ANALYSIS OF AN OPTIMIZED AGGLOMERATIVE CLUSTERING ALGORITHM

comparative analysis of standard agglomerative clustering algorithm and optimized divide & conquer algorithm

PROF. SUMAIYA THASEEN (*Mentor*)
Professor at School of Computer Science & Engineering
VIT University, Chennai Campus
Chennai, Tamil Nadu
India

Yash Mahendra (*Author*)
B.Tech student at School of Computer Science & Engineering
VIT University, Chennai Campus
Chennai, Tamil Nadu
India

S. Rahul Bhargav (*Author*)
B.Tech student at School of Computer Science & Engineering
VIT University, Chennai Campus
Chennai, Tamil Nadu
India

Somjeet Dasgupta (*Author*)
B.Tech student at School of Computer Science & Engineering
VIT University, Chennai Campus
Chennai, Tamil Nadu
India

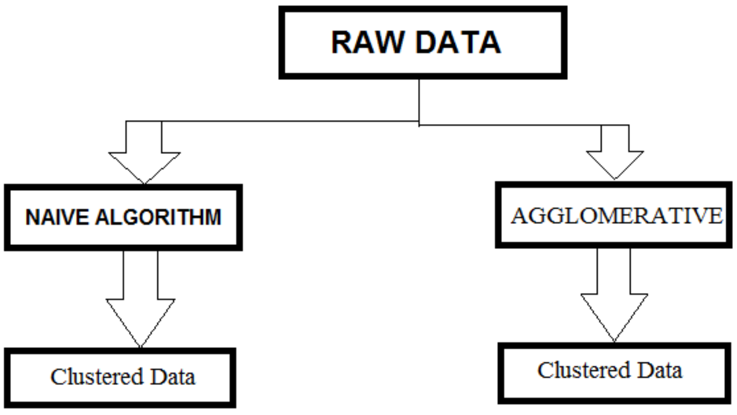
***Abstract*— Clustering is important in data analysis and data mining applications. It is the task of grouping a set of objects so that objects in the same group are more similar to each other than to those in other groups (clusters).**

There are different types of clusters: Well-separated clusters, Center-based clusters, Contiguous clusters, Density-based clusters, Shared Property or Conceptual Clusters. Predictive and the descriptive are the two main tasks of the data mining. Clustering can be done by the different no. of algorithms such as hierarchical, partitioning, grid and density based algorithms. Hierarchical clustering is the connectivity based clustering. Partitioning is the centroid based clustering, the value of k-mean is set. Density based clusters are defined as area of higher density then the remaining of the data set. Grid based clustering is the fastest processing time that typically depends on the size of the grid instead of the data. The grid based methods use the single uniform grid mesh to partition the entire problem domain into cells. In this paper which deals with only agglomerative hierarchical clustering, a comparison of traditional or naive agglomerative clustering algorithm and a faster, more optimized agglomerative clustering algorithm is done.

***Keywords*—** components; agglomerative; clustering**.**

I. INTRODUCTION

A cluster is a collection of data objects that are similar to one another within the same cluster and are dissimilar to the objects in other clusters. A good clustering algorithm is able to identity clusters irrespective of their shapes. Other requirements of clustering algorithms are scalability, ability to deal with noisy data, insensitivity to the order of input records, etc. The data is extracted using two learning approaches i.e. supervised learning or unsupervised clustering.



Supervised Learning: In this, training data includes both the input and the desired results. These methods are fast and accurate. The correct results are known and are given in inputs to the model during the learning process. Supervised models are neural network, Multilayer Perceptron, Decision trees.

Unsupervised Learning: The model is not provided with the correct results during the training. It can be used to cluster the input data in classes on the basis of their statistical properties only. Unsupervised models are different types of clustering, distances and normalization, k-means, self organizing maps.

From a machine learning perspective clusters correspond to hidden patterns, the search for clusters is unsupervised learning, and the resulting system represents a data concept. From a practical perspective clustering plays an outstanding role in data mining applications such as scientific data exploration, information retrieval and text mining, spatial database applications, Web analysis, CRM, marketing, medical diagnostics, computational biology, and many others. Presenting data by fewer clusters necessarily loses certain fine details (loss in data compression), but achieves simplification. It represents many data objects by few clusters, and hence, it models data by its clusters. Clustering is often one of the first steps in data mining analysis. It identifies groups of related records that can be used as a starting point for exploring further relationships.

II. Background Study

A. Hierarchical clustering algorithms

Hierarchical clustering is a method of cluster analysis which seeks to build a hierarchy of clusters. It is the connectivity based clustering algorithms. The hierarchical algorithms build clusters gradually.Hierarchical clustering generally fall into two types: In hierarchical clustering, in single step, the data are not partitioned into a particular cluster. It takes a series of partitions, which may run from a single cluster containing all objects to 'n' clusters each containing a single object. Hierarchical Clustering is subdivided into agglomerative methods, which proceed by series of fusions of the 'n' objects into groups, and divisive methods, which separate 'n' objects successively into finer groupings.

Advantages of hierarchical clustering

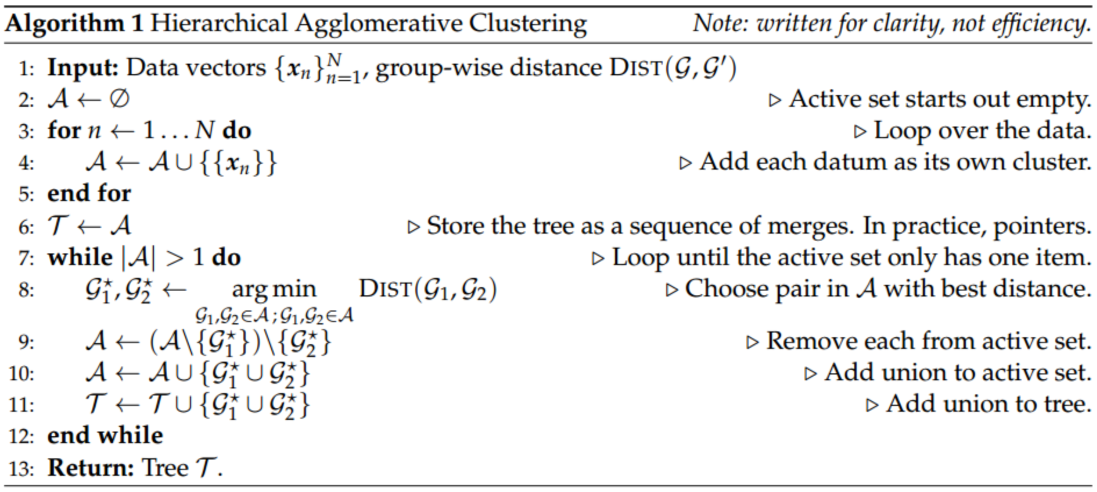
1. Embedded flexibility regarding the level of granularity.
2. Ease of handling any forms of similarity or distance.
3. Applicability to any attributes type.

B. Naive Algorithm and its time complexity

The traditional algorithm relies on comparing all the points to all the other points per iteration to find the closest of them as a pair to cluster, which eventually for large 'n', makes it too slow.

It basically is a brute force type algorithm which compares points even when they have little relevance or it can be said that they are clearly very far from each other based on the criterion of distance we are taking in account. It takes order of O(n^3) in time complexity. As visible from the algorithm below, we compare each point to every other point in every iteration till all the points are clustered eventually. That takes ∑(n-i)*(n-i-1) calculations in the i'th iteration to find the closest pair. Hence, a time complexity of O(n^3).

Figure 1: Algorithm of Hierarchical Agglomerative Clustering

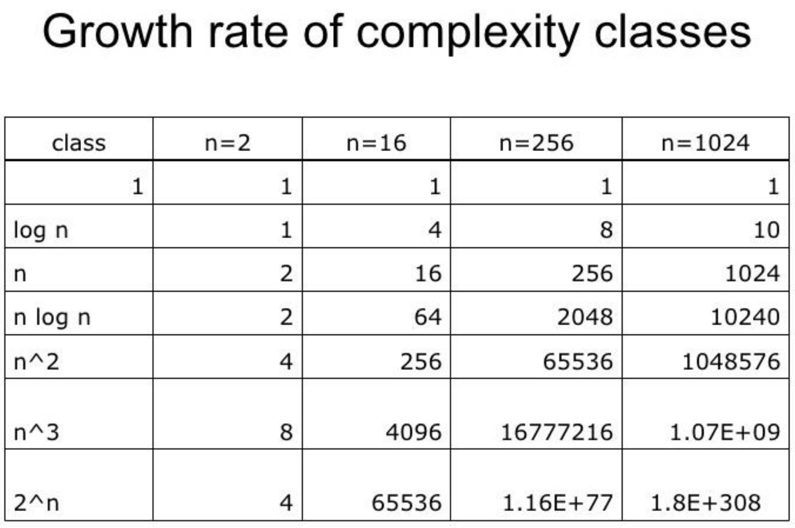


Algorithm 1 Hierarchical Agglomerative Clustering *Note: written for clarity, not efficiency.*

```
1:  Input: Data vectors $\{x_n\}_{n=1}^{N}$, group-wise distance $\textsc{Dist}(\mathcal{G},\mathcal{G}')$
2:  $\mathcal{A} \leftarrow \emptyset$                              ▷ Active set starts out empty.
3:  for $n \leftarrow 1 \ldots N$ do                                ▷ Loop over the data.
4:      $\mathcal{A} \leftarrow \mathcal{A} \cup \{\{x_n\}\}$       ▷ Add each datum as its own cluster.
5:  end for
6:  $\mathcal{T} \leftarrow \mathcal{A}$      ▷ Store the tree as a sequence of merges. In practice, pointers.
7:  while $|\mathcal{A}| > 1$ do                    ▷ Loop until the active set only has one item.
8:      $\mathcal{G}_1^\star, \mathcal{G}_2^\star \leftarrow \underset{\mathcal{G}_1,\mathcal{G}_2\in\mathcal{A};\,\mathcal{G}_1,\mathcal{G}_2\in\mathcal{A}}{\arg\min} \textsc{Dist}(\mathcal{G}_1,\mathcal{G}_2)$   ▷ Choose pair in $\mathcal{A}$ with best distance.
9:      $\mathcal{A} \leftarrow (\mathcal{A}\backslash\{\mathcal{G}_1^\star\})\backslash\{\mathcal{G}_2^\star\}$   ▷ Remove each from active set.
10:     $\mathcal{A} \leftarrow \mathcal{A} \cup \{\mathcal{G}_1^\star \cup \mathcal{G}_2^\star\}$   ▷ Add union to active set.
11:     $\mathcal{T} \leftarrow \mathcal{T} \cup \{\mathcal{G}_1^\star \cup \mathcal{G}_2^\star\}$   ▷ Add union to tree.
12: end while
13: Return: Tree $\mathcal{T}$.
```

III. Problem Specification & Proposed Model

Traditional hierarchical agglomerative clustering takes a time of the order of O(n^3), If per iteration, the number of comparisons for each step of clustering is reduced, the time complexity of the whole solution gets significantly reduced too . Hence, this project is about the implementation of such an algorithm on a 2-dimensional data space, to cluster them into respective cluster centers in the least time possible. Demonstration is done on a 2-D space, but the algorithm itself can be scaled up to more number of dimensions easily. Growth rate of complexity classes given below :



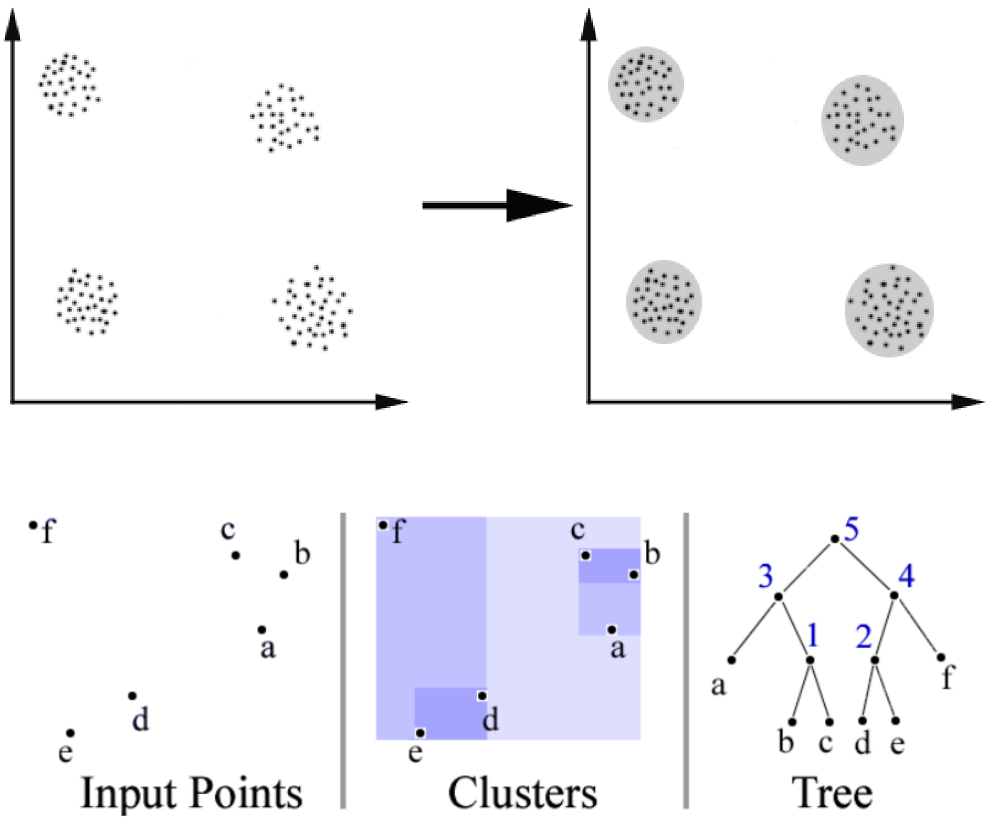
Growth rate of complexity classes

class	n=2	n=16	n=256	n=1024
1	1	1	1	1
log n	1	4	8	10
n	2	16	256	1024
n log n	2	64	2048	10240
n^2	4	256	65536	1048576
n^3	8	4096	16777216	1.07E+09
2^n	4	65536	1.16E+77	1.8E+308

Table 1: Growth rate of different time complexity algorithms

Problem Specification: Reduce the time it takes to cluster points by at least a full order in one part of the main iteration so that the overall time of O(n^3) gets reduced. The algorithm can be implemented in 2 dimensions for the demonstration, but must be robust & scalable enough for further expansion in number of dimensions of each data point. The criterion of the distance is flexible as with different situations and types of attributes, different types of mutual distances can be considered eg. manhattan distance, euclidean distance etc.

Figure 2: Graphical showcase of Clustering – **2.1:** Data Points are not clustered - **2.2:** Clustered Data points – **2.3:** Data points not clustered **– 2.4:** Clustering input points hierarchically – **2.5:** Tree structure of clustering



Goals:

Reduce the time to find the shortest distance pair points or centers to be clustered : The traditional comparisons of the naive algorithm take the order of O(n^2) to work. We aim to reduce that.

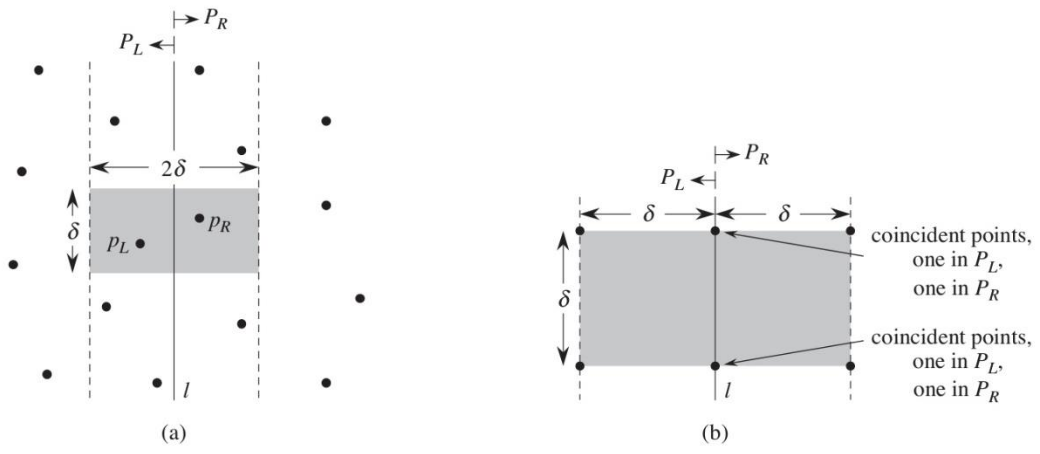
Analyze the optimized algorithm in action : Once the optimized algorithm is implemented, we'd analyze the clustering speed on dummy values representing real life data.

Proposed idea is to use the divide and conquer strategy to cluster the points such that per iteration, no two points which are not needed to be compared for their distances get compared at all in that iteration. Using euclidean distance and average linkage of the cluster centers, we'd cluster an array of 2 dimensional or planar points to demonstrate the algorithm's speed over traditional approach.

Divide & Conquer strategy
Unlike traditional agglomerative clustering algorithm, we have opted for a divide and conquer strategy for getting the pair of points or centers to be clustered. Divide and Conquer strategy ensures that we don't simply compare points that are irrelevant for

the moment, and ensures only possibly close points to be considered for the clustering in that iteration only. The divide and conquer strategy is made clear as shown below.

Figure 3: Graphical showcase of points getting compared after dividing the region. - 3.a: Considering points between the two strips. – 3.b: Calculating distance between the considered points



P_R
P_L
2δ
δ
p_R
p_L
l
(a)
P_R
P_L
δ
δ
δ
coincident points, one in P_L, one in P_R
coincident points, one in P_L, one in P_R
l
(b)

The optimized divide & conquer algorithm

Input: An array of n points P[]

Output: The smallest distance between two points in the given array.

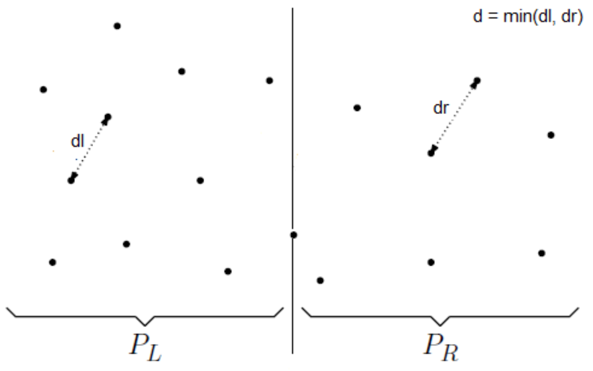
As a pre-processing step, input array is sorted according to x coordinates.

1) Find the middle point in the sorted array, we can take P[n/2] as middle point.

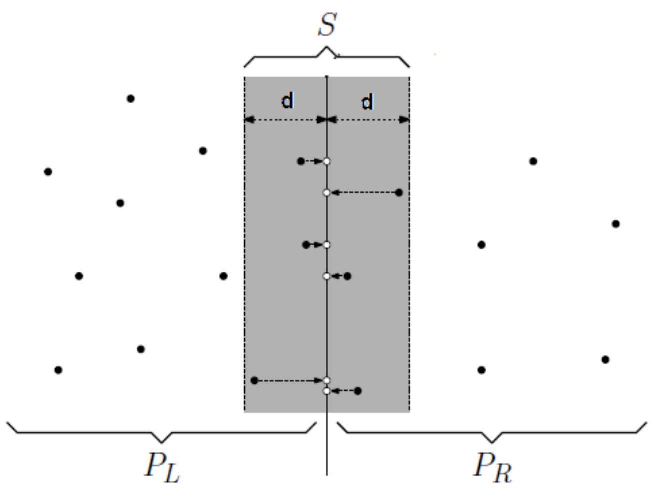
2) Divide the given array in two halves. The first subarray contains points from P[0] to P[n/2]. The second subarray contains points from P[n/2+1] to P[n-1].

3) Recursively find the smallest distances in both subarrays. Let the distances be dl and dr. Find the minimum of dl and dr. Let the minimum be ‘d’.

Figure 4: Taking Minimum distance d from both left andd right regions



4) From above 3 steps, we have an upper bound d of minimum distance. Now we need to consider the pairs such that one point in pair is from left half and other is from right half. Consider the vertical line passing through passing through P[n/2] and find all points whose x coordinate is closer than d to the middle vertical line. Build an array strip[] of all such points.



5) Sort the array strip[] according to y coordinates. This step is O(nLogn). It can be optimized to O(n) by recursively sorting and merging.
6) Find the smallest distance in strip[]. This is tricky. From first look, it seems to be a O(n^2) step, but it is actually O(n). It can be proved geometrically that for every point in strip, we only need to check at most 7 points after it (note that strip is sorted according to Y coordinate).
7) Finally return the minimum of d and distance calculated in above step (step 6).

What can be further done to optimize the algorithm further?

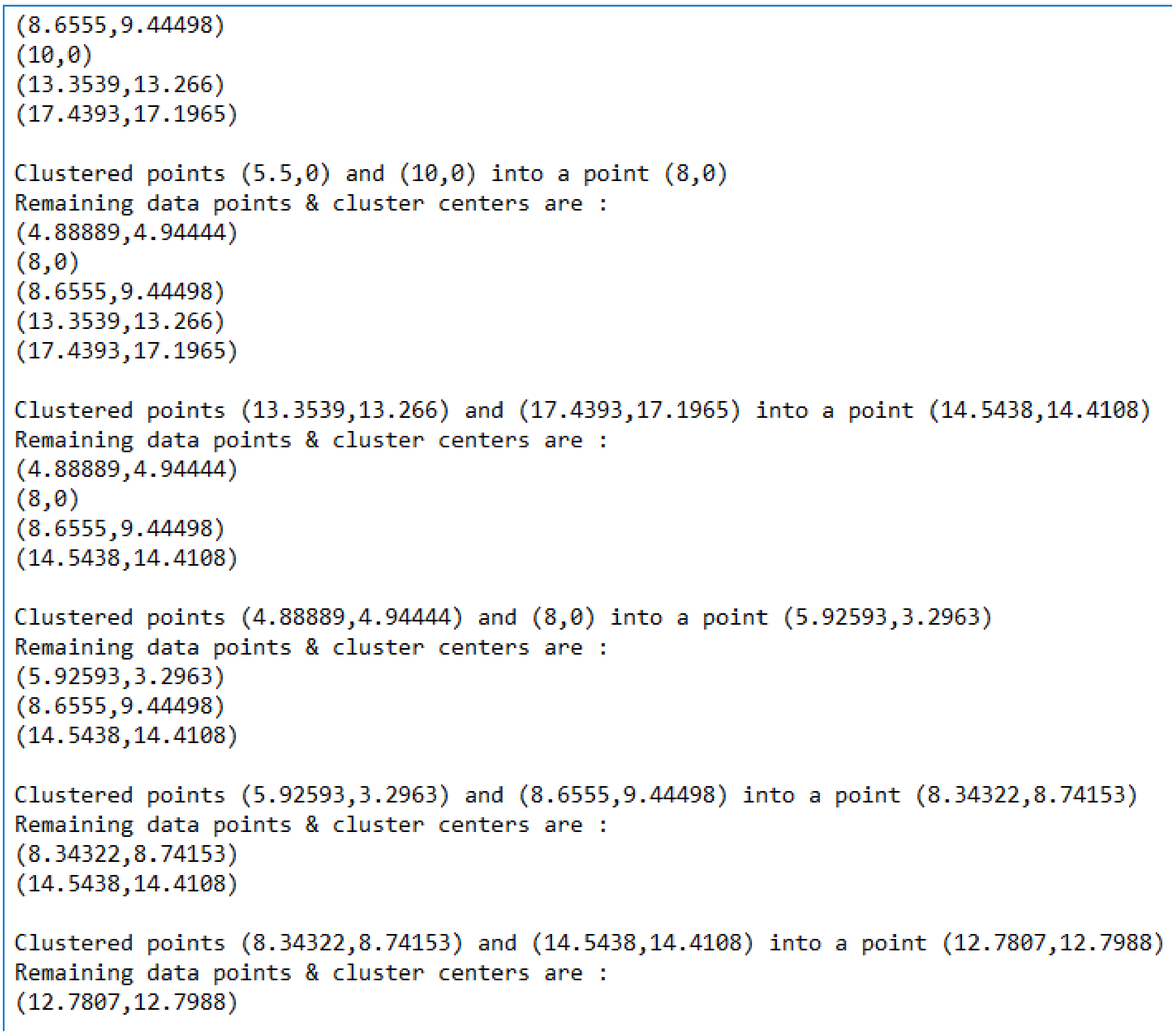
Concept of dynamic programming can be applied to remember the pairs of points which have already been compared before. As a result, if two points have been compared before, they can be cached and can be retrieved much faster than calculating the distance all over again every time.

The algorithm here, for demonstration purpose, is for 2 dimensions only but can be scaled up to more number of dimensions and the time complexity becomes O(d*n*log(n)) instead.

IV. Experimental Results & Conclusion

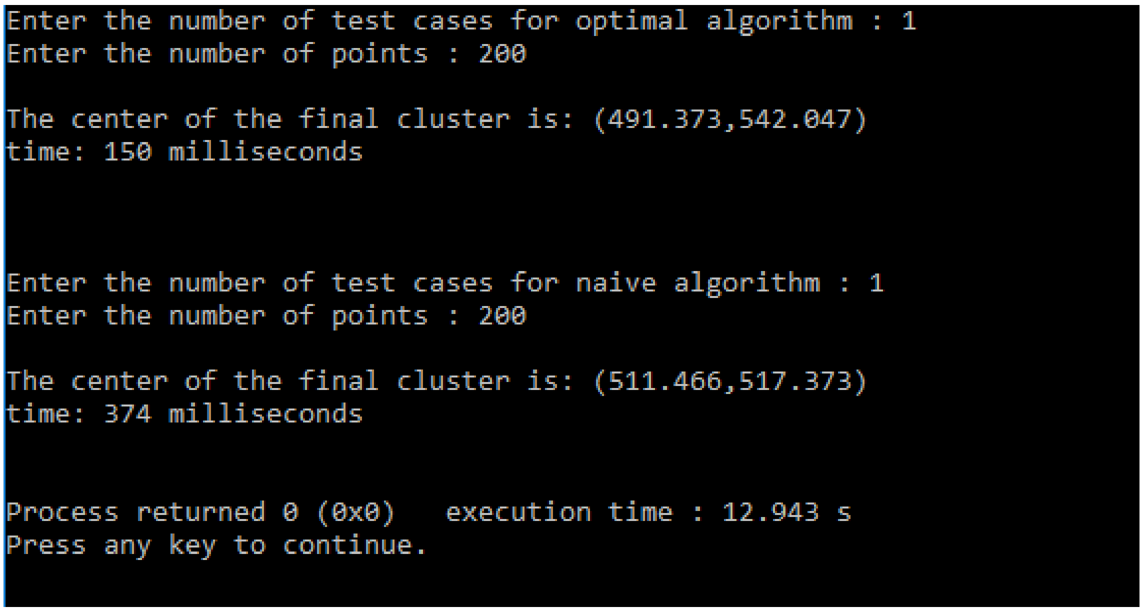
We've analyzed both the naive and the divide and conquer algorithms on a CSV data with 3 comma seperated values per data point. The first two values are the 2- dimensions or X and Y values of the data points, the third value of the point is its weight. Having the third value as weight enables us to input already a cluster as a point and changing the weight changes the number of points the cluster holds.
The algorithm takes O(n^2*log(n)) time as compared to the O(n^3) in the traditional algorithm.
We've implemented the code in C++ and used its time library to measure the start and stop times of the main algorithm. We can clearly see that the time difference is almost more than double for just 200 points. The effects are much more pronounced as the size of input increases further. The DENDROGRAM is generated in per level view representing each clustering in the order it occurs in the algorithm

Output 1: SAMPLE ITERATIONS (as shown in output file)


```
(8.6555,9.44498)
(10,0)
(13.3539,13.266)
(17.4393,17.1965)

Clustered points (5.5,0) and (10,0) into a point (8,0)
Remaining data points & cluster centers are :
(4.88889,4.94444)
(8,0)
(8.6555,9.44498)
(13.3539,13.266)
(17.4393,17.1965)

Clustered points (13.3539,13.266) and (17.4393,17.1965) into a point (14.5438,14.4108)
Remaining data points & cluster centers are :
(4.88889,4.94444)
(8,0)
(8.6555,9.44498)
(14.5438,14.4108)

Clustered points (4.88889,4.94444) and (8,0) into a point (5.92593,3.2963)
Remaining data points & cluster centers are :
(5.92593,3.2963)
(8.6555,9.44498)
(14.5438,14.4108)

Clustered points (5.92593,3.2963) and (8.6555,9.44498) into a point (8.34322,8.74153)
Remaining data points & cluster centers are :
(8.34322,8.74153)
(14.5438,14.4108)

Clustered points (8.34322,8.74153) and (14.5438,14.4108) into a point (12.7807,12.7988)
Remaining data points & cluster centers are :
(12.7807,12.7988)
```

on DATA 1 :

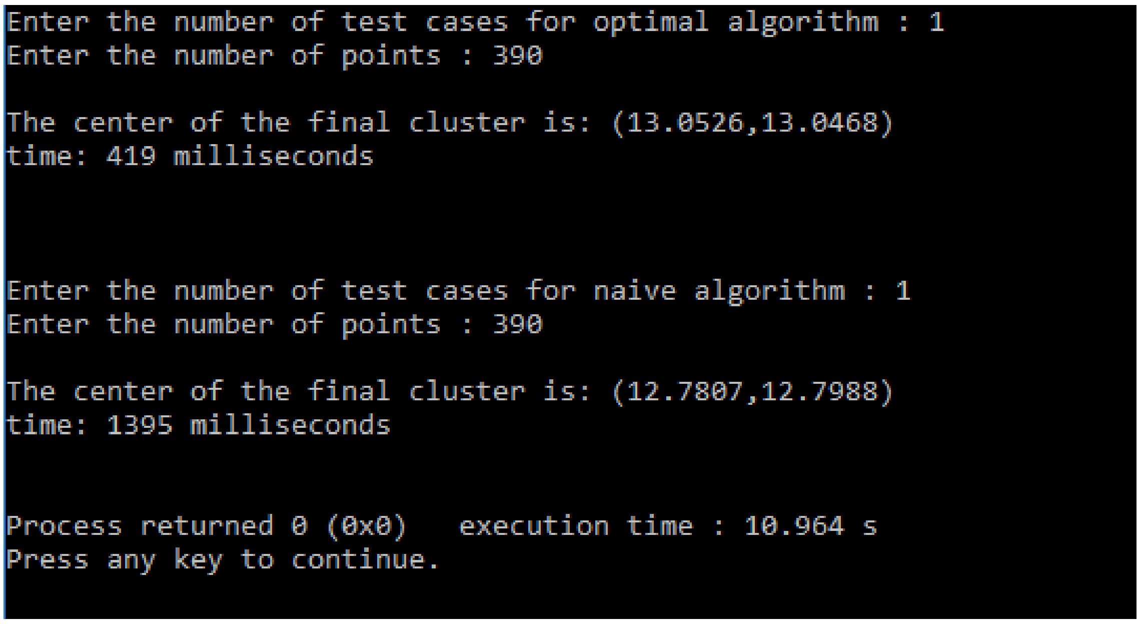
Output 2: Showing result (200 data points) of Optimal algorithm (150 ms time, (491,542) centre) and Naive algorithm (374ms time,(511,517) centre).



```
Enter the number of test cases for optimal algorithm : 1
Enter the number of points : 200

The center of the final cluster is: (491.373,542.047)
time: 150 milliseconds



Enter the number of test cases for naive algorithm : 1
Enter the number of points : 200

The center of the final cluster is: (511.466,517.373)
time: 374 milliseconds


Process returned 0 (0x0)   execution time : 12.943 s
Press any key to continue.
```

on DATA 2 :

Output 3: Showing result (390 data points) of Optimal algorithm (419 ms time, (13,13) centre) and Naive algorithm (1395ms time,(12,12) centre).



```
Enter the number of test cases for optimal algorithm : 1
Enter the number of points : 390

The center of the final cluster is: (13.0526,13.0468)
time: 419 milliseconds



Enter the number of test cases for naive algorithm : 1
Enter the number of points : 390

The center of the final cluster is: (12.7807,12.7988)
time: 1395 milliseconds


Process returned 0 (0x0)   execution time : 10.964 s
Press any key to continue.
```

Conclusion

We've successfully reduced the time it takes in the naive algorithm to cluster a set of points and demonstrated it on flat space in C++ language. We've implemented and analyzed a divide & conquer agglomerative clustering algorithm that takes O((n^2)*log(n)) time which is a full order better that O(n^3). The project does analysis of the algorithm on both kinds of data which are sufficiently sparse and sufficiently dense, working satisfactorily in both the cases.

Acknowledgment

We would like to thank Prof. Sumaiya Thaseen of VIT University, Chennai Campus, Chennai, India for her guidance throughout the subject and help in this project and in shaping up the idea of the algorithm.

References

[1] Pavel Berkhin, "A Survey of Clustering Data Mining Techniques", pp.25-71, 2002.

[2] M.Vijayalakshmi, M.Renuka Devi, "A Survey of Different Issue of Different clustering Algorithms Used in Large Data sets" , International Journal of Advanced Research in Computer Science and Software Engineering, pp.305-307, 2012..

[3] Anoop Kumar Jain, Prof. Satyam Maheswari "Survey of Recent Clustering Techniques in Data Mining", International Journal of Computer Science and Management Research, pp.72-78, 2012.

[4] Pradeep Rai, Shubha Singh" A Survey of Clustering Techniques" International Journal of Computer Applications, October 2010.

[5] Survey Paper on Clustering Techniques Amandeep Kaur Mann (M.TECH C.S.E) Department of Computer Science & Engineering of RIMT Institutions, Navneet Kaur (Assistant.Professor in C.S.E) Department of Computer Science & Engineering of RIMT Institutions, Mandi Gobindgarh, Punjab, India.