CHAPTER –VII

INDEXED K TWIN NEIGHBOUR CLUSTERING ALGORITHM

7.1 INTRODUCTION

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called cluster) are more similar (in some sense or another) to each other than those of other groups (clusters). These clusters presumably reflect some mechanisms at a work in the domain from which instances are drawn, a mechanism that causes some instances to bear a stronger resemblance to each other than they do the remaining instances. Clustering naturally requires different techniques to the classification and association learning. There are different methods used to express the result of clustering. The clusters identified may be exclusive, so that any instance belongs to only one group or they may be overlapping so that an instance may fall into several groups or they may be probabilistic, where by an instance belongs to each group with a certain probability or they may be hierarchical, such that there is crude division of instances into groups at the top level, and each of the group is filtered further – perhaps all the way down to individual instances. Really, the choice among these possibilities should be detected by the nature of mechanisms that are thought to underlie the particular clustering phenomenon. The major conventional algorithm fails in optimizing data structure concepts because of overlapping clusters and heterogeneous members of the same cluster. The current chapter of this thesis proposes a new clustering algorithm called Indexed K Nearest Twin Neighbour Clustering Algorithm (IKNTN).

7.2 CLUSTERING TECHNIQUES

Clustering is a useful technique for the discovery of data distribution and patterns in the underlying data. The goal of clustering is to discover both the dense and the sparse regions in a data set. Data clustering has been studied in the statistics, machine learning, and database communities with diverse emphases. The earlier approaches do not adequately consider the fact that the data set can be too large to fit into the main memory. In particular, they do not recognize that the problem must be viewed in terms of how to work with limited resources. The main emphasis has been to cluster with high accuracy as possible, while keeping the I/O cost high. Thus, it is not relevant to apply the classical algorithms in the context of data mining and it is necessary to investigate the principle of clustering to devise

efficient algorithms which meet the specific requirement of minimizing the I/O operation. It is to be noted that the basic principle of clustering hinges on a concept of distance metric or similarity metric. Since the data are invariably real numbers for statistical application and pattern recognition, a large class of metrics exists and one can define one's own metric depending on an explicit requirement. But in databases, there exist objects which cannot be ordered, and their attributes may be categorical in nature. Traditional clustering algorithms only handle numeric data which are not suitable for data mining purposes.

This chapter deals with different clustering techniques for data mining called IKNTN Clustering Algorithm. During the last several years, there has been an increasing interest in devising efficient clustering algorithms for data mining. In a short span of 2 to 3 years a very large number of new and interesting algorithms have been proposed. The aim of this chapter is to present a novel clustering algorithm and this will be useful for optimizing the performance of data structure and for future research in the area of optimizing the performance of data structure and algorithms. The clustering algorithms for numerical data are again categorized into two classes: partition and hierarchical. There are three different partitioning algorithms namely PAM, CLARA, and CLARANS.

7.2.1 CLUSTERING PARADIGMS

There are two main approaches available for clustering paradigms, namely *hierarchical clustering* and *partitioning clustering.* Besides, clustering algorithms differ among themselves in their ability to handle different types of attributes, numeric and categorical, in accuracy of clustering, and in their ability to handle disk-resident data.

7.2.1.1 PARTITIONING ALGORITHM

Partitioning algorithms is constructed partitions of a database with N data points, into a set of k clusters. The construction involves determining the optimal partition with respect to an objective function. There is approximately $kN/k!$ ways of partitioning a set of N data points into k subsets. An exhaustive enumeration method that can find the global optimal partition is practically infeasible except when N and k are very small. The partitioning clustering algorithm usually adopts the Iterative Optimization paradigm also it is one of the main limitations of the partitioning algorithm. It starts with an initial partition and uses an iterative control strategy. It tries swapping data points to see if such a swapping improves the quality of clustering technique. When swapping does not yield any improvements in

clustering, it finds a locally optimal partition. This quality of clustering is very sensitive to the initially selected partition. Some of the most famous partitioning algorithms are *k*-means and *k*-medoid.

In fields such as data structure, search and optimization method deals with how to find the best model(s) that makes minimum computation. Data mining concept especially the clustering concept can play very important role in optimizing the data structure concepts such as search. One of the biggest challenges in search, the search has to carryout throughout the data. So this increases the computational cost. A potential remedy to this problem is clustering the data and search done only within the predefined cluster. Clustered data structure may take less computational cost than the traditional data structure. IKNTN focuses on improvements to basic operations on data structures, specifically focus on the search operation performance on larger data, also introduces a new clustering algorithm specially designed to achieve optimal reduction of computational cost in any data structure. The chapters V and VI show the key role of data structure and its computational cost in LZW data compression algorithm. So in order to get the optimal reduction of computational cost, a novel clustering algorithm is designed and proposed. The IKNTN clustering algorithm solves the issues of traditional clustering algorithms like overlapping clusters and huge iteration. This algorithm runs with minimal computational cost compared with other partition clustering algorithms.

7.3 INDEXED K NEAREST TWIN NEIGHBOUR CLUSTERING ALGORITHM (IKNTN)

Clustering is group of the same or similar elements gathered or occurring closely together. This section of the thesis discusses a novel algorithm called Indexed K Nearest Twin Neighbour Clustering Algorithm. This algorithm is branched under partitioning clustering algorithms. Apart from the traditional partitioning clustering algorithms, the proposed algorithm has several advantages i.e., the clusters formed by the proposed algorithm don't have any overlapping. This means that each cluster has its own independent boundary and also unlimited number or huge iteration to find the efficient cluster which is not required. The latter sections of this chapter deals with the proposed algorithm which is specially designed to optimize the performance of any kind of data structure or this algorithm reduces the computational complexity of any data structure currently in use. The proposed algorithm finds the nearest twin Neighbour and all nearest twins are clustered and then all individual Clusters are indexed. Each cluster has only nearest twin patterns or a cluster contains only a

group of nearest twin patterns. After the process of clustering, the search is performed only within the particular cluster with the help of cluster Index.

The primary objective of the proposed clustering algorithm is to optimize the ability of data structure for a huge data set; especially this algorithm gives optimal reduction in any search, insertion or deletion using any fundamental data structure. The first phase of the algorithm is sorting the given dataset or data points in ascending or descending order. The sorting phase has a key role in the proposed algorithm. This phase partially brings the nearest data points to the neighboring position. For example, it is to be assumed that $D_1, D_2 \ldots D_n$ are the data points to be clustered using the IKNTN algorithm is shown in the Figure 7.1. After sorting phase, the data points are rearranged or it seems to be a sorted manner, then the data set must follow $1 < i < n$ order if it is in arranged in an ascending order. Then the neighboring values of i may be approximately equal to $i + 1$ that is $i - 1 \approx i \approx i + 1$ where i indicates the i^{th} element in the data set. So after this preprocessing stage the efficiency of the IKNTN algorithm is much improved. Sorting phase has a very important role in the computational complexity of this algorithm, if the algorithm is designed with low complexity sorting algorithm, then the cost of the algorithm is also reduced. Next step is to find the minimal length data available in the data set D and the minimal length which is stored in the variable L is shown in the figure-7.1. If the L and number of Clusters K are related i.e., if L increases then K increases or if L decreases then K reduces, this step can be combined with the sorting phase and then the additional computation is reduced. The next step is initializing the value for K, where K indicates the number of cluster, initially the K is initialized with zero. During the invention of new cluster from the data set, the value of K will be incremented. This algorithm does not define the total number of clusters at the initial stage, i.e., there is a possibility of generating a new cluster when processing the last data point $D_{|D|}$, where $|D|$ represents the size of the data set D. Next the algorithm starts iteration and fetches the first two data points D_i and D_{i+1} from the sorted or preprocessed data set and the Euclidean Distance of both data points are calculated but only the L length of the data points are used for calculating the Euclidean Distance i.e., shown in the Algorithm 7.1 line number 21. The Euclidean_Distance function returns the distance between two data points D_i and D_{i+1} respectively and it is stored in the variable ED. If the ED equals 0.0 then the algorithm is considered as both data points D_i and D_{i+1} as Twin Neighbour, if both the elements are twin grouped into Cluster C_k, initially $K = 0$ then C_0. Initially the size of $|C_0|$ is zero, but after formation of all available clusters for data points in D no empty cluster is

available or the minimum number of members inside each clusters is at least one. The size of each cluster is represented as $|C_i|\ where\ i = 0,1,2,..k-1$. D_i is added to the K^{th} cluster. If both elements are D_i and D_{i+1}, Euclidean Distance is not zero then are the elements not twin, then only a new index is generated with D_i for the K^{th} cluster using the formula given in the figure 7.1. The new index is assigned to $Index[K]$ and a new cluster is created by incrementing K. The iteration is continued up to $|D|$. The algorithm creates K number of clusters with unique index (shown in the figure 7.3).

1. Algorithm IKNTN(D)
2. Sort the elements of D_i where $i = 1 \ldots |D|$
3. Find the $\min\{|D_i|$ and store to L
4. Initialize K=0
5. For i =1 to |D|
6. ED=Euclidean_Distance(D_i, D_{i+1}, L)
7. If ED = 0.0 then // Twin Neighbour
8. Add D_i to the $C_K{}^{th}$ Cluster // use appropriate data structure
9. Else
10. Index[K]= Calculate_Index(D_i, L)//Assigns the K^{th} Cluster index and generate an index table
11. Increment K by one
12. End if
13. End for
14. Read the pattern X is be searched
15. In= Calculate_Index(X_i, L)
16. Search for the In in the Index table and store Index in N
17. If search fails X is not exist
18. Else Search in the N^{th} Cluster using appropriate data structure
19. End if
20. Function Euclidean_Distance(D_i, D_{i+1}, L)
21. Return $\sqrt{\sum_{j=1}^{L}\left(D_{i_j} - D_{(i+1)_j}\right)}$
22. End function
23. Function Calculate_Index(D_i, L)
24. $return(\sum_{i=1}^{L}\left(\left(ascii\left(CUR_{VAL}(i^{th}\ charecter\ of\ D_i)\right)-1\right)*\left(256^{L-i}\right)\right)+1)$
25. End function

Figure 7.1 Indexed K Nearest Twin Neighbour Algorithm

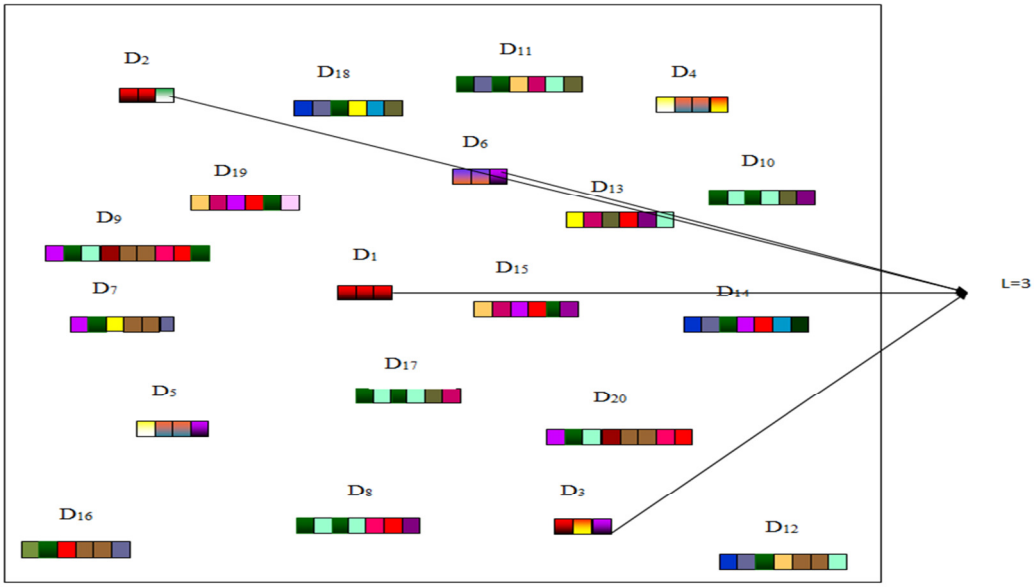


Figure 7.2 Data points before clustering

The next phase of the algorithm is the search to perform the lookup operation, and the lookup value is read and stored in the X, the index values for the X using the formula are to be found as shown in figure 7.1. If the algorithm finds the index in index table $Index[i]$, then the search is performed in the particular cluster C_i, else the element does not exist in the data set. The computational complexity of the algorithm purely based on the data structure employed with the IKNTN algorithm. The figure 7.2 shows the data points and the figure 7.3 illustrates the IKNTN clustering algorithm (after clustering the data points using IKNTN).

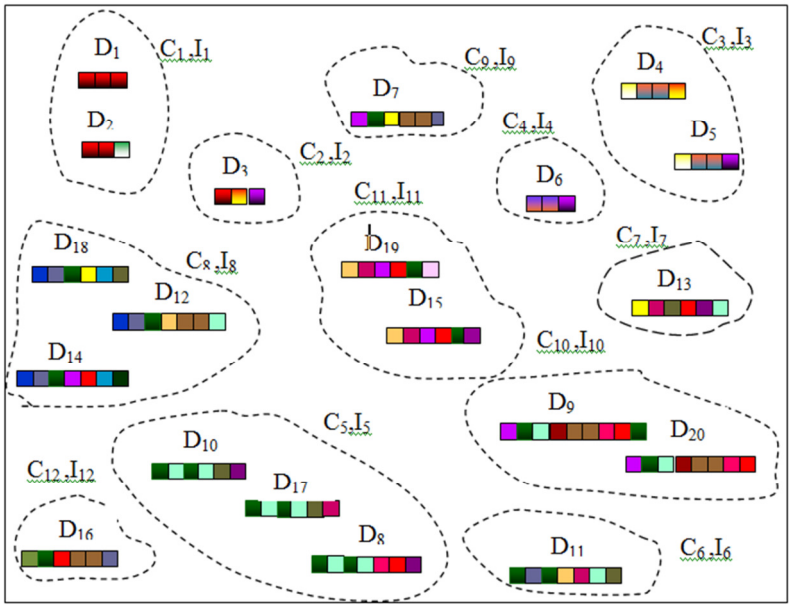


Figure 7.3 Example of IKNTN clustering algorithm (Data points after clustering)

7.3.1 COMPUTATIONAL COMPLEXITY ANALYSIS OF IKNTN ALGORITHM

The IKNTN algorithm has three phases such as sorting, clustering and searching. The computational complexity is calculated by the sum of computation required for all these three phases. In the first phase, computation is calculated depending on the sorting algorithm used with the phase. For example, if the quick sort is applied then the computational cost $O\ (n\ log\ n)$ refer Theorem 3.8., if Binary insertion sort (BIS) means $O\left(log\left((n-1)!\right)^{\frac{1}{(n-1)}}\right)$ $Computation$ required. In the next two phases, Computational complexity is purely based on the data structure employed with the algorithm. This thesis experiments four primary data structures with IKNTN, namely BST, Linear Array, Hash table and Binary insertion sort with Binary search.

Definition -1

Number of clusters is K. Length of each cluster is represented as $|C_i|$ where $i = 1,2,3 \ldots k$. D is the data set and size of the data set is |D|, each data point in the data set is represented as D_i where $i = 1,2,3 \ldots |D|$.

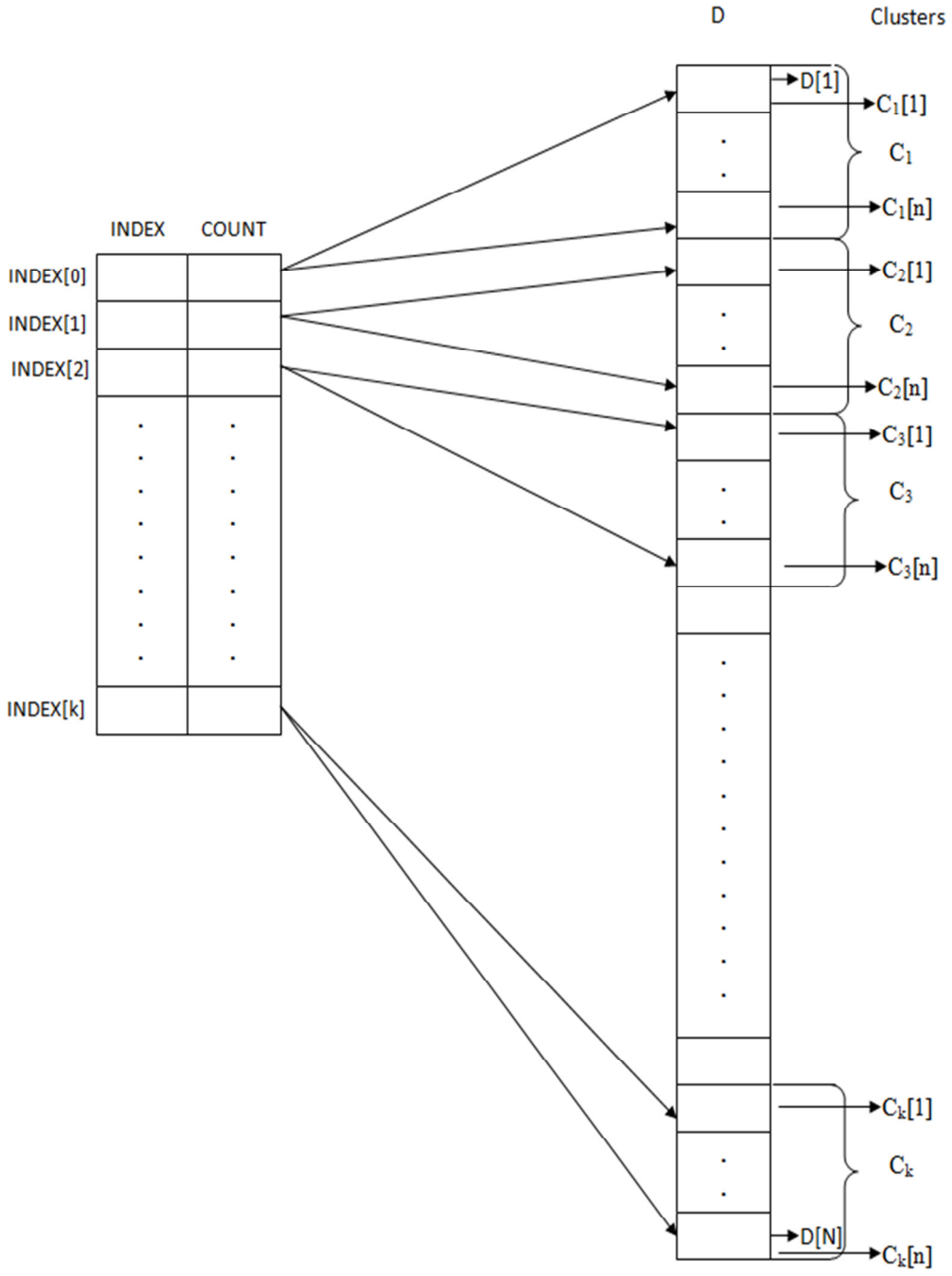
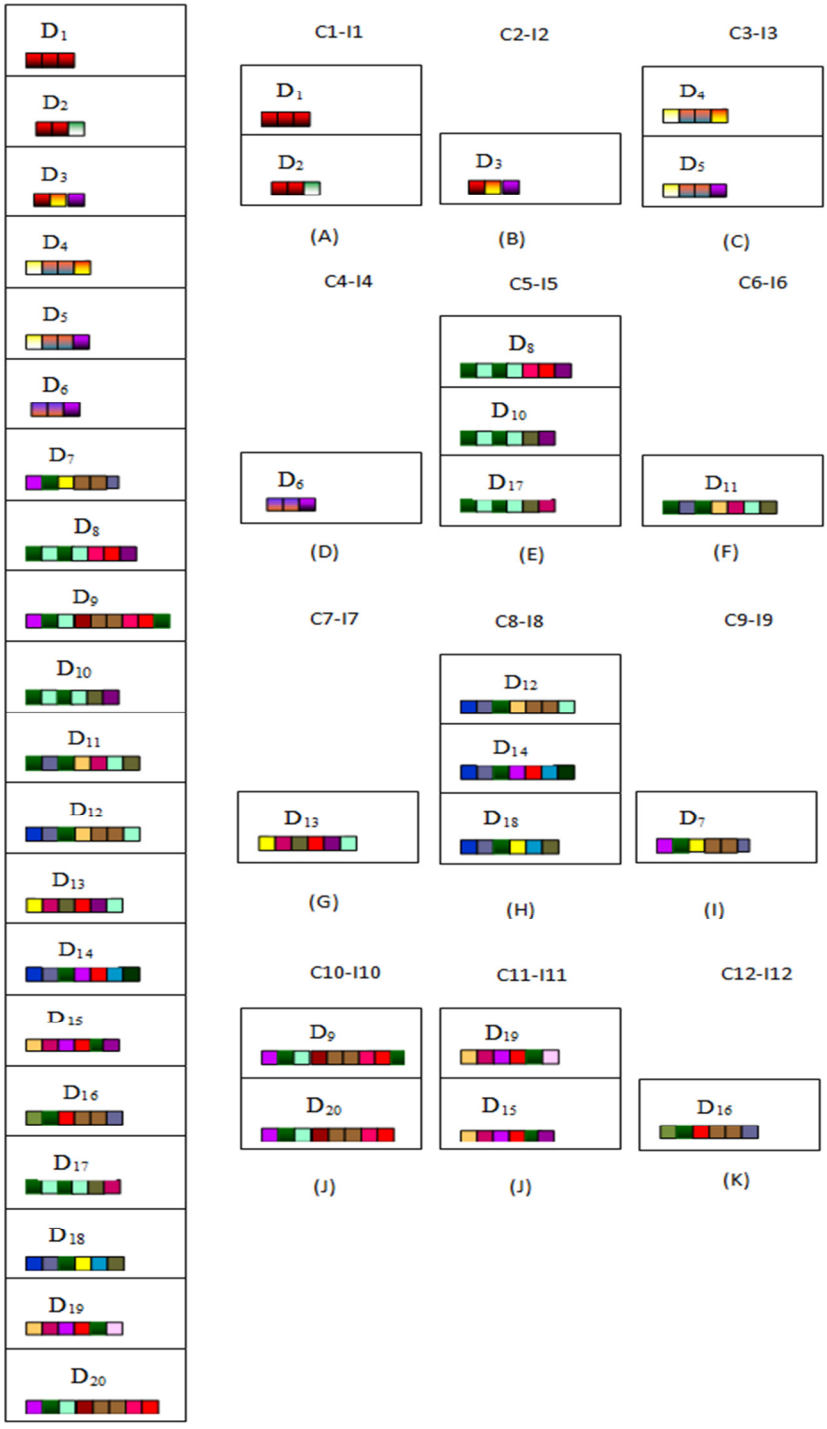


Figure 7.4 Illustrates Indexed clusters using IKNTN algorithm

7.3.1.1 LINEAR ARRAY IMPLEMENTATION OF IKNTN ALGORITHM



7.5 Linear array representation of IKNTN clustering algorithm

In the proposed IKNTN clustering algorithm, the linear array is employed with each cluster that is shown in the figure 7.5. Each time a new member is found in D, the algorithm inserts the member or data point to the particular cluster that has the same twin Neighbour. Otherwise it will create a new cluster and store the data point or the member in that cluster. In

the linear array implementation each cluster is uniquely maintained with different array data structures. The algorithm prefers hash table to build cluster index table.

Theorem 7.1 Linear Array Implementation of IKNTN Algorithm takes

$$O\left(\frac{1}{|D|}\left(|D| + (|D|log\ |D|) + |D| + (|D| * (\frac{1}{K}\sum_{i=1}^{K}\left(\frac{|C_1|+1}{2}\right))\right)\right) Computation\ per\ search$$

Proof:

After fetching each data points D_i during clustering phase single insertion is made to the corresponding cluster is shown in the figure 7.4, insertion to the linear array approximately required less than O(1) time, and to fulfill single iteration approximately O(1), then the total cost per iteration is approximately calculated as O(1). Then total computational complexity of clustering phase is |D| * O (1) that is simplified as O (|D|).

So the total cost is $O\ (|D|\ log\ |D|) + |D|$ for the sorting and clustering phase.

Next the average computation required for the search in linear representation of clusters has to be calculated.

Search in an array requires

$$\frac{|D|+1}{2} Computation$$

So the total average cost is calculated by

$$\frac{1}{K}\left(\frac{|C_1|+1}{2} + \frac{|C_2|+1}{2} + \frac{|C_3|+1}{2} + \cdots + \frac{|C_k|+1}{2}\right) \quad (7.1)$$

The above equation is simplified as

$$\frac{1}{K}\sum_{i=1}^{K}\left(\frac{|C_i|+1}{2}\right) \quad (7.2)$$

So the average cost per search is shown in Figure 7.2. Then the total cost is calculated as

$$|D| * \left(\frac{1}{K}\sum_{i=1}^{K}\left(\frac{|C_1|+1}{2}\right)\right) \quad (7.3)$$

Computation time required per indexing $O(1)$ because hash table preferred

So the total computation required for indexing is $|D| * O(1) = O(|D|)$

Then the total computational cost including three phases of the algorithm is calculated as follows:

$$O\left(\frac{1}{|D|}\left(|D| + (|D|log\ |D|) + |D| + (|D| * (\frac{1}{K}\sum_{i=1}^{K}\left(\frac{|C_1|+1}{2}\right))\right)\right) \quad (7.4)$$

7.3.1.2 BST IMPLEMENTATION OF IKNTN ALGORITHM

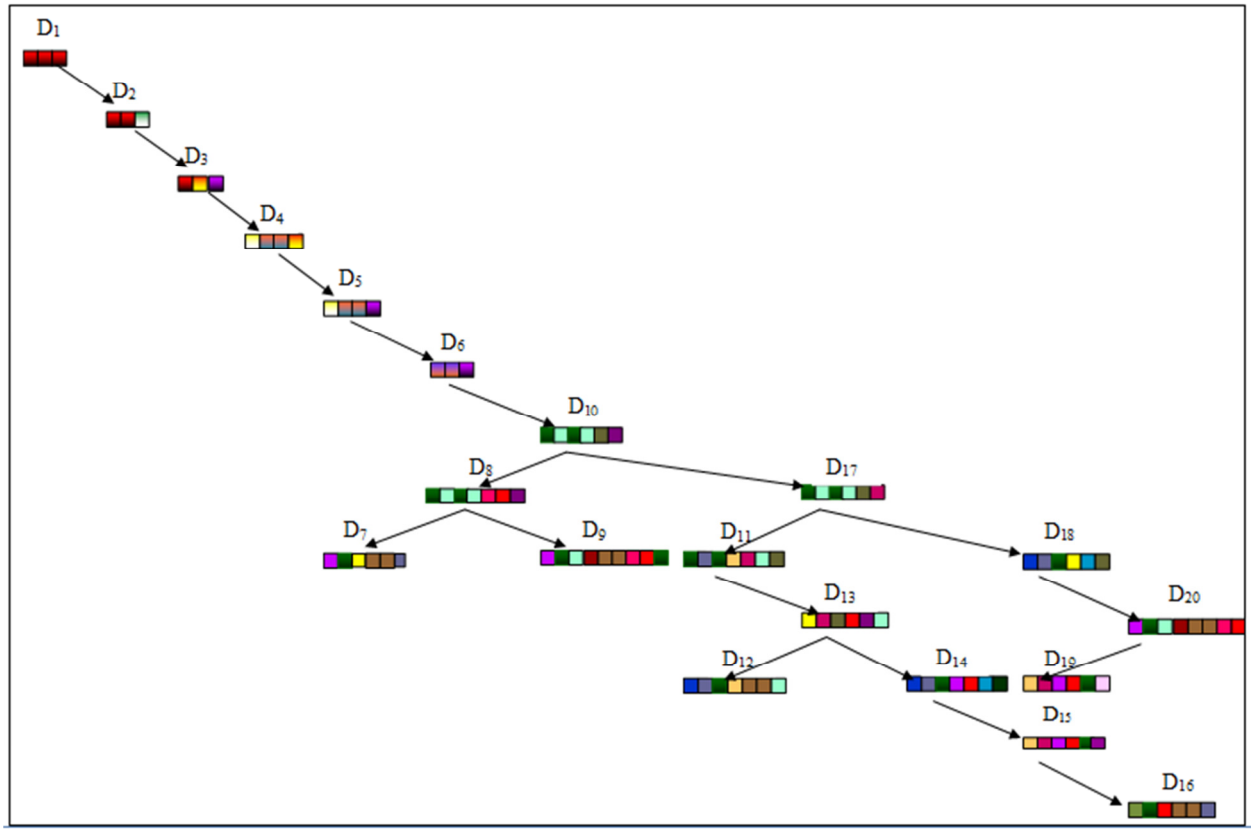


Figure 7.6 BST Structure before Clustering

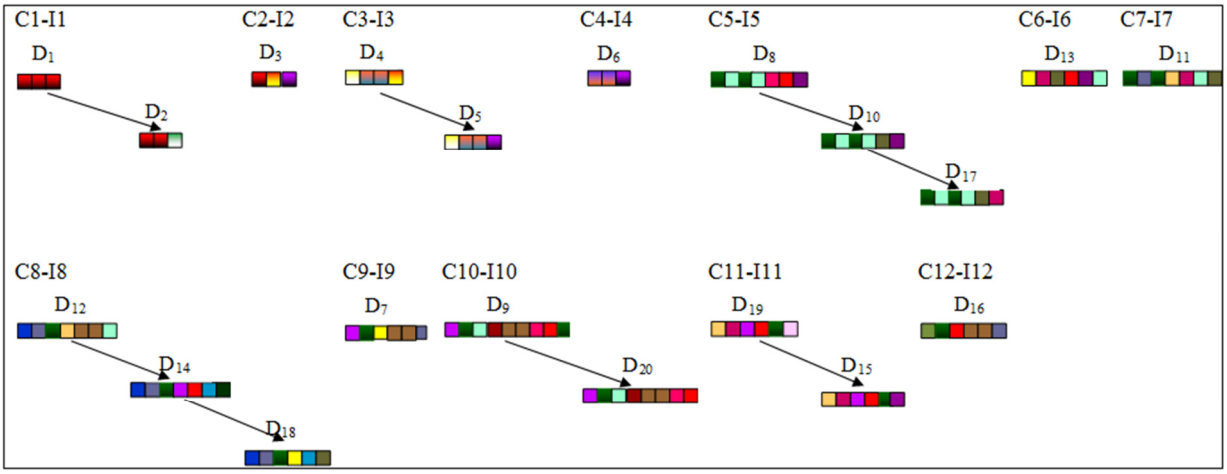


Figure 7.7 Example BST Structure after Clustering using IKNTN Algorithm

In the proposed IKNTN clustering algorithm, the BST array is employed with each cluster that is shown in the figure 7.7. Each time a new member is found in the D the algorithm inserts the member or data point to the particular cluster which has the same twin Neighbour. Otherwise it will create a new cluster and store the data point or the member in

that cluster. Each time a new cluster is identified in the data set for which the algorithm automatically creates a new binary search tree (BST) data structure. In the BST implementation each cluster is uniquely maintained with unique BST data structure. This approach also prefers hash table to build cluster index table. The figure 7.6 shows the BST data structure before clustering and figure 7.7 shows the BST data structure employed in each cluster during IKNTN clustering algorithm.

Theorem 7.2 BST implementation of IKNTN algorithm takes

$$O\left(\frac{(|D|\ log\ |D|) + +(|D| * (\frac{1}{K}\sum_{i=1}^{k}(|C_i| * ((\log\prod_{i=1}^{|C_i|} i)^{\frac{1}{|C_i|}}))) + |D| + \left(\frac{1}{K}\sum_{i=1}^{k}(|C_i|\log|C_i|\right)}{|D|}\right)$$

$Computation\ per\ search$

Proof: After fetching each data points D_i during clustering phase single insertion is made to the corresponding cluster which is shown in the figure 7.7

Insertion to the BST approximately requires less than $O\ log(n)$ time, and to fulfill single iteration is approximately calculated as follows:

To build a BST requires approximately

$$O\ N * \log((\prod_{i=1}^{N} n_i)^{\frac{1}{N}})\ computation$$

So the total average cost for constructing clusters using BST is calculated as follows:

$$\frac{1}{K}\left(O\left(|C_1| * log\left(\left(\prod_{i=1}^{|C_1|} i\right)^{\frac{1}{|C_1|}}\right)\right) + \left(|C_2| * log\left(\left(\prod_{i=1}^{|C_2|} i\right)^{\frac{1}{|C_2|}}\right)\right) + \left(|C_3| * \right.\right. \tag{7.5}$$

$$\left.\left. log\left(\left(\prod_{i=1}^{|C_3|} i\right)^{\frac{1}{|C_3|}}\right)\right) + \cdots + \left(|C_k| * log\left(\left(\prod_{i=1}^{|C_k|} i\right)^{\frac{1}{|C_k|}}\right)\right)\right)$$

By simplifying the above equation, the following is obtained.

$$\frac{1}{K}\sum_{i=1}^{k}\left(|C_i| * log\left(\left(\prod_{i=1}^{|C_i|} i\right)^{\frac{1}{|C_i|}}\right)\right) \tag{7.6}$$

So total cost is calculated as follows:

$$|D| * \left(\frac{1}{K} \sum_{i=1}^{k} \left(|C_i| * log \left(\left(\prod_{i=1}^{|C_i|} i \right)^{\frac{1}{|C_i|}} \right) \right) \right) \quad (7.7)$$

Next have to calculate the search complexity using BST.

The search complexity per cluster is $O(|C_i|\ log\ |C_i|)$

Then the average computation is calculated by

$$O\left(\frac{1}{K} \sum_{i=1}^{k} (|C_i|\ log\ |C_i|) \right) \quad (7.8)$$

So the total computation required including indexing using hashing is

$$\left(\frac{(|D|\ log\ |D|) + (|D| * (\frac{1}{K} \sum_{i=1}^{k} (|C_i| * ((\log \prod_{i=1}^{|C_i|} i)^{\frac{1}{|C_i|}})))) + |D| + \left(\frac{1}{K} \sum_{i=1}^{k} (|C_i| \log |C_i| \right)}{|D|} \right) \quad (7.9)$$

7.3.1.3 HASH TABLE IMPLEMENTATION OF IKNTN ALGORITHM

In the proposed IKNTN clustering, algorithm is implemented using chained hash table. It is used with each cluster that is shown in the figure 7.8. Each time a new member is found in the D for which the algorithm inserts the member or data point to the particular cluster that has the same twin Neighbour, and the clusters function with the help of chained hash table. Otherwise it will create a new cluster and store the data point or the member in that cluster. Each time a new cluster is identified in the data set, the algorithm automatically creates a new Chained hash table. In this implementation, each cluster is uniquely maintained with unique hash table. In this approach, both clusters and index tables are maintained with hash table but with small variance: clusters are maintained with the chained hash table and index is maintained with the ordinary hash table. The figure 7.8 shows the chained data structure before clustering and figure 7.9 shows the chained data structure employed in each cluster during IKNTN clustering algorithm

Theorem 7.3 Chained hash table implementation of IKNTN algorithm takes

$$O\left(\frac{(|D|\ log\ |D|) + \left(|D| * \left(\frac{1}{K} \sum_{i=1}^{k} \left(\frac{\sum_{i=1}^{|C_i|} \alpha_i}{|C_i|} \right) \right) \right) + |D| + \left(\frac{1}{K} \sum_{i=1}^{k} (O(1 + \alpha_i)) \right)}{|D|} \right) computation$$

Proof: After fetching each data points D_i during clustering phase single insertion is made to the corresponding cluster which is shown in the figure 7.6.

Insertion to the BST approximately requires less than $O\ log(n)$ time, and to fulfill single iteration approximately is calculated as follows

To build a Chained Hash table requires approximately

$\frac{\sum_{i=1}^{n}\alpha_i}{n}\ time\ \ based\ on\ the$ Theorem

So the total average cost is of constructing clusters using Chained hash table is calculated as follows:

$$\frac{1}{K}\left(\left(\frac{\sum_{i=1}^{|C_1|}\alpha_i}{|C_1|}\right)+\left(\frac{\sum_{i=1}^{|C_2|}\alpha_i}{|C_2|}\right)+\left(\frac{\sum_{i=1}^{|C_3|}\alpha_i}{|C_3|}\right)+\cdots+\left(\frac{\sum_{i=1}^{|C_k|}\alpha_i}{|C_k|}\right)\right) \tag{7.10}$$

By simplifying the above equation the following is obtained:

$$\frac{1}{K}\sum_{i=1}^{k}\left(\frac{\sum_{i=1}^{|C_i|}\alpha_i}{|C_i|}\right) \tag{7.11}$$

So total cost is calculated as follows

$$|D|*\left(\frac{1}{K}\sum_{i=1}^{k}\left(\frac{\sum_{i=1}^{|C_i|}\alpha_i}{|C_i|}\right)\right) \tag{7.12}$$

Next have to calculate the search complexity using Chained Hashed Table.

The search complexity per cluster is $O(1+\alpha_i)$

Then the average computation is calculated by

$$O\left(\frac{1}{K}\sum_{i=1}^{k}(O(1+\alpha_i))\right) \tag{7.13}$$

So the total computation required including indexing using hashing is

$$O\left(\frac{(|D|\ log\ |D|)+\left(|D|*\left(\frac{1}{K}\sum_{i=1}^{k}\left(\frac{\sum_{i=1}^{|C_i|}\alpha_i}{|C_i|}\right)\right)\right)+|D|+\left(\frac{1}{K}\sum_{i=1}^{k}(O(1+\alpha_i))\right)}{|D|}\right) \tag{7.14}$$

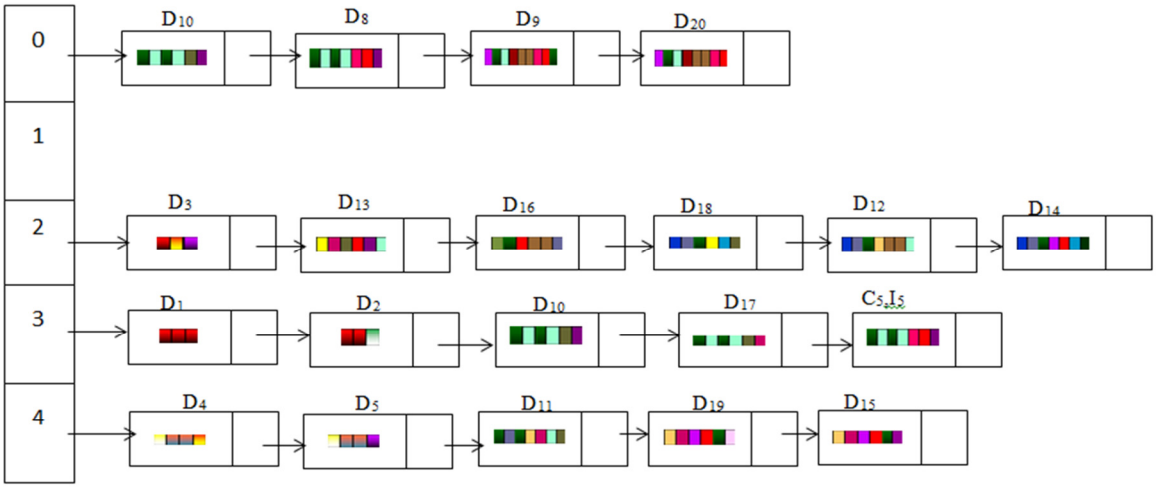


Figure 7.8 Chain Hash Table Before Clustering

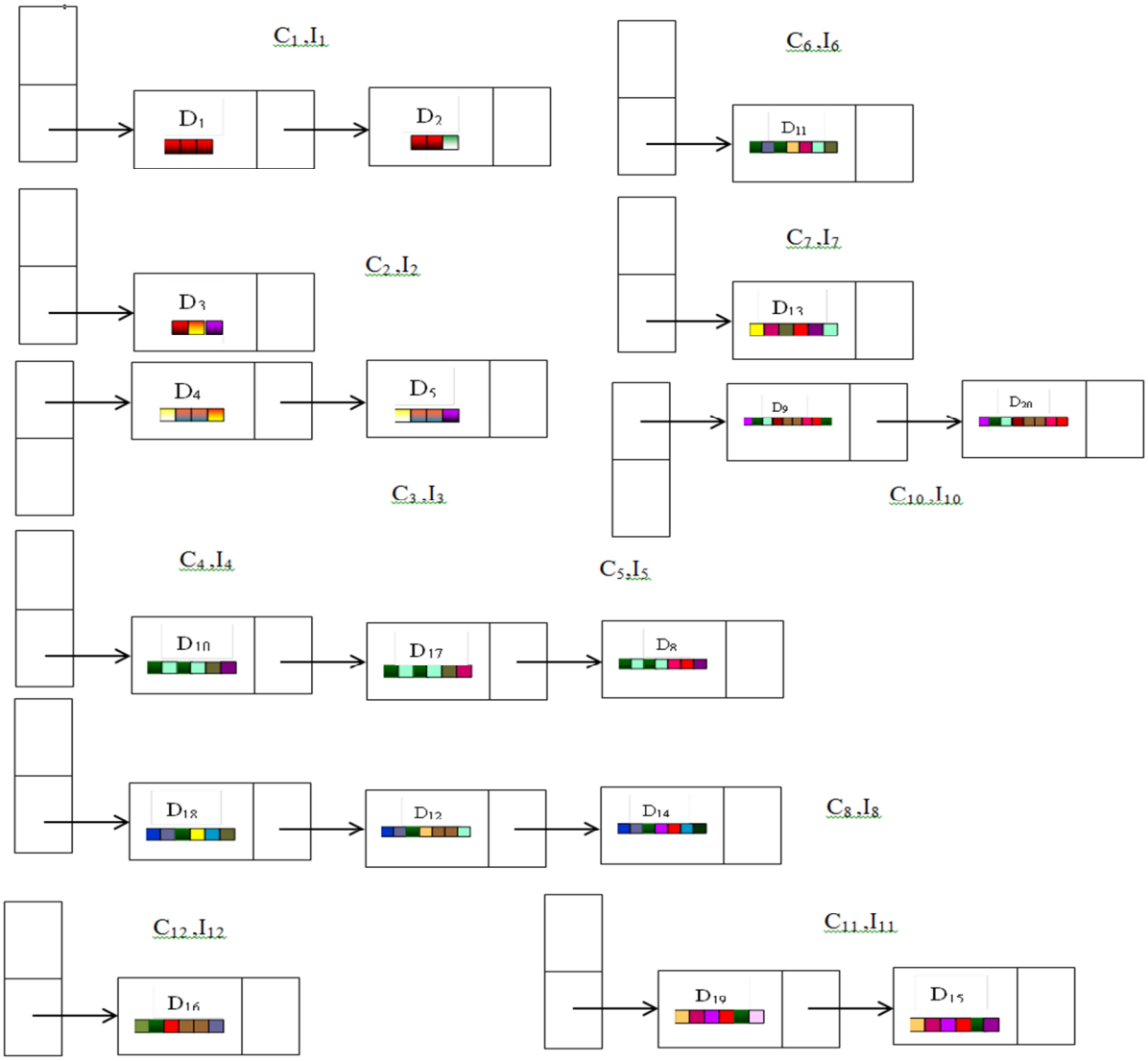


Figure 7.9 Chain Hash Table after Clustering

7.3.1.4 BIS IMPLEMENTATION OF IKNTN ALGORITHM

BIS is incorporated within the proposed IKNTN clustering algorithm implementation, the BIS is used with each cluster that is shown in the figure 7.8. Each time a new member is found in the D, the algorithm inserts the member or data point to the particular cluster that has the same twin neighbour, the clusters function with the help of BIS. Otherwise it will create a new cluster and store the data point or the member in that cluster. Each time a new cluster is identified in the data set, the algorithm automatically creates a new data structure for BIS. In this implementation each cluster is uniquely maintained with unique data structure. In this approach only the index table is maintained with hash table and clusters are maintained with BIS. The figure 7.10 shows the chained data structure before clustering and figure 7.11 shows the chained data structure employed in each cluster during IKNTN clustering algorithm

Theorem 7.4 BIS implementation of IKNTN algorithm takes

$$O(\frac{1}{|D|}\Bigg((|D|\ log\ |D|) + +(|D| * (\frac{1}{K}\sum_{i=1}^{k}(|C_i| * ((\log \prod_{i=1}^{|C_i|} i)^{\frac{1}{|C_i|}})))) + |D| + \left(\frac{1}{K}\sum_{i=1}^{k}(|C_i| \log |C_i|\right)\Bigg))\text{Computation per search}$$

Proof: After fetching each data points D_i during clustering phase single insertion is made to the corresponding cluster which shown in the figure 7.6.

Insertion to the BIS with BS approximately requires less than $O\ log(n)$ time and to fulfill single iteration approximately is calculated as based on the theorem 7.2.

$$O(\frac{1}{|D|}\Bigg((|D|\ log\ |D|) + +(|D| * (\frac{1}{K}\sum_{i=1}^{k}(|C_i| * ((\log \prod_{i=1}^{|C_i|} i)^{\frac{1}{|C_i|}})))) + |D| + \left(\frac{1}{K}\sum_{i=1}^{k}(|C_i| \log |C_i|\right)\Bigg)) \quad (7.15)$$

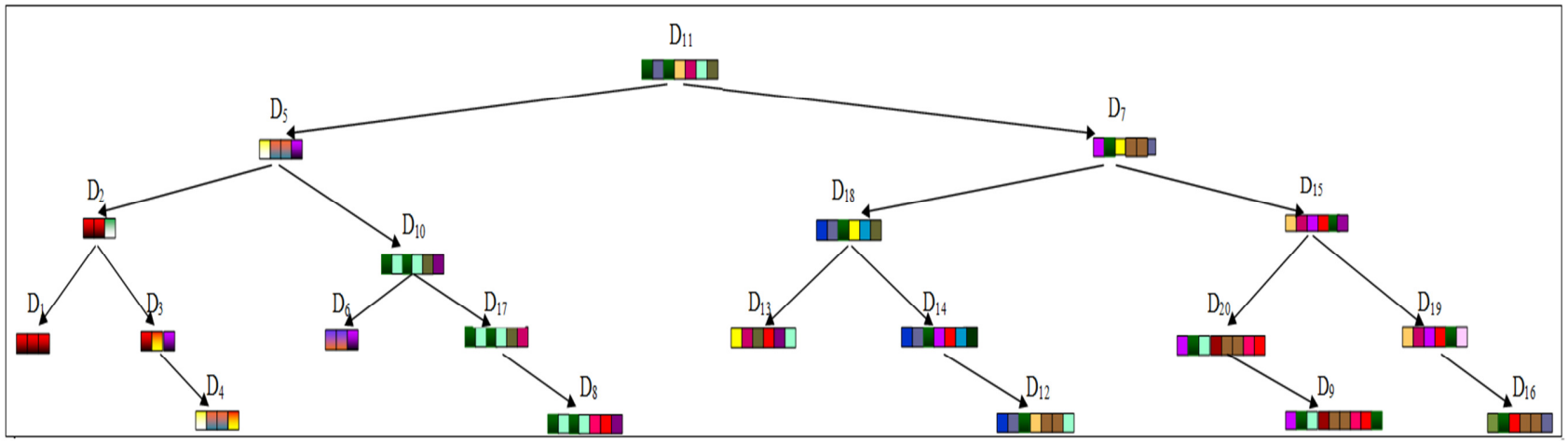


Figure 7.10 Illustrates BIS and Binary search

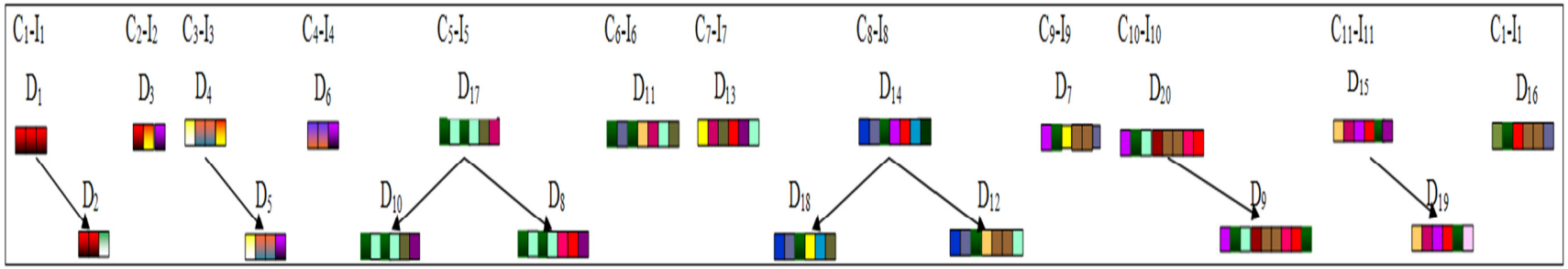


Figure 7.11 illustrates BIS with Binary search clustering using IKNTN algorithm

7.3.2 SPACE COMPLEXITY OF IKNTN ALGORITHM

Space complexity of the algorithm is calculated by the total usage of main memory by the execution time. The algorithm uses five variables such as $i, L, K, E, D,\ and\ In$, So the algorithm occupies one unit for each variable, for five variables it requires five units of memory and the data set D fully resides in the memory during the clustering. The size of the dataset is |D|. So the total space required is $|D| + 5$, the size of the index table is K. i.e. approximately $O(|D| + K)$, and the sorting phase and search phase require the additional variables. Then the clusters occupy the same size of memory utilized by the Data set D. Then total memory or space required is $O(2 * |D| + K)$.

7.4 EXPERIMENTAL RESULTS

The data strucures and algorithms which are used for the experimentation of the data structures specially employed with the LZW compression algorithm is discussed in the Chapter IV, they are Linear array with linear search , BST, Chained Hash table and BIS with Binary search. Each algorithm is exprerimented before and after clustering, for this expreimentation, name data set is used, and the size of the data set is 129090. The selected data structure and algorithm is executed 100 times to find out the Minimum time taken (Min(t)) and Maximum Time taken (Max(t)). The experimental results are shown in the figure 7.12 (before clustering) and figure 7.13 (after Clustering) and table 7.1. Linear search achives 99.68482, BST achieves 53.80868 (shown in the figure 7.14 (before clustering) and figure 7.15 (after clustering)); Hash table achieves 84.15043 (shown in the figure 7.16 (before clustering) and figure 7.17 (after clustering)) and BIS with Binary Search 64.87128 percentage of improvement after clustering is shown in the figure 7.18 (before clustering) and figure 7.19 (after clustering) with IKNTN algorithm. The experimentation results prove the ability of IKNTN algorithm in order to reduce the computational complexity of the existing data structure and algorithms. T is the total time required for search all elements in the dataset D. Where t_i is the time requiring for the search operation for the i^{th} element.

$$T = \sum_{i=1}^{|D|} t_i \tag{7.16}$$

Then the average time required for searching individual element is calculated as follows:

$$average\ time\ per\ search\ = \frac{T}{|D|} \tag{7.17}$$

The total time save is achieved after clustering with IKNTN for data structures and algorithms is shown in the figure 7.22, and the comparative analysis is shown in the figure 7.20 and figure 7.21.

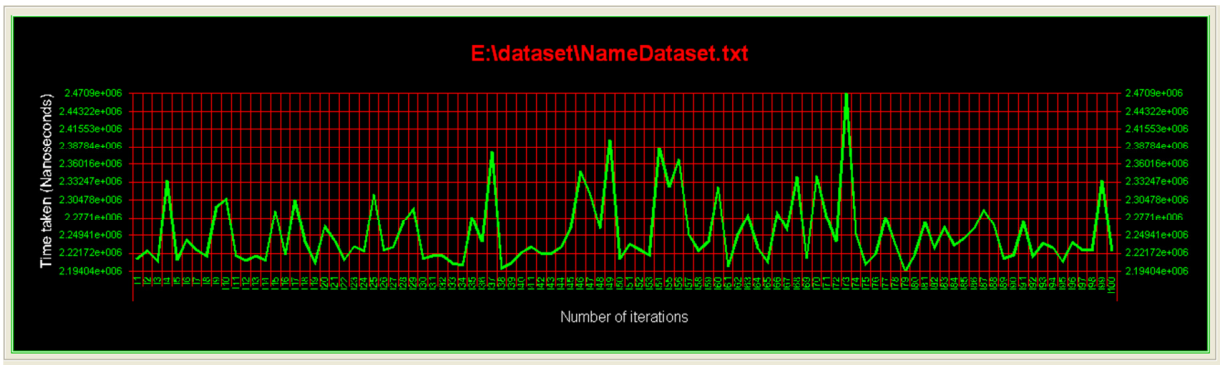


Figure 7.12 Time taken by Linear Array with Linear Search Before Clustering (in nanoseconds)

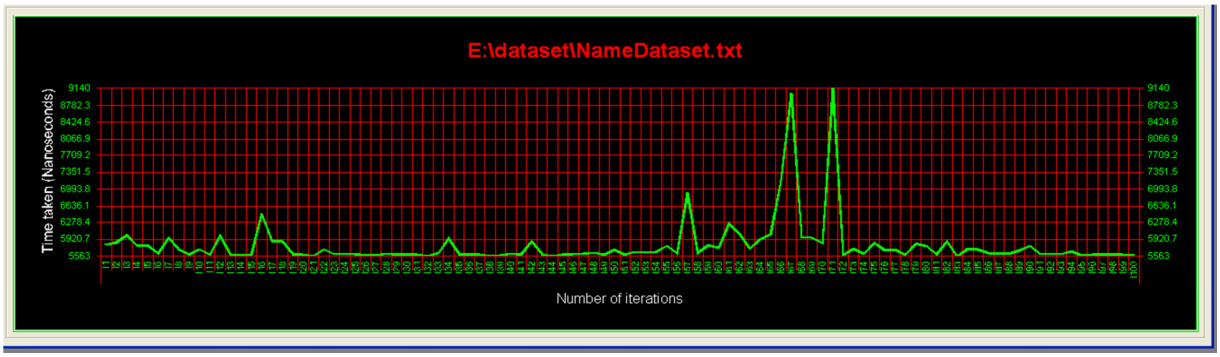


Figure 7.13 Time taken by Linear Array with Linear Search After Clustering (in nanoseconds)

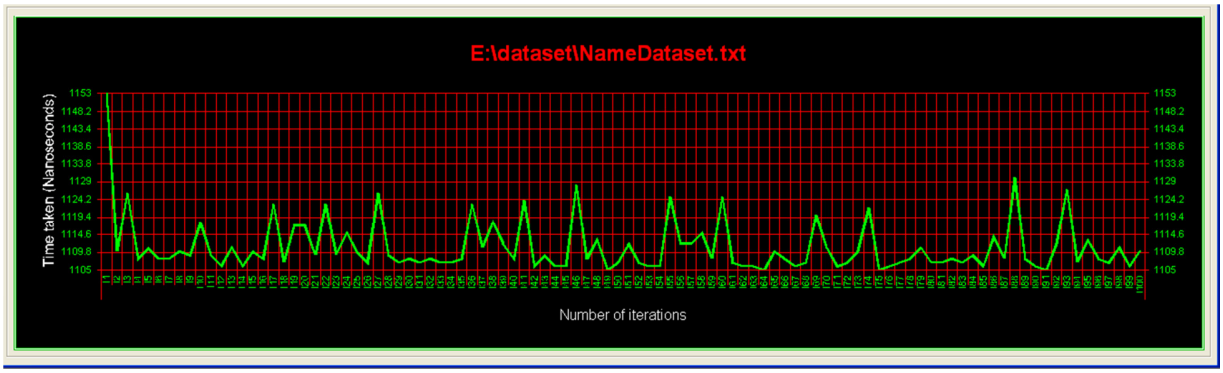


Figure 7.14 Time taken by BST Before Clustering (in nanoseconds)

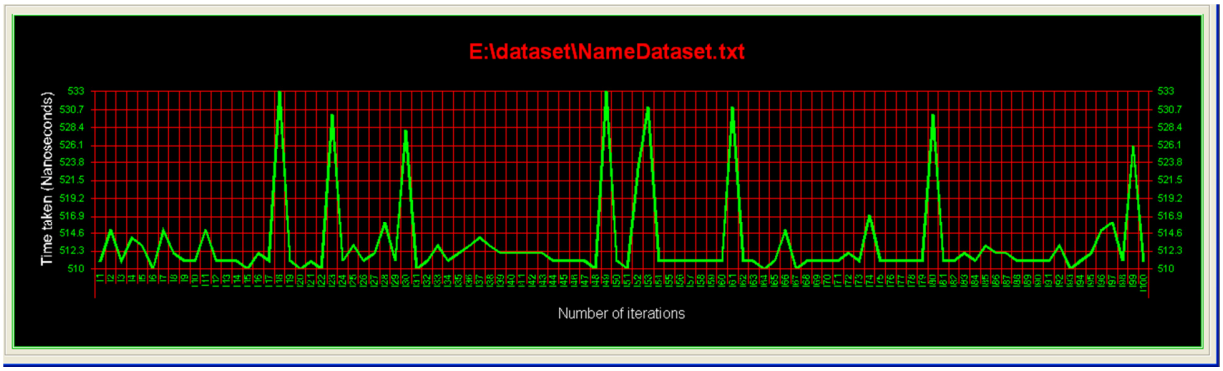


Figure 7.15 Time taken by BST after Clustering (in nanoseconds)

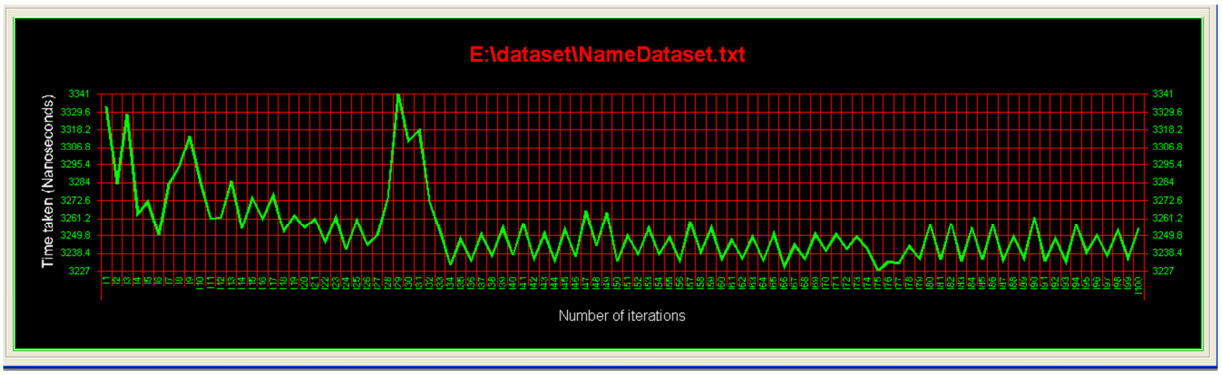


Figure 7.16 Time taken by Hash table before Clustering (in nanoseconds)

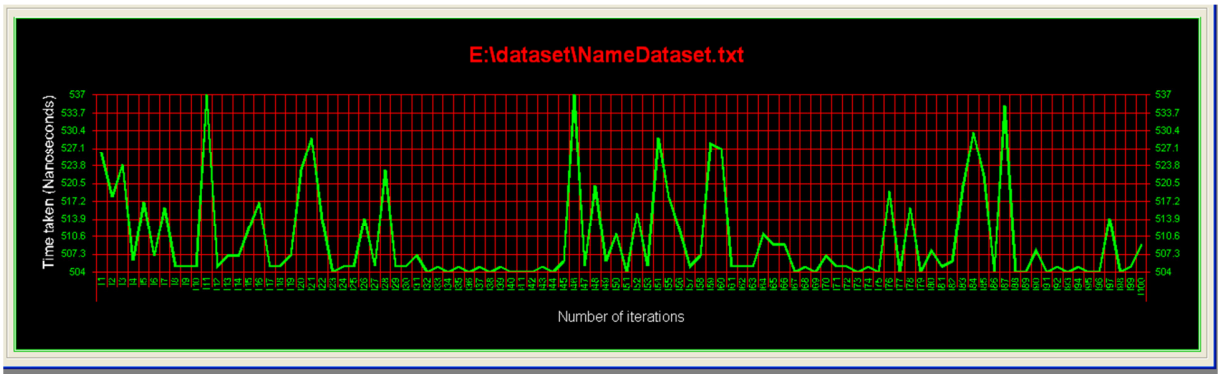


Figure 7.17 Time taken by Hash table after Clustering (in nanoseconds)

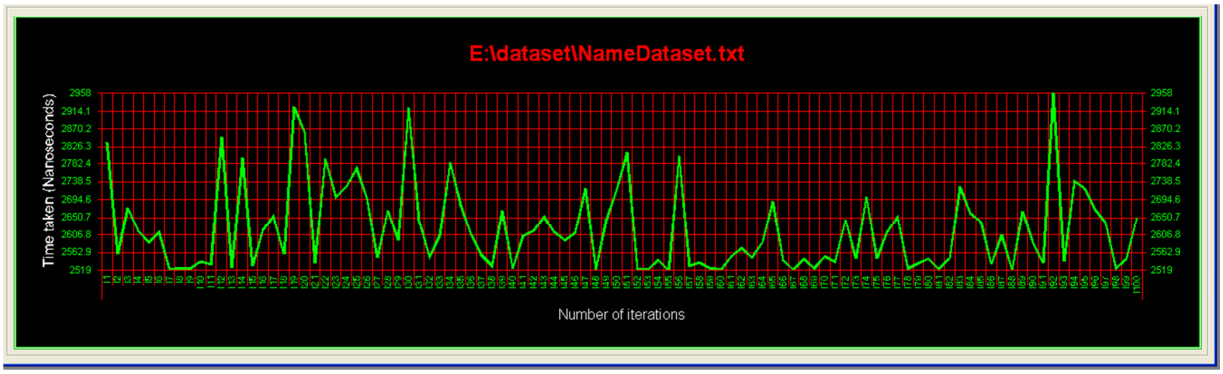


Figure 7.18 Time taken by BIS with Binary Search before Clustering (in nanoseconds)

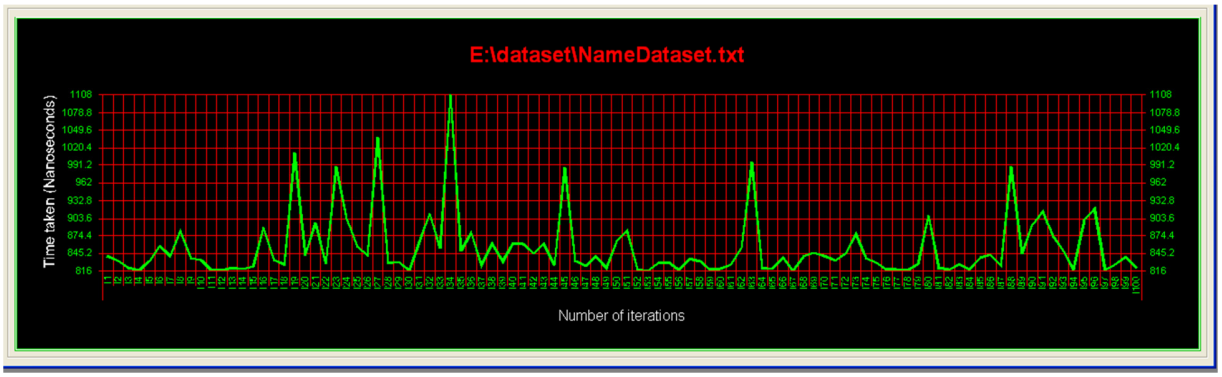


Figure 7.19 Time taken by BIS with Binary Search after Clustering (in nanoseconds)

Table 7.1 Time taken by Aata Structure and Algorithms Before and Afrter IKNTN Clustering Algorithm

Data Structures or algorithm used for experimentation		**Min(t)**	**Max(t)**
Linear array with Linear Search	Before Clustering	2194035	2470902
	After clustering using IKNTN	5563	9140
BST	Before Clustering	1105	1153
	After clustering using IKNTN	510	533
Hash Table	Before Clustering	3227	3341
	After clustering using IKNTN	504	537
BIS with Binary Search	Before Clustering	2519	2958
	After clustering using IKNTN	816	1108

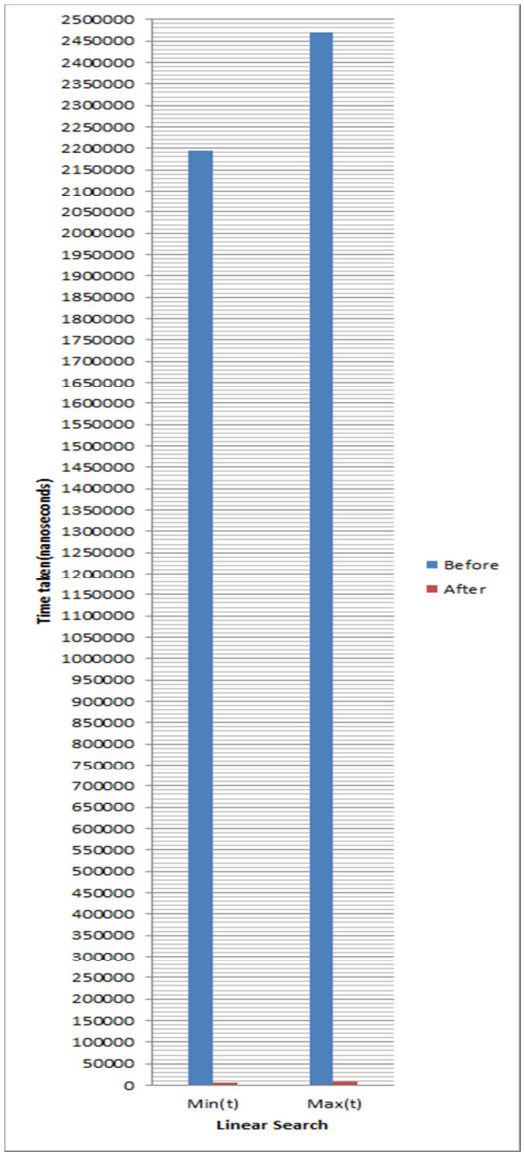


Figure 7.20 Time complexity analysis of data structure and algorithms before and afrter IKNTN

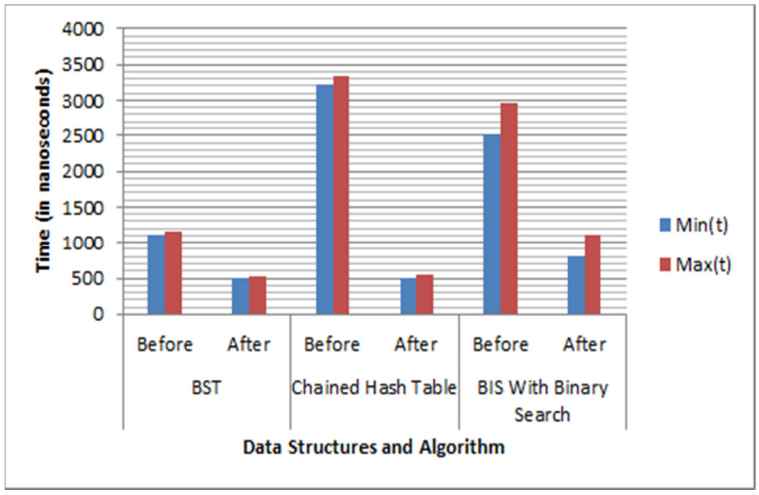


Figure 7.21 Time complexity analysis of data structure and algorithms before and afrter IKNTN clustering Algorithm

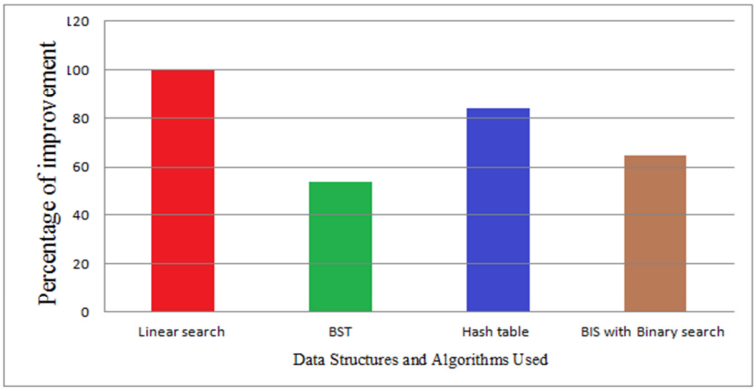


Figure 7.22 Data structures and algorithms time reducion after IKNTN clustering Algorithm (in Percentage)

7.5 SUMMARY

A novel Clustering algorithm which is proposed in this chapter is called Indexed K Nearest Twin Neighbour (IKNTN) Clustering algorithm. The proposed algorithm effctively clusteres all the possible Nearest Twin Neighbour with the help of Euclidean distance and each cluster is uniquily indexed using a novel indexing method. The algorithm is specially designed to optimize the perfomence of data structure. Data structure and algorithm which are experimented with the IKNTN clustering algorithms are Linear array with Linear search, BST, Chained Hash Table and BIS with Binary Search. The experimentation result shows 99.68482, 53.80868, 84.15043 and 64.87128 percentages of time complexity reduction on Linear array with Linear search, BST chained Hash table achieves and BIS with Binary search respectively. The experimentation result shows the effectiveness and efficiency of the IKNTN clustering algorithm. So the proposed IKNTN algorithm is the best for optimizing any kind of search and the performance of data structure. The computational complexity of data structures and algorithms is reduced after clustering with IKNTN algorithm. The next phases of this thesis use the IKNTN Clustering algorithm to reduce the time complexity of LZW data compression algorithm.