xilinx Vivado platform and VHDL (VHSIC Hardware Description Language) to create the design code. The clock used for the design is a 200 MHz differential clock. Fig. 17 compares the convergence time of the pre-clustering algorithm with an increasing number of slave processes as well as data size. This is the time taken by the algorithm when all the clusters are singletons i.e., none of the objects are within the threshold distance of each other. This is the worst case that can hardly happen and therefore shows the maximum possible number of clock pulses required by the method. It is shown that increasing slave processes remarkably decreases the use of clock pulses for a clustering action, thereby making the algorithm suitable for clustering large data volume. To check the performance of the design on hardware, the algorithm has been implemented on a small One-Dimensional random dataset [63]. Table 7 and Table 8 respectively show the speedup achieved and the post-implementation resource utilization with increasing slave processes when applied on the 1D dataset. To check the performance of the algorithm on multivariate real-word datasets we conducted experiments with some datasets of Table 1 and the speed improvement is presented in detail in Table 9. To check the effect of the proposed method on the clustering result we have applied Rand Index [57] and the R.I of ‘1.0’ in all cases implies that the clustering result does not get affected by increasing slave processes. Fig.18 shows the speeding up of the algorithm with variable data size and the number of clocks saved by our algorithm is found to increase with an increase in data size. Thus, our algorithm is beneficial for big data analysis which is a serious challenge envisaged by experts in the field of data science. We have not implemented the clustering of these multivariate data points in FPGA and have only shown the speedup that can be reached by our algorithm when applied to these datasets. It is quite obvious that an increase in dimensions will increase resource utilization as shown in Table 10 and for clustering a data with a very large number of attributes an advanced version of FPGA with higher resource availability will be preferred.

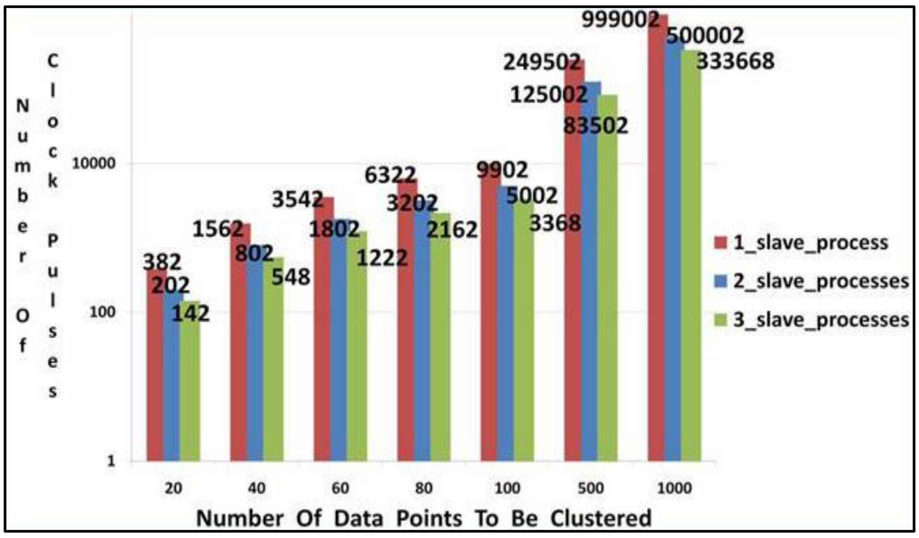


Figure 17: Variation of Convergence time of the Algorithm with varying slave processes and data size.

Table 7: Speedup achieved vs. no of slave processes for 1D dataset.

Data	Threshold	Total No of clocks required for Distance Computations			R.I
		1 Slave Process	3 Slave Process	5 Slave Process	
Random	0.5	4870	1694	1078	1.0
	2.5	1736	668	452	1.0
	4.5	1202	490	348	1.0
	6.5	988	420	314	1.0

Table 8: Post-implementation resource utilization report for increasing slave processes for ID clustering.

Resource	Available	Utilized	Utilized %	Utilized	Utilized %	Utilized	Utilized %
		One slave Process		Three slave processes		Five Slave Processes	
LUT	203800	18458	9.06	47590	23.35	72949	35.79
FLIPFLOP	407600	8354	2.05	9585	2.35	9995	2.45
IO	500	11	2.20	11	2.20	11	2.20
BUFG	32	1	3.12	1	3.12	1	3.12

Table 9: Speedup achieved vs. no of slave processes for large real datasets

Data		Threshold	No of clocks (in Millions) required for Distance Computations			R.I
			1 Slave Process	3 Slave Process	5 Slave Process	
Pen-digits	Testing	10	12.22	4.06	2.44	1.0
		20	12.12	4.04	2.42	1.0
		30	11.46	3.82	2.28	1.0
		40	9.96	3.32	1.98	1.0
	Training	10	56.14	18.72	11.22	1.0
		20	55.90	18.64	11.18	1.0
		30	53.12	17.70	10.62	1.0
		40	45.52	15.16	9.1	1.0
Letter		3	216.66	72.22	43.34	1.0
		5	125.86	41.96	25.18	1.0
		7	65.72	21.92	13.16	1.0
		9	34.54	11.52	6.92	1.0

Shuttle	5	296.80	98.96	59.40	1.0
	7	158.88	53.00	31.82	1.0
	9	92.42	30.84	18.52	1.0
	11	57.24	19.12	11.48	1.0

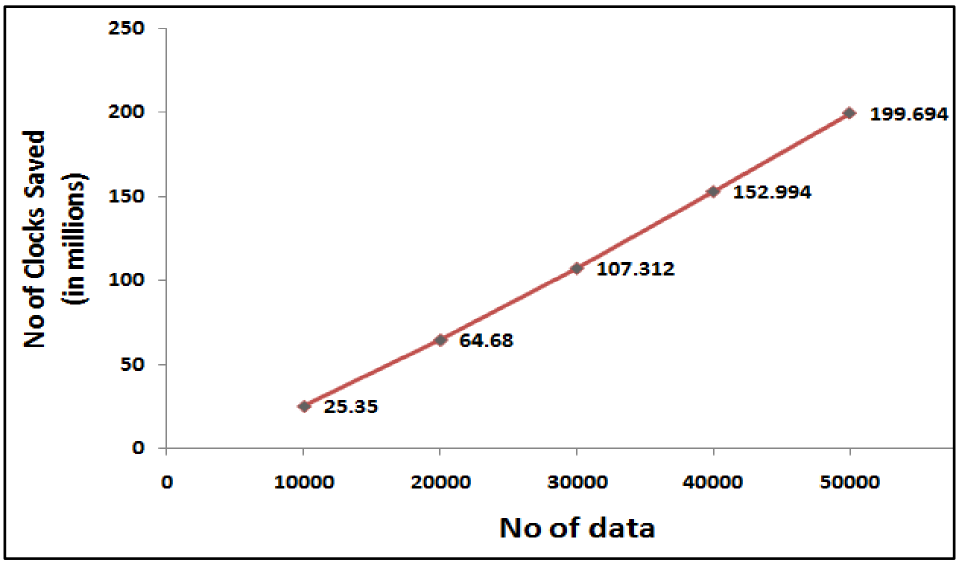


Figure18: No. of clocks saved for Shuttle dataset with increasing data size. # Slave processes = 5, Threshold=5

Table 10: Post implementation resource utilization report of Manhattan metric at one and four Dimensions.

LUT			Flip Flops		
Utilization		Availability	Utilization		Availability
1D	4D		1D	4D	
3	10	203800	4	13	407600

7.6. Conclusion

The paper presents a hardware-implemented version of the Leader algorithm called PLEADER that uses a parallelism scheme using a master-slave architecture model to speed up the convergence of the algorithm. The method instead of using a large number of slave processes reutilizes the existing slave processes on every iteration in parallel thereby optimizing the resource consumption along with providing high speed. The algorithm has been implemented on FPGA for One Dimensional dataset with a maximum of five Slave Processes and the increase in speed is found to be remarkable even without consuming half of the FPGA Resources. This work has also shown an unavoidable effect of an increase in dimensions on the resource utilization of the design. Experiments are conducted on large real multivariate datasets to show the speedup achieved by our algorithm on these data and the result is found to be highly beneficial, especially for large data sets.

Chapter-8

Accelerated Single Linkage Algorithm Using the Farthest Neighbour Principle

Single Linkage algorithm is a hierarchical clustering method that is most unsuitable for large datasets because of its high convergence time. The paper proposes an efficient accelerated technique for the algorithm for clustering univariate data with a merging threshold. It is a two-stage algorithm with the first one as an incremental pre-clustering step that uses the farthest neighbor principle to partially cluster the database by scanning it only once. The algorithm uses the Segment Addition Postulate as a major tool for accelerating the pre-clustering stage. The incremental approach makes it suitable for partial clustering of streaming data while collecting it. The Second stage merges these pre-clusters to produce the final set of Single Linkage clusters by comparing the biggest and the smallest data of each pre-cluster and thereby converging faster in comparison to those methods where all the members of the clusters are used for a clustering action. The algorithm is also suitable for fast-changing dynamic databases as it can cluster newly added data without using all the data of the database. Experiments are conducted with various datasets and the result confirms that the proposed algorithm outperforms its well-known variants.

8.1. Pre-clustering algorithm

The algorithm has been divided into two clustering methods. In the first stage, the algorithm uses an incremental pre-clustering technique that partially clusters the input data by scanning the dataset only once. The algorithm uses the farthest neighbor method which assigns input data to a cluster only when the maximum distance of the data from the cluster lies within the user-specified threshold. In other words, data can be assigned to a cluster if the member of the cluster furthest from the data lies within the threshold limit.

The algorithm assigns the first input data as the smallest object of the first cluster. Here data gets a title of the smallest and biggest based on its magnitude. The next step is to find the position of the next data concerning the first data in the magnitude scale. Here, the magnitude scale refers to a one-dimensional scale with an increase in magnitude in the upward direction. In other words, data larger in magnitude from another will always lie above the latter on the magnitude scale. The lowest and highest position of a cluster refers to the magnitude of the smallest and the biggest object of the cluster on the magnitude scale at that instant. If data is found to lie below or at the lowest position of the first cluster then its distance from the biggest object of the cluster is estimated. This provides the maximum distance of the data from the cluster. If the cluster is of size unity, then its distance is computed from the smallest object of the cluster. If the calculated distance is found to be within the threshold limit, then assign the data as the new smallest object of the cluster. The algorithm when places a data to the position of the smallest object of a cluster does not erase the previous smallest object rather reassigns it as the biggest object of the same cluster if the cluster is of size unity else assigns it a member of the same cluster with its position on magnitude scale somewhere between the magnitude of the current smallest and biggest object of the cluster. If the data lies far away from the threshold distance of the smallest object of the first cluster, then compare the data with the second cluster if available. While comparing an input data to a cluster if the data is found to lie above or at the highest position of the cluster on the magnitude scale, then the data is assigned as the biggest object of that cluster if its distance from the smallest object of the cluster is within the threshold limit. The previous object at the highest position of the cluster is then assigned to any vacant intermediate (between the highest and lowest) position of the cluster. This is because any error in assigning (based on magnitude) the members of a cluster between its lowest and highest position will not affect the maximum intra-cluster distance of that cluster unless and until the two farthest objects are not disturbed. If the position of data on the magnitude scale is found to be above the highest position of the cluster but not within the threshold distance from the lowest position then the data will be compared in the same way with the next cluster if available. Thus, the cluster always has its smallest and biggest object within the threshold distance which in turn confirms the distance between all the other members of the clusters within this predefined threshold. So, at any stage of comparison if a data is found to lie anywhere between the biggest and smallest object of the cluster on the magnitude scale, then the data will be assigned directly to that cluster without doing any kind of

distance computation. These speeds up the algorithm much more as it saves the cost of distance computation for those data which lies somewhere in between the two members of a cluster at the extreme ends of its magnitude scale. Data that cannot be assigned to any existing cluster is assigned as the smallest member of a new cluster. The process continues unless and until all the data points are clustered.
The pre-clustering algorithm follows a priority rule for comparison. A cluster that forms earlier gets a higher priority in being compared with data. So if there are three clusters and all are formed following the order (1st → 2nd → 3rd) then the first cluster will get the foremost chance of being compared with a data followed by the second and third clusters respectively. So, all the existing clusters will get priority of being compared with input data according to their order (on the time scale) of formation. But if during this process a data gets merged with an existing cluster then it will not be compared to any more clusters and the entire process will start newly with the next data point. If we consider a Pre-cluster as a chain of data then the smallest and the biggest object of this chain denote the edges of the chain. Figures 19-22 illustrate the pre-clusters on a one-dimensional (1D) magnitude scale with the magnitudes of the data increasing vertically upwards.

a) Objects A and B denote the biggest and smallest object of cluster C_1 respectively. Similarly, objects C and D denote the biggest and smallest objects of cluster C_2 respectively.
b) B and D are the first entry of clusters C_1 and C_2 respectively.
c) All other intermediate objects are within the threshold distance from the two extreme ends of their corresponding clusters on the magnitude scale.
d) All the members of C_1 are above C_2 on the magnitude scale. In other words, all the members of C_1 are of higher magnitude than those of C_2.
e) ζ is the user-defined threshold.

Case 1:

Let us consider that a data x' has arrived somewhere between the largest and the smallest member of C_1 as shown in Figure 19.

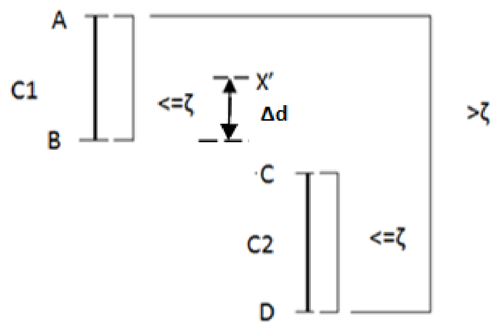


Figure 19: Pre-cluster chains with a new data b/w edges of the upper chain

There may come two subcases under case 1:

a) C_1 has formed before C_2.

Say a new data x' appears which is Δd distance above B. As the data lies between the two extremes (highest and lowest) positions of cluster C_1 so it will be merged to C_1 and there is no need to compute its distance from any other members of the cluster.

b) C_1 has formed after the formation of C_2

The data x' will be compared to C_2 before being compared to C_1.

$C\text{-}D=\zeta-\boldsymbol{\varepsilon}$, $\boldsymbol{\varepsilon}$ can range from 0 to ζ . (23)
$A\text{-}B=\zeta-\boldsymbol{\varepsilon}'$, $\boldsymbol{\varepsilon}'$ can range from 0 to ζ. (24)
$x'\text{-}B=\Delta d$, $\Delta d < \zeta$. (25)

As C_2 has formed before C_1, so B must be compared with C_2 before being compared to C_1. Figure 19 shows that the members of C_1 are above C_2 on the magnitude scale, hence the maximum distance of B from C_2 is the distance between B and the smallest object of C_2. As B has been assigned to C_1, so the maximum distance of B from C_2 must be > ζ else it would have been assigned to C_2.

$B\text{-}D >\zeta =\zeta+Q$, $Q > 0$. (26)
24+26 gives,
$A\text{-}D >\zeta=2\zeta+Q-\boldsymbol{\varepsilon}'$. (27)

24-25 gives,

$A-x'=\zeta-\varepsilon'-\Delta d$ (28)

27-28 gives,

$x'-D=2\zeta+Q-\varepsilon'-\zeta+\varepsilon'+\Delta d=\zeta+Q+\Delta d$ (29)

Eqn. 29 shows the maximum distance of x' from C_2 is more than 'ζ' distance away and hence x' cannot be merged with C_2. So, it will be next checked with C_1 and as the data is found to lie between its two extreme values so it will get merged with C_1.
So, the formation of clusters as shown in Figure 20 is not possible.

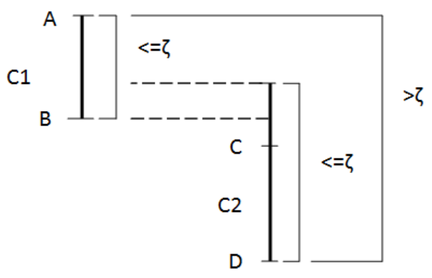


Figure 20: Pre-Clusters sharing the same space on the magnitude scale

Case 2:
Let us consider that a data x' has appeared somewhere between the largest and the smallest member of C_2 as shown in Figure 21.

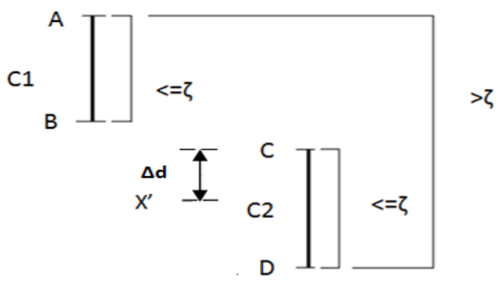


Figure 21: Pre-Cluster Chains with a new data b/w edges of the lower chain

a) C_1 has formed after the formation of C_2.

So here C_2 will get the first chance of being compared with x' before C_1. As x' lies between the two extreme positions of C_2 so it will get merged with C_2 and there is no need to compute its distance from any other members of C_2.

b) C_2 has formed after the formation of C_1.

As C_1 has formed before C_2 so x' will get compared with C_1 before C_2. Figure 21 shows that 'C' has been assigned to C_2 but not C_1 because the maximum distance of 'C' from C_1 must be greater than the threshold.

So, $A-C=\zeta+Q,\ Q>0.$ (30)

Again, $C-x'=\Delta d\ ,\ \Delta d<\zeta.$ (31)

$\therefore$ $A-x'=\zeta+Q+\Delta d.$ (32)

Eqn. '32' implies that the maximum distance of x' from C_1 is more than ζ and therefore it will not get merged to C_1. So the formation of clusters as shown in Figure 22 is not possible. Now, C_2 comes next in the priority list of being compared to x'.

As x' lies somewhere between the lowest and the highest position of C_2 so it will get merged to C_2 instantly.

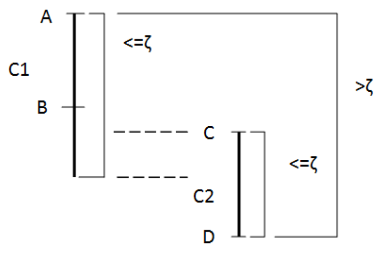


Figure *22*: Pre-Clusters sharing the same space on the magnitude scale

Thus, we see that formation of clusters sharing the same region (marked by dashed lines) on the magnitude scale as shown in Figures 20 and 22 is not possible using this pre-clustering algorithm.

Eqn. 23 and 26 give,

$$C-D=\zeta-\varepsilon$$

$$B-D=\zeta+Q$$

Therefore,

$B-C=Q+\varepsilon$

As mentioned previously, Q can get any value above '0'and 'ε' can range from 0 to ζ.

∴

$$B\text{-}C=Q \text{ for } \varepsilon=0 \tag{33}$$

$$B\text{-}C=Q+\zeta \text{ for } \varepsilon=\zeta. \tag{34}$$

If $Q < \zeta$ for $\varepsilon=0$ then $B\text{-}C < \zeta$ implying the minimum inter-cluster distance to be less than ζ. Eqn. 27 shows the maximum inter-cluster distance to be greater than ζ.

A Single Linkage algorithm merges clusters if the minimum distance between them is within the threshold limit. Using the pre-clustering algorithm, we are forming clusters whose maximum inter-cluster distance is above the threshold limit. But those clusters having minimum inter-cluster distance within the threshold limit should get merged to form final Single Linkage clusters. The second step of the algorithm serves that purpose, to provide final clusters identical to that provided by the Classical Single Linkage Clustering algorithm.

8.1.1 Steps of the Single Linkage pre-clustering (SLP) algorithm

Algorithm 8: Pseudo-code of the SLP Algorithm

```
Function dist (a, b)
{
distance = | a - b |; / Taking absolute value only/
Return distance
}
{}← initially; / $\mathcal{C}$ = a set that will contain the clusters/
Ɗ = {$d_1$, $d_2$, $d_3$, ....., $d_n$} / Ɗ = a set of input data/
i = 1, j = 1;
smallest_$C_j$ = $d_i$; /$C_j$ =$j^{th}$ cluster/
cluster_no = 1; /gives the number of clusters /
$\mathcal{C}$ = $\mathcal{C}$ U$C_j$;
i = i + 1;
While (i <= n)
{
  j = 1;
  While ($d_i$ has not been assigned to any cluster)
    {
     If (j <= cluster_no) / while all clusters have not been compared /
```

```
{
  **If** ($d_i$ <= smallest_$C_j$) then
   {
    **If** (largest_$C_j$ exists) then
     {
       Max_distance = dist (largest_$C_j$, $d_i$)
       **If** (Max_distance <= $\zeta$) then
        {
            Assign smallest_$C_j$ any intermediate location between the min
            and max value of $C_j$;
            smallest_$C_j$ = $d_i$;
            i = i + 1;   /fetch next data/
         }
        **else**
         {
              j = j + 1; /check with next cluster/
          }
       }
else
        {
          Max_distance = dist(smallest_$C_j$, $d_i$)
          **if** (Max_distance <= $\zeta$)
            {
            largest_$C_j$ = smallest_$C_j$;
            smallest_$C_j$ = $d_i$;
            i = i + 1;
            }
         **else**
           {
            j = j + 1;
           }
        }
      }
        **else**
           {
            **if** (largest_$C_j$ exists)
              {
               **if**($d_i$ >= largest_$C_j$)
                 {
                 Max_distance = dist($d_i$, smallest_$C_j$)
                 **if** (Max_distance <= $\zeta$)
                   {
                      assign largest_$C_j$ any intermediate location between the
smallest and largest value of $C_j$;
                     largest_$C_j$ = $d_i$;
                      i = i + 1;
                     }
                   else
                     {
                      j = j + 1;
                     }
                   }
                  else
                   {
                    $C_j$ = $C_j U d_i$;
                    i = i + 1;
                    j = cluster_no + 1; /to stop checking with next cluster/
                   }
                  }
                else
                  {
               Max_distance = dist ($d_i$, smallest_$C_j$)
               if (Max_distance <= $\zeta$)
                {
                 largest_$C_j$ = $d_i$;
                 i = i + 1;
                }
               else
                {
                  j = j + 1;
                }
               }
              }
            }
            else
              {
              smallest_$C_j$ = $d_i$;
              cluster_no = cluster_no + 1;
              $\mathcal{C}$ = $\mathcal{C} U C_j$;
              i = i + 1;
              }
           }
          }
```

8.1.2 Accelerated Single Linkage pre-clustering (ASLP) algorithm

The proposed pre-clustering algorithm can be computationally expensive while dealing with large datasets. The distance computation part of data from clusters is the most computationally expensive part of the algorithm, hence to reduce the convergence time of the algorithm the distance computation part has been further modified as follows.

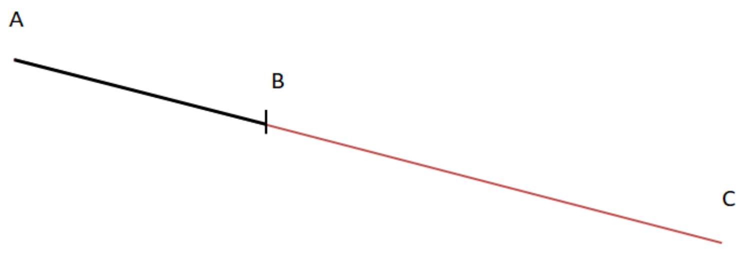


Figure *23*: Illustration of Segment addition postulate

If we are given two data points on a line segment A and C then a data B lies on the line segment AC only if the data points satisfy the equation AB+BC=AC. This is the well-known Segment Addition Postulate (SAP) and is the main principle behind the ASLP algorithm which has reduced the necessity of calculating a lot of distances without affecting the final output. Instead of calculating the distance between A and C, uses previously computed distances AB and BC to find the distance between A and C and thereby converging earlier but without hampering the cluster quality.

Algorithm 9: Pseudo-code of the proposed ASLP algorithm.

```
if ($d_i$<=smallest_$C_j$)
   {
    if (largest_$C_j$ exists)
     {
      Max_distance=max_dist_$C_j$ - (dist ($d_i$, smallest_$C_j$)); /SAP is used here/
     }
   else
     {
      Max_distance =abs (dist ($d_i$, smallest_$C_j$));
```

```
        }
      }
else
    {
      if (largest_$C_j$ exists)
      {
      if ((dist($d_i$, smallest_$C_j$))<= max_dist_$C_j$))
       {
        Max_distance=max_dist_$C_j$
        }
     else
        {
        Max_distance= (dist ($d_i$, smallest_$C_j$));
        }
      }
    else
        {
          Max_distance=dist ($d_i$, smallest_$C_j$)
        }
      }
```

The above pseudo-code shows only the distance computation part of the algorithm shown in 8.1.1. An array is maintained that stores only the longest intra-cluster distance (max_dist_C_j) for any cluster C_j.

Whenever input data is compared to a cluster C_j, instead of computing the maximum distance of the data from the cluster we can use the SAP as shown in the pseudo-code above. Every time after assigning an input data to a cluster C_j, the max_dist_C_j in the array for that C_j will be updated with the current maximum intra-cluster distance (Max_distance) of that cluster. The array is formed and updated hand in hand while forming the clusters and hence does not need any extra distance computation to calculate the longest intra-cluster distances.

8.1.3 Time and space complexity of the Accelerated pre-clustering algorithm (ASLP)

If 'm' is the number of clusters provided by the pre-clustering algorithm and 'n' is the total number of data points, then the pre-clustering algorithm will continue unless and until all the data points are scanned. To find the maximum number of distance computations for forming 'm' clusters we have made two assumptions,

a) Firstly 'm' clusters of size unity are formed.
b) All the remaining n- m data will be assigned to the mth cluster.

For each input data, the loop of comparing it to a cluster will rotate unless and until the data is assigned to any cluster or as a new cluster. If there are 'm' clusters formed by the pre-clustering algorithm then the maximum distance computations will be 'Z' where,

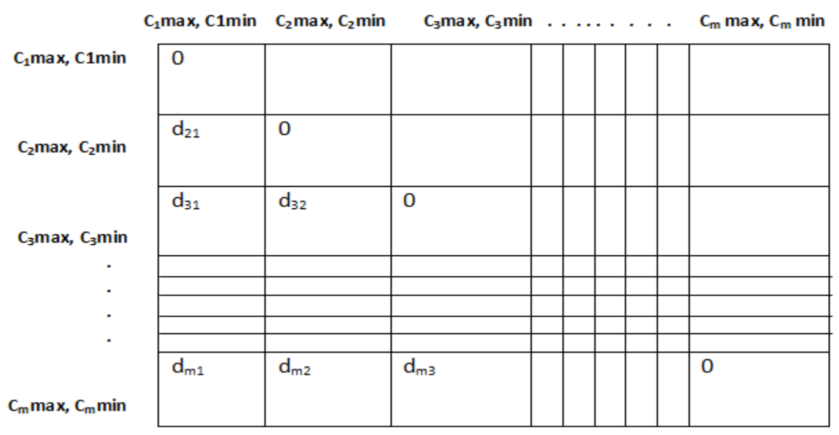
$$Z = 1 + [1+2+3+4+\ldots\ldots\ldots\ldots (m-1)] + (n-m)(m) \quad (35)$$

$$= \frac{2nm - m^2 - m + 2}{2} \quad (36)$$

The first '1' in eqn. 35 denotes the time taken to assign the first input data as a new cluster. The second term denotes the number of distance computations to form 'm' clusters where all 'm' clusters are singletons. The 3rd term denotes the number of distance computations required to assign all other remaining data points to the mth cluster.
So the time complexity of the algorithm is $O(mn)$ where 'n' is the total number of data points. The space required to store these 'm' clusters having 'n' data points and the maximum intra-cluster distances are $O(n)$ and $O(m)$ respectively.

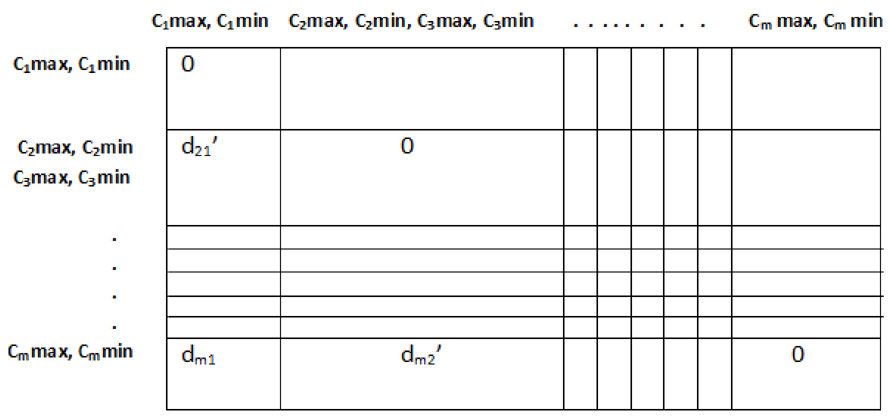
8.2. The second stage of the algorithm

The second stage of the algorithm is based on the principle of Sibson's method but with certain modifications. It simply maintains two arrays A_v and A_d storing the nearest neighbors of each cluster and the corresponding distances respectively. A distance matrix (always symmetric) as shown in Figure 24 is computed to create this A_v, A_d array where each column and row of the distance matrix is assigned to the largest and smallest member of each cluster and each element in a column gives the minimum distance of that cluster from the cluster at the corresponding row. Here C_i_max and C_i_min refer to the elements having the maximum and minimum value of the Cluster C_i respectively. We always need to find the minimum inter-cluster distance so d_{21}=min [dist (C_2_max, C_1_min), dist (C_2_min, C_1_max)] =min [dist (C_2, C_1)].



	C_1max, C1min	C_2max, C_2min	C_3max, C_3min	. . . .	C_m max, C_m min
C_1max, C1min	0				
C_2max, C_2min	d_{21}	0			
C_3max, C_3min	d_{31}	d_{32}	0		
. . . .					
$C_m$max, $C_m$min	d_{m1}	d_{m2}	d_{m3}		0

Figure 24: Distance matrix to be created to form the Av, Ad array



	C_1max, C_1min	C_2max, C_2min, C_3max, C_3min	. . . .	C_m max, C_m min
C_1max, C_1min	0			
C_2max, C_2min C_3max, C_3min	d_{21}'	0		
. . . .				
$C_m$max, $C_m$min	d_{m1}	d_{m2}'		0

Figure *25*: Distance matrix created with a merged cluster

8.2.1 Steps of the algorithm:

1. An '$m \times m$' distance matrix as shown in Figure 24 is computed to find the nearest neighbor of each cluster where 'm' denotes the number of clusters provided by the pre-clustering algorithm.
2. The distance matrix is scanned column-wise (except for the diagonal elements) to find the nearest neighbor of the cluster corresponding to that column and to find the distance from that neighbor. This is done to create the A_v and A_d array.
3. The single Linkage algorithm merges clusters with at least two data points (one from each cluster) lying within the threshold limit. Hence, we will go on scanning the A_d array unless and until we get the two closest

neighbors lying within the threshold distance. If two such neighbors are found then merge them and go to step 4 else go to step 8.

4. After every merging, the min-max of the merged cluster is required to be adjusted. Update the column of the distance matrix corresponding to that merged cluster as shown in Figure 25 to find the minimum distance of that merged cluster from the other clusters. There is no need to update the other columns as those distances are already computed and are stored in the A_d array.
5. Update the A_v and A_d array corresponding to that merged cluster by scanning the column of the distance matrix corresponding to that merged cluster.
6. Scan the A_d array again to find the closest neighbors within the threshold limit. If such neighbors are found then merge them and go to step 7 else go to step 8.
7. Repeat from step 4, unless and until the minimum distance between all the clusters surpasses the threshold.
8. Output the finally formed clusters with all of their members.

8.2.2 Algorithm used to scan a column of the distance matrix to find the nearest neighbor:

To find the nearest neighbor of a cluster we need to scan the column of the distance matrix corresponding to that cluster to find the smallest of all the distances in that column. To find the smallest of all the distances we can sort the distances of that column in ascending order and can choose the smallest one as the nearest neighbor distance. But this process requires a time complexity of $O(dlogd)$ where '*d*' denotes the total number of distances to be sorted. Hence the idea for finding the minimum element in an array [23] is used here, which requires a time complexity of $O(d)$ to find the smallest of '*d*' distances. The algorithm has a space complexity of $O(1)$ as it requires only a single location for storing the 'd_{first}' value and is updated at every iteration.

Algorithm 10: Pseudo code to find the smallest element in a column

```
Pointer → indicating each distance in a column.
d_first=the first distance of the column.
d=total number of distances in the column.
d_smallest=the smallest of all the distances in the column.
```

```
$d_{Pointer}$=the distance at location ‘Pointer’.
Pointer=2
**While** (Pointer <=d)
    {
     **if** ($d_{Pointer}$>= d_first) then
      {
        Pointer=Pointer+1;
       }
    **else**
      {
       $d_{first}$=$d_{pointer}$;
       Pointer=Pointer+1;
      }
    }
d_smallest=d_first;
```

8.2.3 Time complexity of the algorithm:

a. The distance matrix in step1 of section 8.2.1 is computed with the ‘min’ and ‘max’ values of all the clusters provided by the pre-clustering algorithm. Hence if ‘m’ clusters are provided, the distance matrix computation requires O(m^2) distance computations.
b. Scanning each column of the distance matrix to find the nearest neighbor of each cluster requires O(m) time complexity. The total time complexity to scan the entire ($m \times m$) distance matrix is thereby O(m^2).
c. The A_d array for the first time will contain ‘m’ distances corresponding to ‘m’ nearest neighbor of ‘m’ clusters. Scanning the entire A_d array will require O(m) time to find the two closest neighbors within the threshold distance.
d. Adjusting the min-max of the merged cluster requires O(1) time at every step.
   Updating and scanning the column of the distance matrix corresponding to that merged cluster requires O(m) time. This step finds the nearest neighbor of the merged cluster and is required to update the location of the A_v and A_d array corresponding to that merged cluster.
e. At every iteration, the A_v, A_d array contains one member lesser than the number of members it holds at the previous iteration. This is because, at

each iteration, two clusters are getting merged into a single cluster. Thus, for every iteration after the first one, the value of 'd' should be less than '*m*'. Scanning the A_d array to get the neighbors within threshold distance requires O(*d*) time where '*d*' is the number of distances present in the A_d array at that instant.

f. If all of the input data points are such that they can be merged into a single cluster for the given threshold, then step 'd' and 'e' will repeat for '*m*-2' times. This is because step 'a' to 'c' has already performed the first merging of the entire process. Therefore, to execute all the steps from 'a' to 'f', the algorithm requires O(m^2) time where '*m*' is the number of clusters provided by the pre-clustering algorithm.

8.2.4 Space complexity of the algorithm:

The algorithm requires maintaining two arrays each of size '*m*' that tell the nearest neighbor of each cluster and the corresponding distances respectively. The algorithm computes the distance matrix only for the first time and it does not require storing the distances as the distance between a cluster and its nearest neighbor is already present in the A_d array. It only updates a single column of the distance matrix whenever a merging is done. Thus, the algorithm requires *O*(*m*) space complexity.

8.3. Total time and space complexity of the two-stage algorithm:

The time complexity of the pre-clustering stage and the final stages are O(*mn*) and O(m^2) respectively. As '*m*' is always less than '*n*' (unless and until all the clusters are singletons), hence the total time complexity is O(*mn*).
The Pre-clustering step requires *O*(*n*) space to store all the '*m*' clusters and the maximum intra-cluster distance of each cluster. The space complexity of the Second stage is, *O*(*m*) as explained in section 8.2.4. Unless and until all the clusters are singletons '*m*' is less than '*n*', hence the total space complexity of the algorithm is *O*(*n*).

8.4. Effect of threshold on the quality of the clusters

Cluster Cohesion measures the degree of association between the members of a cluster, or in other words, it measures the compactness of clusters. Therefore, a high value of cohesion is desirable for good quality clusters [69]. To measure the cohesion, the sum of squared error (SSE) method is a very useful technique which is done by doing the sum of the square of all the distances between the objects of a cluster and the mean value of that cluster. The cohesion of the cluster increases with a decrease in the value of SSE.

$$\textbf{SSE}= \sum_{i=1}^{c}(x_i - \mu)^2$$

Here $'\mu'$ refers to the mean of all objects in a cluster, and 'c' refers to the total number of objects in a cluster.

If the cohesion of the dataset is high then the algorithm yields compact clusters. With an increase in threshold value, probability of merging distant data to a cluster increases thereby decreasing the internal quality (compactness) of the clusters. Thus, the relation between the SSE of a cluster with the user-defined threshold and the cluster's cohesion is given by,

$$\textbf{SSE} \propto \text{Threshold} \propto \frac{1}{Cohesion}$$

Figure 29 shows the variation of cluster numbers with the threshold. The histograms in Figure 27 and 28 shows the maximum distance between two data in the Gaussian datasets [61]. If the threshold reaches this maximum distance level, the pre-clustering output will also result in a single cluster accommodating all the data within it as shown in Figure 29 for the Gaussian datasets. However, this property is true for any kind of dataset irrespective of its nature. This might be the most undesirable result in most applications. Hence, the choice of threshold must be applied dependent and an improper specification of the threshold may yield improper clustering results.

It should be noted that the proposed algorithm always provides identical clusters as that by the original Single Linkage algorithm for any threshold, and the effect of threshold on the quality of clusters is not only a disadvantage of our algorithm but of all the clustering algorithms in data mining.

8.5. Effect of threshold on the convergence time of the algorithm:

Increasing the value of the threshold increases the size of the pre-clusters provided by the pre-clustering algorithm. This is because with an increasing threshold more and more input data are likely to get accommodated in a cluster. This, in turn, decreases the cluster quantity at its output as more and more data points are getting merged into the existing clusters thereby decreasing the chance of the formation of new clusters. Thus, increase in threshold decreases the value of 'm' as shown in Figure 29 which in turn decreases the overall convergence time of the algorithm due to the dependency of the time complexity of the algorithm on 'm' as shown in section 8.3.

8.6. Effect of the ordering of the input data on the cluster quality:

All incremental algorithms are affected by the ordering of the input data points and so is our proposed pre-clustering algorithm. Let us consider two sets of pre-clusters $\mathbf{\Pi}_1$= {C_1, C_2......, C_n} and $\mathbf{\Pi}_2$= {C_1', C_2'......, C_n'}. The pre-clusters follow the property of min_intercluster_dist<= ζ, but in some cases, there is a possibility of min_intercluster_dist <= ζ. So, the second stage merges two clusters if the min_intercluster_dist <= ζ to form Single Link Clusters. Any two data x and x' are neighbors if they follow the criterion of $|x-x'|<= ζ$. If $x \in C_i$ and $x' \in C_j$ and x, x' are neighbors, the second stage merges C_i and C_j as their min_intercluster_dist <= ζ. The same rule also holds for $\mathbf{\Pi}_2$ when $x \in C_i'$ and $x' \in C_j'$. The ordering of the input data points can only vary the distribution of the data in the pre-clusters but the final stage irrespective of their distribution merges two clusters if they have two neighbors, one in each cluster. Thus, even if $\mathbf{\Pi}_1 \neq \mathbf{\Pi}_2$, it follows that the final clustering result remains unaffected and satisfies the criterion of min_intracluster_dist and min_intercluster_dist to be within and greater than 'ζ' respectively.

8.7. Advantages and disadvantages of the algorithm

a. Clustering a frequently updated database:

In today's world, the size of the dataset is not only increasing but they are also dynamic. Databases containing records of financial transactions and public records of birth/death registrations are all dynamic as they get updated with time. Clustering such database is a challenge as they get frequently updated and the cluster structure changes with each inclusion and deletion of data from the database. The proposed algorithm uses an incremental pre-clustering method that does not require the entire dataset in advance and hence is suitable in those cases where data in the database is getting added incrementally. Therefore, the algorithm can partially cluster the data points along with collecting them. In case of a frequently updated database, our algorithm will just increase the size of the A_v, A_d array to include the newly added data point and will find the cluster nearest to the data and its distance from it.

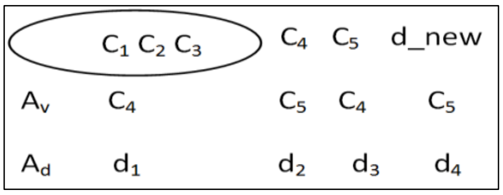


Figure 26: Arrays containing the nearest neighbors and the corresponding distances.

As mentioned previously C_1, C_2, ..., C_5 denote clusters although in the A_v array we are not including all the members of the clusters but only the objects at the edges of each cluster. If a new data d_new is added to the database then its nearest neighbor can be obtained by computing the minimum distance of d_new from all the cluster edges provided by the Pre-clustering phase or because of any previous merging. Here we see C_5 is the nearest neighbor of d_new and d_4 is the distance between them. After creating the A_v, A_d array the A_d array will be searched unless and until we find two closest clusters lying within the threshold distance from each other. If two clusters are found to be within the threshold limit, then they will be merged and the process of creating the A_v, A_d array will repeat as explained in section 8.2.1. Thus, here we are clustering a new data point to some existing clusters without using all the data points of the cluster making the process faster and easier.

b. Clustering large data volume:

The proposed algorithm does not use all of the datasets to do a clustering action and hence is more suitable than all those methods that require creating a dissimilarity matrix to initiate the action. The proposed method has brought down the time complexity of the SL algorithm much less than $O(n^2)$ as '*m*' is less than '*n'* if at least a single cluster of size greater than unity is formed by the pre-clustering phase. It thus outperforms Classical, Sibson's, and MST approaches of the SL algorithm in case of clustering large data volume.

In addition to that, the use of SAP in the pre-clustering phase speeds up the pre-clustering part to a large extent as confirmed by experimental results.

Thus, the proposed algorithm is beneficial for clustering large data volumes and especially where parallelization schemes cannot be applied to achieve higher speedups.

c. Streaming data clustering

The algorithm does not require the entire database in advance and since the pre-clustering phase is an incremental one so it is suitable for partial clustering of the incoming data alongside the collection. The final clustering, however, is done at the second stage when the pre-clustering phase is over. However, the receiving speed of the incoming data must match its processing speed to avoid data loss. Although techniques have been adopted in this algorithm to speed up the Pre-clustering phase, proper front-end electronics must be set up to synchronize the system if required.

d. Uncompromised Cluster quality

To compare the quality of clusters provided by two algorithms we have used a very popularly used method called the Rand Index. To ensure that the proposed algorithm provides identical clusters to that of the Conventional SL algorithm, we have used Rand Index on the Clustering results and the Rand Index of '1' proves that the final quality of the clusters produced by our algorithm is not affected by the speeding up process adopted by our method.

e. Speeding up is dataset independent

It has been proved that the time complexity of the algorithm is reduced to $O(mn)$ with 'n' being the number of data points and 'm' is the number of pre-clusters. This is true for any kind of dataset irrespective of its volume and type. Although the number of pre-clusters may vary, it should be noted that 'm' should always be less than 'n' if at least a single cluster of size more than unity is formed. Therefore, the convergence time of the algorithm should always be lesser than those methods where distance matrix computation is necessary at least once. Besides, the use of the Segment Addition Postulate in the pre-clustering phase is also likely to speed up the process in all cases. Hence with a good choice of threshold, the advantage of the proposed method can be obtained under all circumstances.

f. Limitation of the algorithm

The major disadvantage of the algorithm is its inapplicability for bivariate or multivariate data clustering. The concept of the highest and lowest position of a cluster used in the pre-clustering algorithm is true for univariate data analysis. This is because for multivariate data input data can't be compared to other data based on a single variable as more than one attribute is describing the pattern of that data at a time. Hence, although the algorithm proves itself highly beneficial for clustering large input volumes but can only be applied for clustering data based on a single variable.

8.8. Experimental results

To show the effectiveness of our algorithm over different datasets, we have implemented the design on two Gaussian datasets and some large real-world dataset as shown in Table 1. Figure 27 and Figure 28 exhibit the distribution of the data of the Gaussian datasets in histogram format. Table 11 shows that the ASLP algorithm outperforms the SLP method as the former converges in a shorter time doing lesser distance computations. As the time complexity of our algorithm depends highly on the pre-clustering output, hence Figure 29 shows the variation of the pre-clustering output drawn from the Gaussians and the real-world data with the threshold. Table 12 shows that our algorithm outperforms all those methods where distance matrix computation is required as the distance matrix computation itself requires computing much more distances than that required by our entire algorithm making the latter converge faster. Figure 30 shows the speeding up of the ASLP technique with variable data size and the effect is seen to get more pronounced with increasing data volume. The convergence time of the algorithms is calculated by taking an average of 1000 computations for each case to increase accuracy. To compare the cluster quality provided by the two algorithms we have used the Rand Index (R.I) on the clustering results and the R.I of ‘1.0’ in all cases of Table 11 and Table 12 implies that the clustering result is not affected by the speeding process and is identical to that provided by original Single Linkage algorithm.

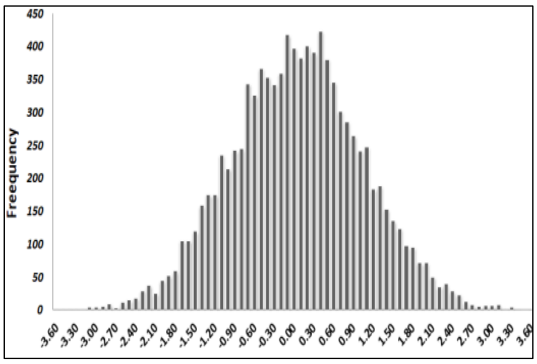


Figure 27: Histogram of the Gaussian dataset- 1

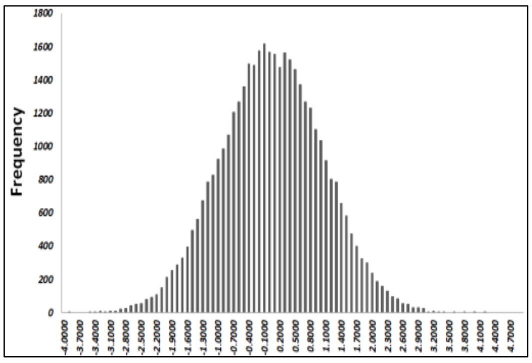


Figure 28: Histogram of Gaussian dataset- 2

Table 11: Comparative performance report of ASLP and SLP algorithm when applied on varying large datasets.

Data Set	Threshold	Distance Computations (in millions)		Time (in Sec)		R. I
		SLP	ASLP	SLP	ASLP	
Gaussian dataset-1	0.001	21.0	12.5	504.09	439.939	1.0
	0.005	7.01	3.7	157.94	132.32	1.0
	0.01	3.91	2.02	85.90	71.49	1.0
	0.05	0.85	0.43	15.56	12.08	1.0
	0.1	0.42	0.21	9.69	7.89	1.0
Gaussian dataset-2	0.001	134.36	71.88	2932.72	2606.042	1.0
	0.005	33.23	16.92	711.606	580.464	1.0
	0.01	17.29	8.72	374.105	304.546	1.0
	0.05	3.66	1.83	81.174	65.597	1.0
	0.1	1.89	0.95	42.084	33.867	1.0
Distillate flowrate	0.1	114.35	64.64	2532.529	2435.691	1.0
	0.5	36.75	19.65	800.820	665.287	1.0
	1	19.87	10.47	430.454	361.420	1.0
	2	10.23	5.3	217.985	184.70	1.0
	3	6.69	3.4	142.068	118.924	1.0

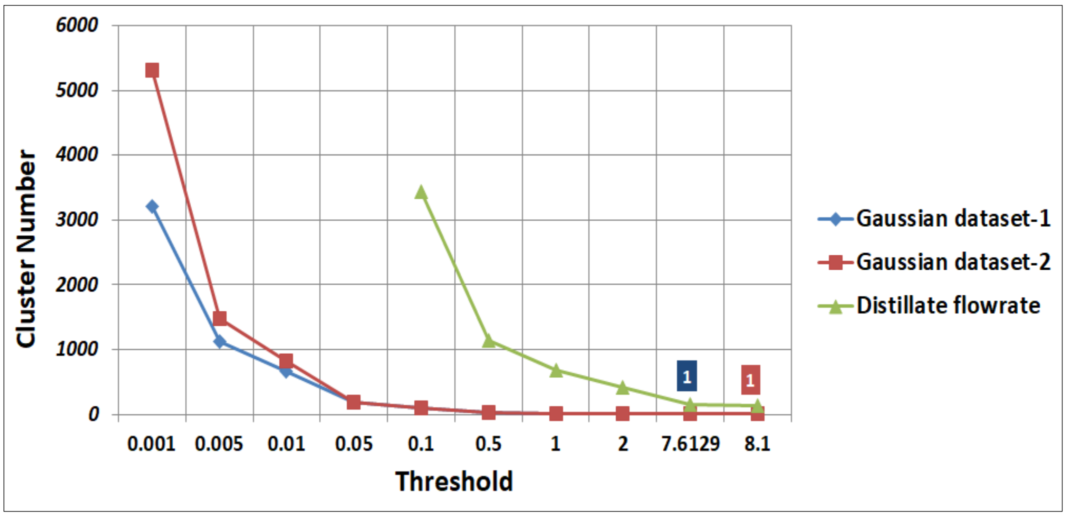


Figure. 29: Variation of Cluster number with the threshold at the ASLP output.

Table 12: Comparative performance report of the proposed algorithm and the distance matrix method when applied to datasets of varying size

Dataset	Threshold	Distance Computations (millions)		Time (mins)		R.I
		Distance Matrix	Proposed algorithm	Distance Matrix	Proposed algorithm	
Gaussian dataset -1	0.001	49.99	33.64	73.57	40.02	1.0
	0.005		6.22		7.01	1.0
	0.01		2.89		3.87	1.0
	0.05		0.49		1.378	1.0
	0.1		0.23		1.192	1.0
Gaussian dataset -2	0.001	799.98	128.22	1177.37	127.90	1.0
	0.005		21.24		17.98	1.0
	0.01		10.05		8.82	1.0
	0.05		1.90		2.28	1.0
	0.1		0.96		1.72	1.0
Distillate flow rate	0.1	996.342	88.19	1470.94	76.35	1.0
	0.5		22.23		15.88	1.0
	1		11.41		8.40	1.0
	2		5.67		4.57	1.0
	3		3.60		3.23	1.0

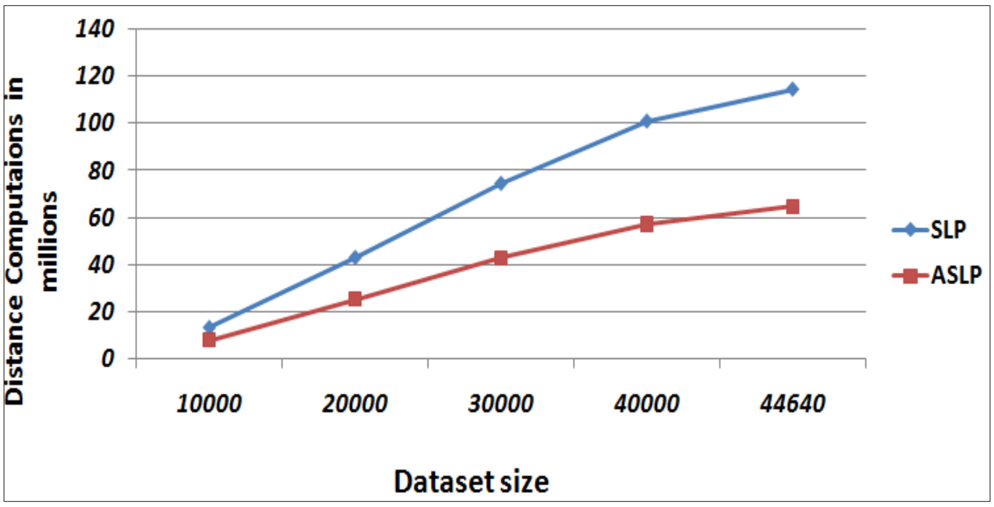


Figure 30: Number of distance Computations vs. data size for distillate flowrate data with ζ=0.1

8.9. Conclusions

The paper presents a two-stage, partially incremental Single Linkage algorithm that does not need the entire database in advance and is only applicable for univariate data analysis. The first stage of the algorithm forms a Dendrogram of pre-clusters using the farthest neighbor method and the second Stage produces nearest neighbor clusters by merging these chains of pre-clusters depending on the distance between the edges of the adjacent chains. Although the proposed algorithm has an incremental stage the final result remains independent of the ordering of the input data, unlike other incremental algorithms. The proposed algorithm without sacrificing the quality of clusters decreases the time complexity to a much lower value in comparison to its well-known variants as confirmed by experimental results. A threshold is an important criterion of the Single Linkage algorithm as an increase in threshold value decreases both the quantity and quality (compactness) of clusters by absorbing distant data to the already available clusters. Hence it is the sole responsibility of the user to choose the correct threshold to get a good quality of clusters.

Chapter-9

Accelerated Single Linkage Algorithm Using Triangle Inequality

The limitation of the Single Linkage algorithm proposed in chapter-8 is its unsuitability for multi-dimensional data. This limitation is removed in this work where not only the time complexity of the single Linkage algorithm is reduced but is also made applicable for all dimensional data. It is a two-staged algorithm with the first one is an incremental pre-clustering step that uses the triangle inequality method to eliminate unnecessary distance computations. The incremental approach makes it suitable for partial clustering of streaming data along with the collection. The second step using the property of the Single Linkage algorithm itself takes a clustering decision without comparing all the patterns. This method shows how the neighborhood between the input patterns can be used as a tool to accelerate the algorithm without hampering the cluster quality. Experiments are conducted with various standard and large real datasets and the result confirms its effectiveness for large datasets.

9.1. Accelerated SL algorithm

The proposed algorithm is a two-stage method where the first stage involves a pre-clustering algorithm followed by the second stage which produces the final clusters.

9.1.1. Pre-clustering algorithm

The algorithm has been divided into two clustering methods. In the first stage, the algorithm uses an incremental pre-clustering technique that partially clusters the input data by scanning the dataset only once. Here we have applied the well-known Leader algorithm as the pre-clustering step. To further reduce the distance computation of the Leader algorithm, its accelerated form using triangular inequality is used, details of which is already provided in section 4.1 of chapter-4. The only difference is that the threshold used in this case is the user-defined-value.
The accelerated Leader method used in this case is depicted in Algorithm 11.

Algorithm 11:

```
Let $l_i \in L$ (i=1) be the first scanned pattern, 'T' being the user-defined threshold.
1. Assign the first input pattern 'x' as $l_i$ with i=1. $L \rightarrow \{l_1\}$
2. For any input pattern $x \in D$, if d (x, $l_1$) >T, assign 'x' as a new leader $l_i$ for i=i+1.
3. Merge li with L such that $L = L \cup l_i$.
4. Store d ($l_1$, $l_i$).
5. If all 'x' are scanned go to '16' else go to '6'.
6. For any new input pattern $x \in D$, compute d (x, $l_1$).
7. If (d (x, $l_1$)>T) then go to step '9' else go to step '8'.
8. Assign 'x' as a follower to $l_1$. Go to '5'.
9. For any unchecked leader $l_i \in L$, if d ($l_i$, $l_1$)>=2d (x, $l_1$) then reverse triangle inequality says d (x, $l_i$)>=d (x, $l_1$).
10. Repeat '9' with each $li \in L$ and go to '11' if '9' holds for all $l_i \in L$.Go to '14' if '9'fails for any $l_i$.
11. Assign x as a new leader $l_i$ for i=i+1.
12. Store d (x, $l_1$).
13. $L=L \cup l_i$. Go to 5.
14. Compute d (x, $l_i$) for the corresponding $l_i$.
15. If d (x, $l_i$) <T, assign x as the follower of li and go to 5 else repeat from 9.
16. Output all the leaders with followers.
```

In the Single Linkage Algorithm, $\forall a \in A$ and $\forall b \in B$ where A and B are two distinct clusters, all A and B are merged if $d_{min}(a, b)$ <=T. So the final set of clusters has the following two properties.

(i)The minimum intra-cluster distance is less than the threshold.

(ii)The minimum inter-cluster distance is always greater than the threshold.

The pre-clustering algorithm takes a clustering decision only if the input pattern comes in threshold with a leader. It may be possible that the distance of the merged object is either greater than the threshold distance from all or any of the other elements (except the leader) of the cluster or it may be within the threshold distance from all or any of them. In either case, the minimum intra-cluster distances of every cluster remain within the threshold limit thus satisfying the first criteria of the Single Linkage clusters.

As the Leader algorithm is an incremental one and scans every data point only once, so once the entire dataset is scanned the algorithm stops doing further merging of clusters. It may be possible that the minimum inter-cluster distance between any two clusters can be still less than the threshold and thereby needs to get merged otherwise the second property of the Single Linkage clusters will not be satisfied. This function is performed by the second stage of the algorithm.

All incremental algorithms are sensitive to the ordering of the data points and so is the pre-clustering algorithm. As the second stage performs the final merging of the pre-clusters following the property of Single Linkage clusters, it does not allow the final quality of the clusters to get affected by the ordering of the input data. Hence for an input dataset, the output will remain the same, irrespective of the variation in the ordering of the input data.

9.1.2. The second stage of the algorithm.

The second half of the algorithm creates a neighborhood matrix but does not fill all the cells by computing the pair-wise distance between all the data points. It creates a 'm-1×1' matrix for 'm' pre-clusters provided by the first half of the proposed method. Each cell of the matrix is assigned with a flag which represents the neighborhood between the clusters corresponding to that particular column and row. Every time two clusters are found in the threshold, the flag of that cell is set high else low.

The operation starts with a single column corresponding to any cluster and scans the entire column to find all the neighbors of that cluster. For a cluster C_j(corresponding to the column), 'C_i'(corresponding to a row) will be a neighbor if $d_{min}(C_j, C_i)$ is within the threshold. The flag of the cell corresponding to that 'C_j' and 'C_i' is set high if they are found as neighbors.

After scanning the entire column 'C_j' is merged with any 'C_i' for which the flag is set high. The row corresponding to this 'C_i' is removed and now 'C_j' will be replaced by 'C_jUC_i'. The contents of the other cells will remain unchanged. If at every iteration, the newly merged 'C_i' is denoted by 'C_m' then the cluster representing the column will now be updated as 'C_j'= 'C_jUC_m' i.e. 'C_j' has now included a new cluster 'C_m' as its member. After merging 'C_m' with 'C_j', the row corresponding to 'C_m' will be removed from the matrix. In the next step, we have utilized the property of the SL algorithm to speed up the method. SL algorithm merges clusters only if the minimum distance between them lies within the user-specified threshold. In other words, the SL algorithm refers to two clusters as neighbors if ‘at least’ a single pair-wise distance between two clusters is found to lie within the threshold limit. To best utilize the word ‘at least’ it will check the neighborhood between ‘C_m’ and those unmerged ‘C_i’s for which the flag is still low. This is because those ‘C_i’s for which the flag is already high are already considered a neighbor to ‘C_j’ and will get merged to it at any later stage. So checking the neighborhood of those ‘C_i’s from ‘C_m’(which is itself a part of ‘C_j’) again will lead to unnecessary distance computations. The

flags of all the cells which is found as a neighbor to ‘C_m’ will go high. After checking the neighborhood of all the unmerged ‘C_i’s (with flag low) from ‘C_m’, ’C_j’ will be merged to any of the ‘C_i’ of that column for which the flag is high. The newly merged ‘C_i’ will be then assigned as a ‘C_m’. The row corresponding to the ‘C_m’ will be removed.

This scanning process for a particular ‘C_m’ will stop after checking its neighborhood from all the ‘C_i’s with flag low. After the end of the scanning process for a particular ‘C_m’, the process repeats with a new C_i (with flag high) as C_m. This assignment of C_m to a C_j will continue unless and until no more ‘C_m’s can be assigned to that C_j. When such a condition is reached following three operations are performed:

- The C_j with all its merged ‘C_m’s will be considered as a fully grown cluster.
- If for a column only one ‘C_i’ is left with its flag low then that will also be identified as a fully grown cluster and the corresponding row will be removed.
- If the flag of more than one ‘C_i’s are still low, then the entire process will be repeated from the beginning with any of these ‘C_i’s as a new ‘C_j’.

The algorithm will converge when all the rows are removed.

This entire clustering principle used in the second stage is depicted in algorithm 12.

Algorithm 12

```
$C_j$=$P_i$ for i=1, /P1 be the first pre-cluster/
$C^*$= {$C_j$} /C* be the final set of clusters/
$C_i$=$P_i$ for all ‘i’s for which $C_i \neq C_j$
$C_m$=$C_j$
j=1
k=j+1
no_of_$C_i$_with_flag_high=0;
While (all rows are not removed)
```

```
        {
While (all $C_i$s with flag low are not compared with current $C_m$)
          {
            If dist ($C_m$, $C_i$) <= T then
              {
                Set flag _ $C_i$ high
                no_of_$C_i$_with_flag_high = no_of_$C_i$_with_flag_high + 1;
              }
            else
              {
                Set flag _ $C_i$ low
              }
            }
          If (no_of_$C_i$_with_flag_high > 0)
            {
            for (any $C_i$ with flag high) do
              {
                $C_j = C_jUC_i$
                $C_m = C_i$
                no_of_$C_i$_with_flag_high = no_of_$C_i$_with_flag_high - 1;
                Remove $C_m$ row from matrix
              }
            }
        else
            {
            for (any $C_i$ with flag low) do
              {
                j = k
                k = k + 1
              Assign $C_j = C_i$
              $C_m = C_j$
              $C^* = C^*UC_j$
              Remove $C_m$ row from matrix
              }
            }
          }
Output C *
```

9.2. Time and space complexity of the proposed method

9.2.1. Calculation of time and space complexity of accelerating Leader algorithm

With $'m'$ and $'n'$ being the size of the leader set and the input data respectively, the time complexity of the accelerated Leader algorithm is calculated as follows.

$$=[(1+2+3+4+\cdots(m-1)]+[(n-m)(m)] \quad (37)$$

$$=\left[\left(\frac{m-1}{2}(2+(m-1-1))\right)\right]+[(nm-m^2)] \quad (38)$$

$$=\frac{(m^2-m\,)}{2}+(nm - m^2) \quad (39)$$

$$=\frac{(m^2-m\,)+(2nm-2m^2)}{2} \quad (40)$$

$$=\frac{2nm-m^2-m}{2} \quad (41)$$

The first expression within [ ] in eqn. 37 gives the time required for creating 'm' leaders and the second one provides the time required for assigning 'n' data points as followers of $'m^{th}'$ leader. Here 'n-m' points have been assigned to the mth leader to increase the number of distance computations to get the exact time Complexity in the worst case. So, the time complexity of the accelerating Leader algorithm is $O(mn)$ as shown by eqn. 41. The clusters provided by the pre-clustering algorithm require $O(n)$ space and the distance of the leaders from the first generated leader occupies $O(m-1)$ space which results in a total space complexity of $O(n)$.

9.2.2. Time complexity of the second stage

To calculate the time complexity of the second stage of the algorithm, let us make three assumptions:

a. All the pre-clusters have an average number of members saying 'P'. This assumption is made to make the calculation less complex.
b. At every iteration, whenever a column is scanned by a C_i (with flag low), 'f' neighbors can be obtained.
c. At every iteration, whenever a column is scanned by a C_i (with flag high), no neighbors can be obtained.

Assumption ‘c’ helps in getting the maximum possible number of distance computations that our algorithm can undergo. This is because if more and more neighbors are found on a single scan, a lesser number of distance computations will be done at the next iterations.
For ‘m’ pre-clusters there are ‘m-1’ rows with the mth cluster being assigned as ‘C_j’.So to scan the column following the above assumptions, the number of distance computations for the first time scanning will be P [m-1] P. This is because the comparison of ‘C_j’ with ‘m-1’ clusters require the pair-wise distance computations between all the points of ‘C_j’ with that of the ‘m-1’ clusters in the worst case. If ‘f’ neighbors are reached, the next iteration requires distance computations between all the points of ‘C_m’ with that of the ‘m-1-f’ clusters. All the neighbors will perform the scanning resulting in f × (m-1-f) computations. After completing scanning with all ‘C_i’s with flag high, a new C_i with flag low will begin scanning the remaining (m-2-f) ‘C_i’s with flag low. This process will continue unless and until all rows are removed. The second stage avoids the leader-to-leader distance computations between a ‘C_j’ and all ‘C_i’s as the leader-to-leader computations are already done at the pre-clustering stage.
The maximum number of distance computations required for comparing the pre-clusters following the above conditions is,

$$= \mathbf{[}\{P\,(m-1)\,P\} + f\,\{P\,(m-1-f)\,P\} + \{P\,(m-2-f)\,P\} + f\,\{P\,(m-2-2f)\,P\} + \{P\,(m-3-2f)\,P\} \ldots\ldots + \{P\,(m-\eta-(\eta-1)\,f)\,P\} + f\,\{P\,(m-\eta-\eta f)\,P\}\mathbf{]} - \mathbf{[}\{(m-1)\} + f\,\{(m-1-f)\} + \{(m-2-f)\} + f\{(m-2-2f)\} + \{(m-3-2f)\} \ldots\ldots + \{(m-\eta-(\eta-1)f)\} + f\{(m-\eta-\eta f)\}\mathbf{]} \tag{42}$$

$$= \mathbf{[}\{P^2(m-1)\} + \{P^2(m-2-f)\} + \{P^2(m-3-2f)\} \ldots\ldots\ldots\ldots + \{P^2\,(m-\eta-(\eta-1)f)\}\mathbf{]} + \mathbf{[}f\,\{P^2\,(m-1-f)\} + f\,\{P^2\,(m-2-2f)\} + f\,\{P^2(m-3-3f)\} + \ldots\ldots\ldots + f\,\{P^2\,(m-\eta-\eta f)\}\mathbf{]} - \mathbf{[}\{(m-1)\} + \{(m-2-f)\} + \{(m-3-2f)\} \ldots\ldots\ldots\ldots + \{(m-\eta-(\eta-1)f)\}\mathbf{]} - \mathbf{[}f\{(m-1-f)\} + f\,\{(m-2-2f)\} + f\,\{(m-3-3f)\} + \ldots\ldots\ldots\ldots f\,\{\,(m-\eta-\eta f)\,\}\mathbf{]} \tag{43}$$

In the worst case, the process will run for ‘η’ times identification of ‘f’ set of neighbors after which all rows are removed. In other words, after getting ‘f’ neighbors ηth time, no more C_i with flag low will be left for being scanned.

m-η-ηf = 0.

$\frac{m}{1+f}$ = η

Applying A.P. series, and replacing 'η' in eqn. 43,

$$=\frac{P^2m^2}{2} - \frac{P^2m}{2}\frac{(1+f^2)}{1+f} - \frac{m^2}{2} + \frac{m}{2}\frac{(1+f^2)}{1+f} \qquad (44)$$

$$=\frac{n^2}{2} - \frac{P^2m}{2}\frac{(1+f^2)}{1+f} - \frac{m^2}{2} + \frac{m}{2}\frac{(1+f^2)}{1+f} \qquad (45)$$

The last two series in eqn. 43 give the number of distance computations saved by avoiding the leader-to-leader distance computation of a 'C_j' or 'C_m' from all the 'C_i's with flag low. As more and more neighbors (f) are obtained on a single scan, the lesser will be the number of distances computed by the algorithm.

Thus, although the time complexity is O(n^2) as shown by eqn. 45, the number of distance computations is highly dependent on 'f' which in turn depends on the data type and the user-defined threshold. The higher the proximity of the patterns in a dataset, the more will be the chance of getting neighbors on a single scan. The chance also increases with an increasing threshold. Thus, by utilizing the neighborhood of the input patterns, the proposed algorithm has reduced the redundant distance computations required in traditional variants of SL. Moreover, to get a neighbor, it may not be required in practice to compute the pair-wise distances between all the data points of two clusters but can be stopped whenever a neighbor is identified. So, for any data and a suitable threshold where the SL algorithm can provide good clusters, our algorithm will be proved as a beneficial one.

Its usefulness has been verified experimentally in 'Section 9.4' after being applied to various real-world datasets.

9.2.3 Total time and space complexity of the algorithm

The total time complexity of the algorithm is given by O(mn+n^2) = O(n^2). However, the space complexity remains O(n) to store 'n' data points of all the clusters and the status of the 'm-1' flags of the neighborhood matrix.

9.3. Advantages of the algorithm over other well-known SL algorithms

- The pre-clustering algorithm merges data to a cluster only if it comes in threshold to the leader of a cluster. In other words, merging data to a cluster does not require the computation of its distance from all the members of that cluster. It should also be noted that the pre-clustering algorithm is itself an accelerated approach to the Leader algorithm and also saves a lot of unnecessary distance computations between an input pattern and all the leaders.
- The second stage of the algorithm only computes the inter-cluster distances of the provided pre-clusters and hence the proposed algorithm saves all the intra-cluster follower to follower distance computations under all circumstances and thereby converges faster in comparison to the other well-known variants.

Let us consider two cases to find the number of distance computations that can be saved by the algorithm for the 'n' number of data points.

- There are 'n-2' pre-clusters with 'n-3' being singletons and the remaining one of size three with two followers.
- There are 'm' pre-clusters with 'm -1' being singletons and the remaining one has 'n- m +1'members or 'n- m' followers.

For case (i), the number of intra-cluster (follower to follower) distance computations is 1. This is because the largest pre-cluster has only two followers and all other pre-clusters are singletons. However, for Case (ii), $\frac{(n-m)(n-m-1)}{2}$ follower to follower distance computations is saved. Thus depending on the type of data and also on the threshold, the minimum number of distance computations that can be saved ranges between 1 to $\frac{(n-m)(n-m-1)}{2}$.

Moreover, if the second stage of the algorithm recognizes more than one 'C_i's as a neighbor to a 'C_j' then those two 'C_i's will be merged without calculating any more inter-cluster distances among those 'C_i's, and hence the algorithm will be able to merge two clusters even without measuring the distances between the members of those clusters.

Thus, although the time complexity of the proposed algorithm is mathematically O(n^2), it is always an improved version of the Single Linkage algorithm, especially for large datasets as confirmed by the experimental results.

Table 13: Comparison of the proposed algorithm with the other well-known variants

Features	Proposed Algorithm	Classical Method	Sibson's method	MST approach by Gower and Ross
Distance matrix computation required	No	Yes	Yes	Yes for creating MST
Requires all the data points at once	No	Yes	Yes	Yes
Time Complexity	$O(n^2)$	$O(n^3)$	$O(n^2)$	$O(n^2)$
Convergence Time	Lowest	High	Low	Low
Space Complexity	$O(n)$	$O(n^2)$	$O(n)$	$O(n)$
Partial clustering of streaming data	Yes	No	No	No
Suitability of clustering large dataset	High	Lowest	Low	Low
Cluster Quality	Same as original			

9.4. Experimental results

a) Table 14&15 shows the performance of the accelerated Leader algorithm for various real-world datasets and for varying thresholds which are not shown by the authors [64]. It shows that in these cases also the accelerated leader outperforms the Leader algorithm by converging in a much shorter time without hampering cluster quality (RI =1).

b) Table 16 shows that only the initial distance matrix computation converges in a higher time in comparison to the proposed Accelerated SL algorithm and hence the latter proves itself a better approach as compared to those existing variants where distance matrix computation is necessary.

c) The RI of '1' in all cases of Table 16 implies that the final cluster quality is not affected by our algorithm and is identical to that produced by the Classical SL algorithm.

c) The RI of '1' in all cases of Table 16 implies that the final cluster quality is not affected by our algorithm and is identical to that produced by the Classical SL algorithm.
d) Table 16 shows that the algorithm provides poor results (few data have been clustered) when applied on datasets like Pen-digits for certain thresholds as compared to Letter and Shuttle. This proves that the performance of the SL algorithm is dataset dependent like all other clustering algorithms and hence the choice of clustering algorithm must be application dependent.
e) Figure 31, 32 shows that a large number of data points have already got clustered at the first stage and hence the second stage converges in a much shorter time as compared to the distance matrix methods. This proves the pre-clustering algorithm as a suitable one for accelerating the SL algorithm.
f) Figure 33 shows the percentage decrease in distance computations of the proposed algorithm from the distance matrix method. It can be seen that the improvement is more prominent with an increase in data size and hence Accelerated SL algorithm is highly suitable for handling large data volume.

Table 14: Results for Standard datasets

Data Set	Method	Threshold	Time (Approx. in Sec)	RI
Pendigits Testing	Leader	0.1	931.399	1.0
	Acc. Leader		112.371	
	Leader	5	927.648	1.0
	Acc. Leader		198.599	
	Leader	10	909.646	1.0
	Acc. Leader		276.219	
	Leader	20	513.322	1.0
	Acc. Leader		226.807	
	Leader	30	191.462	1.0
	Acc. Leader		111.171	
Pendigits Training	Leader	0.1	4343.854	1.0
	Acc. Leader		520.953	
	Leader	5	4426.636	1.0
	Acc. Leader		848.514	
	Leader	10	4279.032	1.0
	Acc. Leader		1165.177	
	Leader	20	2272.713	1.0
	Acc. Leader		951.735	
	Leader	30	746.385	1.0
	Acc. Leader		390.20	

Table 15: Results for large datasets

Data Set	Method	Threshold	Time (Approx. in Sec)	RI
Letter	Leader	3	4343.699	1.0
	Acc. Leader		2983.384	
	Leader	3.5	2696.959	1.0
	Acc. Leader		1961.99	
	Leader	4	1510.311	1.0
	Acc. Leader		1190.581	
	Leader	4.5	946.510	1.0
	Acc. Leader		791.393	
	Leader	5	573.241	1.0
	Acc. Leader		512.118	
Shuttle	Leader	3.5	11609.90	1.0
	Acc. Leader		6099.195	
	Leader	4	8024.368	1.0
	Acc. Leader		2893.298	
	Leader	4.5	6153.844	1.0
	Acc. Leader		2174.786	
	Leader	5	4422.03	1.0
	Acc. Leader		1643.392	
	Leader	5.5	3250.847	1.0
	Acc. Leader		1233.483	

Table 16: Comparison of the convergence time of the proposed Accelerated Single Linkage algorithm and the distance matrix

Dataset	Threshold	No of Clusters	No of distance Computations (approx. In millions)		Time (approx. In mins)		RI
			Distance matrix	Accelerated SL	Distance matrix	Accelerated SL	
Pendigits Testing	10	3426	6.1	1.09	10.8	6.3	1.0
	20	2269		1.10		10.34	1.0
	30	1106		0.61		2.8	1.0
Pendigits Training	10	7357	28	4.39	52.2	25.87	1.0
	20	4813		4.49		39.48	1.0
	30	2295		2.05		9.07	1.0
Letter	3	5183	199.99	16.98	372.4	70.23	1.0
	3.5	3481		12.03		44.84	1.0
	4	2199		8.21		28.05	1.0
	4.5	1505		9.15		30.04	1.0
	5	969		8.83		29.94	1.0

Shuttle	3.5	9149	1681	16.75	3172.56	642.15	1.0
	4	7199		13.16		70.90	1.0
	4.5	5840		11.00		54.92	1.0
	5	4745		9.21		42.65	1.0
	5.5	3890		8.09		34.29	1.0

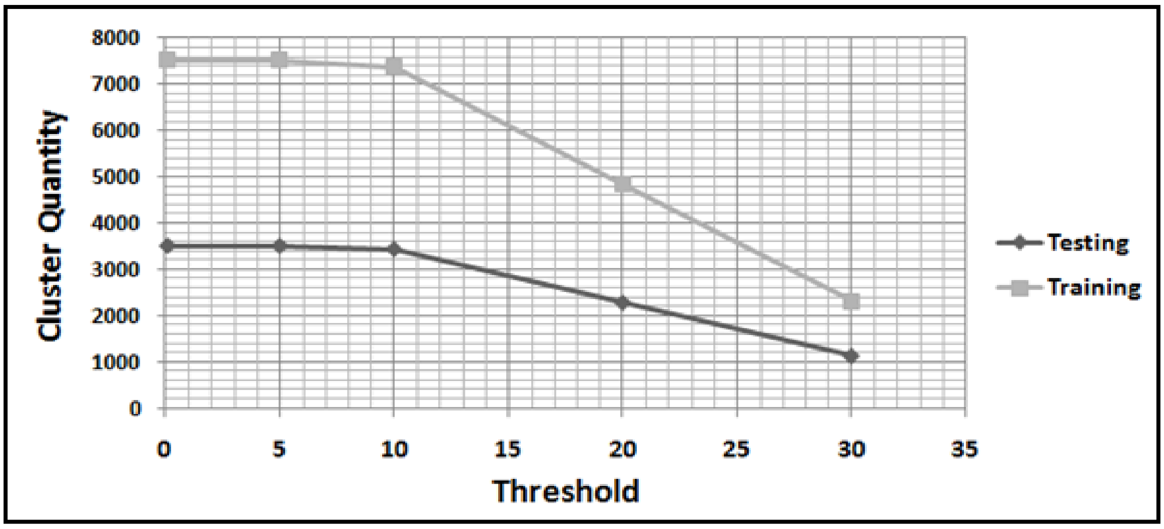


Figure 31: Variation of Cluster number with the threshold at the pre-clustering output for Pendigits dataset.

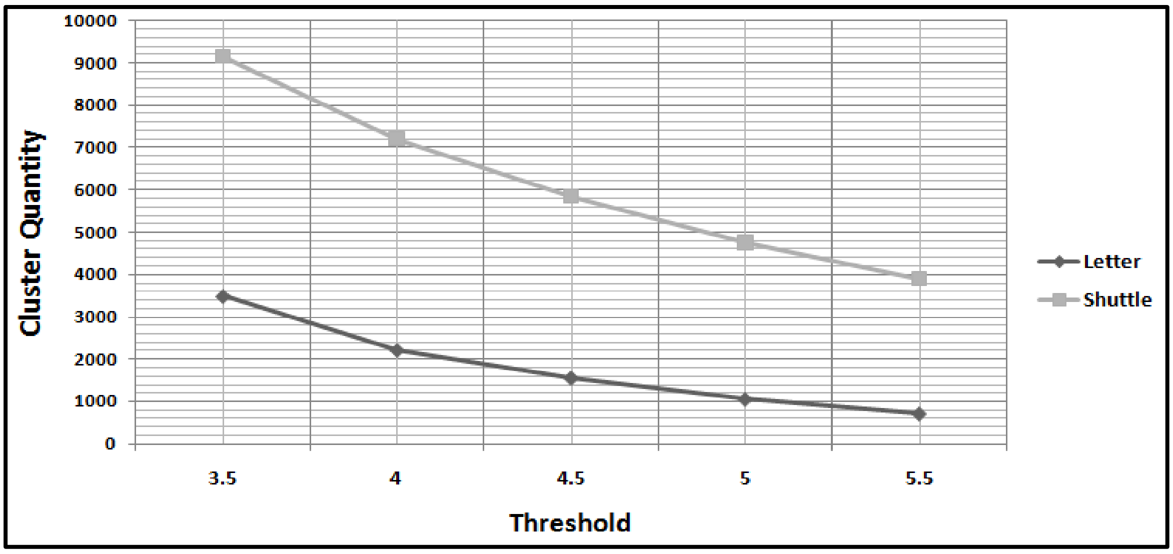


Figure 32: Variation of Cluster number with the threshold at the pre-clustering output for Letter and Shuttle dataset.

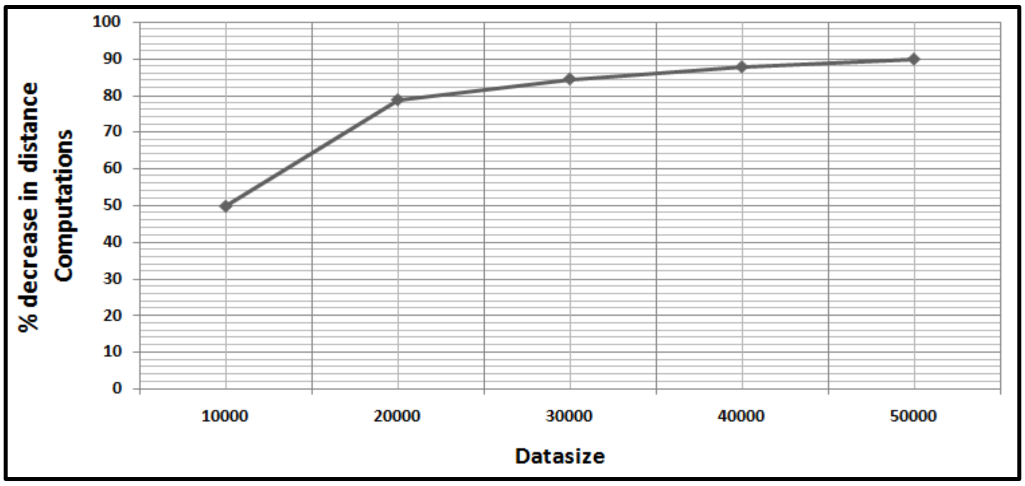


Figure 33: Percentage decrease in distance Computations vs. data size for Shuttle dataset (Threshold=3)

9.5. Conclusion

The paper presents a two-level, partially incremental Single Linkage algorithm that does not need the entire database in advance. The first stage of the algorithm creates some partially formed clusters using an accelerated version of the Leader algorithm and the second stage creates the final Dendrogram that does not have individual data points at its base but contains these partially formed clusters. The first stage employs triangle inequality to avoid the redundant distance computations and the second stage uses the property of the Single Linkage algorithm itself to merge clusters without directly measuring the distance between them. The proposed algorithm thus saves a lot of unnecessary distance computations which in turn decreases the convergence time of the algorithm to a large extent, as compared to its well-known variants. The proposed method although having an incremental stage does not get affected by the ordering of the data points as the final stage compensates for the effect of the ordering of the data. The major advantage of this algorithm is that it does not affect the cluster quality in the process of speeding up the method. A threshold is an important criterion of the Single Linkage algorithm and an increase in threshold value decreases both the quantity and quality (compactness) of clusters by absorbing distant points to the already available clusters. Hence it is the sole responsibility of the user to choose the correct threshold to get good clusters.

Chapter-10

Conclusion and future Scope of the research

These days the major challenge as faced by the data experts is to handle gigantic data volume coming at a very high speed. The problem is not only in handling the data but also analyzing as that demands high space and time. Another challenge as faced by the experts is the dynamic nature of these data sets i.e. changing frequently. The state of art algorithms is not capable of handling this high volume of dynamic data because of their high convergence time which tends to increase with increasing data volume. Hence to tackle this data avalanche, the traditional algorithms are needed to be modified or new algorithms are to be proposed with low time complexity. Clustering is an essential data mining tool that groups data based on certain parameters (user defined) thereby helping in retrieving information from an completely unlabeled data set. There are various kinds of clustering algorithms that differs based on the definition of similarity. Hierarchical clustering is one such well-known clustering algorithm that is widely used in various fields of bio informatics, bio medical research, pattern recognition, image processing etc. The major advantage of Hierarchical clustering algorithm is that it can provide a clear insight about cluster formation using a tree like structure called Dendrogram without any in-advance specification about the cluster number. Hierarchical clustering algorithm based on the 'definition of similarity' has been further classified into various linkage methods like Single Linkage, Complete Linkage, Average Linkage, Centroid Linkage etc. This paper has mainly focused on single and Complete Linkage algorithm because of their vast applicability in various research fields. The research work has proposed six different algorithms making Hierarchical clustering suitable for both large and dynamic databases. We have proposed a hardware implemented version of Leader algorithm that is used in the preclustering phase of various clustering methods. This algorithm converges much faster than the traditional Leader algorithm and guarantees speed irrespective of the data type. Thus, this in turn speeds up K-means, DBSCAN, Single Linkage, Complete Linkage, Average Linkage algorithms which uses this Leader algorithm in the pre-clustering phase. Apart from these, we have proposed algorithms for Complete Linkage algorithm making it suitable for large and dynamic datasets without affecting the properties of Complete Linkage clusters. Algorithms for Single Linkage clustering have also been proposed using triangle inequality and segment addition postulate thereby reducing the time complexity to a large extent. This speed up does not affect the cluster quality and produces identical clusters as that of the original algorithm. All the benefits of the algorithms are backed by experimental results and mathematical proofs. We have applied the algorithms on various real world or random large and standard, dynamic and static databases to prove the benefits experimentally.

Future work includes using hardware software co-design and hybridization techniques like combining partitional and hierarchical clustering algorithms to develop more efficient versions of clustering algorithms. Cleaning these data avalanche like removing noise from this vast and dynamic data can also be a potential research work. We have proposed various speedy techniques for single and complete Linkage algorithms, so proposing the same for other Linkage methods can also be a brilliant future scope of research.

References:

1. Younas, M. (2019) Research challenges of big data. *SOCA,* Vol. **13**: 105–107.
2. Dunham, M, H. (2003) Clustering, In*: Data Mining: Introductory and Advanced Topics*, New Delhi, India, Prentice Hall/Pearson Education: 125-128.
3. Hartigan, J. A. (1975) Introduction, In: *Clustering Algorithms*, New York, USA, John Wiley & Sons: 1-7.
4. Jin X., Han J. (2011) Partitional Clustering, In: *Encyclopedia of Machine Learning. Springer*, Boston, MA.
5. Rafsanjani, M.K., Varzaneh, Z.A., & Chukanlo, N.E. (2012). A Survey Of Hierarchical Clustering Algorithms, *Journal of Mathematics and Computer Science*, Vol.**5(3):** 229 – 240
6. Roux, M. (2018) A Comparative Study of Divisive and Agglomerative Hierarchical Clustering Algorithms, *Journal of Classification*, Springer Verlag, Vol. **35 (2)**: 345-366
7. Murtagh, F. and Contreras, P. (2012) Algorithms for hierarchical clustering: an overview, *Wiley Interdiscip. Rev. Data Min. Knowl. Discov*., Vol. **2**: 86–97
8. Mohbey, K. and Thakur G.s. (2013) An Experimental Survey on Single Linkage Clustering, *International Journal of Computer Applications.,* Vol. **76**. 6-11.
9. Saraçli, S., Doğan, N. and Doğan, İ. (2013) Comparison of hierarchical cluster analysis methods by cophenetic correlation. *J Inequal Appl.*, Vol. **203**
10. Wedel, M. and Kamakura, W. (2000) *Market Segmentation: Conceptual and Methodological Foundations*. ch-5:48-49
11. Saitluanga, B.L., *Himalayan Quality of Life: A Study of Aizawl City*, ch-3: pg. 44.
12. Frank, I. E. and Todeschini, R. *The Data Analysis Handbook*, pg.154.
13. Aggarwal Charu C., Hinneburg A., and Keim Daniel A. (2001) On the Surprising Behavior

of Distance Metric in High-Dimensional Space: In *Proceedings of ICDT 2001*: 420-434.

14. Steinbach, M., Ertoz, L. and Kumar, V. (2003) The challenges of clustering high dimensional data, In: *New Vistas in Statistical Physics -- Applications in Econophysics, Bioinformatics, and Pattern Recognition* , Springer-Verlag : 273–310
15. Madhulatha, T.S. (2012) An Overview on Clustering Methods, *IOSR Journal of Engineering,* Vol. **2(4)**: 719-725.
16. Small, H. and Koenig, M. E. D. (1977) Journal clustering using a bibliographic coupling method, *Information Processing & Management*, 13(5): 277–288
17. Onwukwe C.E and Ezeorah J.N. (2009) Application of single-linkage clustering method in the analysis of growth rate of gross domestic product (GDP) At 1990 Constant Basic Prices (Million Naira), *Global Journal of Mathematical Sciences,* 8(2)
18. Rieck, Konrad & Trinius, Philipp & Willems, Carsten & Holz, Thorsten. (2011) Automatic analysis of malware behavior using machine learning, *Journal of Computer Security,* Vol. **19**: 639-668.
19. Moroke, N. and Pulenyane, M. (2014) Clusters of Leading Death Causes in South Africa, Application of Hierarchical Agglomerative Clustering Technique, *Mediterranean Journal of Social Sciences*, Vol. **5(20**): 848-854
20. Maugeri, A., Barchitta, M., Basile, G. *et al.* (2021) Applying a hierarchical clustering on principal components approach to identify different patterns of the SARS-CoV-2 epidemic across Italian regions, *Sci Rep,* Vol. **11.**
21. Winder, Z., Sudduth, T.L., Fardo, D., et al. (2020) Hierarchical Clustering Analyses of Plasma Proteins in Subjects With Cardiovascular Risk Factors Identify Informative Subsets Based on Differential Levels of Angiogenic and Inflammatory Biomarkers, *Front Neurosci.,* Vol. **14(84)**.
22. Ghaemmaghami, H., Dean, D., Vogt, R., & Sridharan, S. (2012) Speaker attribution of multiple telephone conversations using a complete-linkage clustering approach, *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 4185-4188.
23. Ghaemmaghami, H., Dean, D., Vogt, R., & Sridharan, S. (2016) A study of speaker clustering for speaker attribution in large telephone conversation datasets, *Computer Speech & Language,* Vol. **(40)**: 23-45
24. Ghaemmaghami, H., Dean, D., & Sridharan, S., (2015) A cluster-voting approach for speaker diarization and linking of Australian broadcast news recordings, *In 2015 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2015*: 4829-4833
25. Sousa, Lúcia & Gama, João. (2014) The Application of Hierarchical Clustering Algorithms for Recognition Using Biometrics of the Hand, *International Journal of Advanced Engineering Research and Science (IJAERS) ,* Vol. **1(7)**.
26. Zhang Z, Murtagh F, Van Poucke S, Lin S, Lan P. (2017) Hierarchical cluster analysis in clinical research with heterogeneous study population: highlighting its visualization with R, *Ann Transl Med., ,* Vol. **5(4)**.
27. Lateef, Agbaje & Azeez, Musibau & Badmus, Suaibu & Adigun, Ganiyu. (2021) A decade of nanotechnology research in Nigeria (2010–2020): a scientometric analysis, *Journal of Nanoparticle Research,* Vol. **23**.

28. Lee J.S., Park S., Jeong H.W., et al. (2020) Immunophenotyping of COVID-19 and influenza highlights the role of type I interferons in development of severe COVID-19, *Sci Immunol.,* Vol. **5(49)**
29. Firdaus S. and Uddin M.A. (2015) A survey on clustering algorithms and complexity analysis, *Int. J. Comput. Sci.,* Vol. **12(2):** 62–85
30. Day, W.H.E., Edelsbrunner, H. Efficient algorithms for agglomerative hierarchical clustering methods, *Journal of Classification,*Vol. **1**, 7–24
31. Patel, S., Sihmar, S., & Jatain, A. (2015). A study of hierarchical clustering algorithms, In: *2nd International Conference on Computing for Sustainable Global Development (INDIACom)*: 537-541.
32. Sibson R. (1973) SLINK: An optimally efficient algorithm for the single-link cluster method. *The Computer Journal,* Vol. **16**: 30–34.
33. Rohlf F.J. (1982) Single-link Clustering Algorithms, In: *Handbook of Statistics, Vol.* ***2*** : 267–284.
34. El-Hamdouchi A. and Willett P. (1989) Comparison of hierarchic agglomerative clustering methods for document retrieval. *The Computer Journal,* Vol. **32**: 220-227.
35. Gower J.C. and Ross G.J.S. (1969) Minimum Spanning Trees and Single-Linkage Cluster Analysis, *Journal of the Royal Statistical Society, Series C (Applied Statistics),* Vol. **18 (1):** 54-64.
36. Patra B.K. and Nandi S. (2009) A Fast Single-Link Clustering Method Based on Tolerance Rough Set Model. In: *Proceedings of RSFDGrC 2009*, Vol. **5908**: 414–422.
37. Jin, C., Patwary, M., Agrawal, A., Hendrix, W., Liao, W., & Choudhary, A.N. (2013) DiSC : A Distributed Single-Linkage Hierarchical Clustering Algorithm using MapReduce In: *Proceedings of the 4th International SC Workshop on data intensive computing in the clouds (Data Cloud).*
38. Patra, B.K., Nandi, S. and Viswanath, P.(2011) A distance-based clustering method for arbitrary shaped clusters in large datasets*, Pattern Recognition,* Vol. **44(12)**, 2862–2870.
39. Jain A.K and Dubes R.C. (1988) Clustering Methods and Applications, In: *Algorithms for Clustering Data*: 58-80.
40. Rui Xu, Donald W. (2005) "Survey of Clustering Algorithms", *IEEE Trans Neural Netw*., Vol. **16(3):**645-78
41. Odilia Yim, Kylee T. Ramdeen (2015). "Hierarchical Cluster Analysis: Comparison of Three Linkage Measures and Application to Psychological Data", *Quantitative Methods for psychology*, Vol 11(1).
42. Defays, D.: An efficient algorithm for a complete link method, *The Computer Journal*, British Computer Society,Vol. 20 (4), (1977) 364–366.
43. Althaus E., Hildebrandt A. and Hildebrandt A.K. (2014) A Greedy Algorithm for Hierarchical Complete Linkage Clustering, In: *proceedings of International Conference on Algorithms for Computational Biology*, Vol. **8542**: 25-34.
44. Rehn, A., Possemiers, A., Holdsworth, J. (2018) Efficient hierarchical clustering for single-dimensional data using CUDA, in: *Proceedings of the Australasian Computer Science Week Multiconference*:1-10
45. Davidson I. and Ravi S.S. (2005) Agglomerative Hierarchical Clustering with Constraints: Theoretical and Empirical Results, In: *proceedings of European Conference on Principles*

of Data Mining and Knowledge Discovery, Vol. **3721**: 59-70.

46. Jang, J. & Jiang, H. (2019). DBSCAN++: Towards fast and scalable density clustering, In: *Proceedings of the 36th International Conference on Machine Learning*, *PMLR,* Vol **97**:3019-3029
47. Elkan, C., (2003) Using the Triangle Inequality to Accelerate K-Means, In: *Proceedings of the Twentieth International Conference on Machine Learning (ICML)*: 147–153.
48. Sarma, T.H., Viswanath, P. and Reddy, B.E. (2013) A hybrid approach to speed-up the k-means clustering method, *Int. J. Mach. Learn. & Cyber.,* Vol. **4**, 107–117.
49. T. Riesgo, Y. Torroja and E. de la Torre (1999) Design methodologies based on hardware description languages*, in IEEE Transactions on Industrial Electronics,* vol. **46(1)**, 3-12.
50. D. Becker, R. K. Singh and S. G. Tell (1992) An engineering environment for hardware/software co-simulation, In: *Proceedings of 29th ACM/IEEE Design Automation Conference*: 129-134
51. Pedroni ,Volnei A.(2010) *Circuit Design and Simulation with VHDL*, MIT Press.
52. Palnitkar, Samir.(2003) *Verilog HDL: A Guide to Digital Design and Synthesis*, Prentice Hall.
53. Bruno, F. (2021), *FPGA Programming for Beginners: Bring Your Ideas to Life by Creating Hardware Designs and Electronic Circuits with System Verilog*, Packt Publishing Limited.
54. md isa, mohd nazrin & Zainol Murad, Sohiful Anuar (2015), Field Programmable Gate Array (FPGA): From Conventional to Modern Architectures, In: *Digital and Analogue Electronics Circuits and Systems*: 53
55. https://www.xilinx.com/products/design-tools/ise-design-suite.html
56. R. R. Ryan and H. Spiller (1985) The C programming language and a C compiler, *IBM Systems Journal,* Vol. **24(1)**: 37-48
57. Rand, W.M. (1971) Objective Criteria for the Evaluation of Clustering Methods, *Journal of the American Statistical Association*, Vol. **66(336)**: 846–850.
58. https://archive.ics.uci.edu/ml/datasets/letter+recognition
59. https://archive.ics.uci.edu/ml/datasets/Statlog+(Shuttle)
60. http://archive.ics.uci.edu/ml/datasets/pen-based+recognition+of+handwritten+digits
61. https://tinyurl.com/Gaussian-numbersets
62. https://openmv.net/info/distillate-flow
63. https://goo.gl/RVtSBS
64. B. K. Patra, N. Hubballi, S. Biswas, and S. Nandi. (2010) Distance based fast hierarchical clustering method for large datasets. In: *Proceedings of the 7th international Conference on Rough Sets and Current Trends in Computing*, Vol. **6086**: 50–59.
65. Banerjee P., Chakrabarti A., Ballabh T.K. (2021) An Efficient Algorithm for Complete Linkage Clustering with a Merging Threshold. In: Sharma N., Chakrabarti A., Balas V.E., Martinovic J. (eds) Data Management, Analytics and Innovation. Advances in Intelligent Systems and Computing, Vol. **1175**. Springer, Singapore.
66. Sarma, T.H., Viswanath, P. & Reddy, B.E. (2013) A hybrid approach to speed-up the k-means clustering method, *Int. J. Mach. Learn. & Cyber.*, Vol. **4**, 107–117
67. Amini, A., & Wah, T.Y. (2013). LeaDen-Stream: A Leader Density-Based Clustering Algorithm over Evolving Data Stream, *Journal of Computational Chemistry*, Vol. **1(5)**, 26-31.
68. LaForest C.R., Steffan J.G. (2010) Efficient Multi-ported Memories for FPGAs,

In: *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*: 41-50

69. Nisha and Kaur P J 2015 Cluster quality-based performance evaluation of hierarchical clustering method, In: *1st International Conference on Next Generation Computing Technologies (NGCT)*: 649–653