CHAPTER 4

VORONOI DIAGRAM BASED CLUSTERING ALGORITHMS

4.1 Introduction

Although MST-based clustering methods are effective for complex data, they require quadratic computational time which is high for large number of data points. Hence, we have researched on another neighborhood graph called Voronoi diagram that has less computational cost over MST. In this chapter, we propose three clustering algorithms using the features of Voronoi edges, vertices and circles. Experiments carried out on various two-dimensional synthetic and multi-dimensional biological data are compared with few existing techniques to show the effectiveness of the three proposed algorithms.

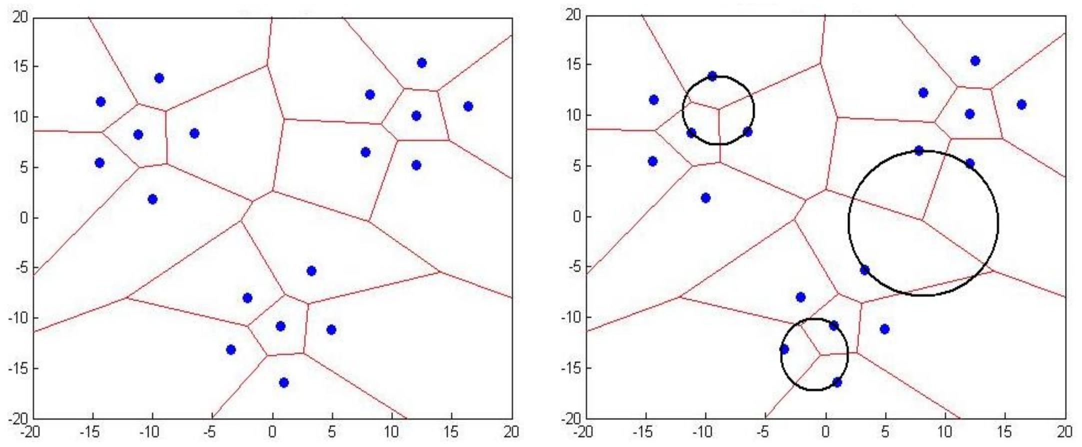
4.2 A Few Terms

The following terminologies are used in the proposed algorithms.

4.2.1 Voronoi Diagram

Given a set of points, Voronoi diagram [34] is a partition of the space into cells, each of which consists of the points closer to one particular object than to any others. It is formally defined as follows. Let $S = \{p_1, p_2,\ldots, p_n\}$ be a set of n points in a d-dimensional Euclidean space and $d(a, b)$ denote the distance between the points a and b in this space. Then the Voronoi diagram of S (see Figure 4.1(a)) is defined as the subdivision of the space into n cells, one for each point in S. A point u lies in the cell corresponding to the point p_i iff $d(u, p_i) < d(u, p_j)$ for each $p_j \in S$ and $j \neq i$. We denote the Voronoi diagram of S by $Vor(S)$ and the cell corresponding to the point p_i by

$V(p_i)$. We call the vertices of a Voronoi diagram as Voronoi vertices. There are maximum $2n$-5 Voronoi vertices in a Voronoi diagram of n points. It is obvious from the definition that for each point $p_i \in S$, $V(p_i)$ contains all the points that are closer to p_i than to other points of S. For a Voronoi vertex v, we define the Voronoi circle of v (see Figure 4.1(b)) with respect to S, as the largest circle with v as its center and contain no point of S in its interior. We denote this circle by $CirS(v)$. The Voronoi vertices have the property that a point q is a vertex of $Vor(S)$ iff $CirS(q)$ contains three or more points of S on its boundary. A point p is said to be covered by a vertex q if and only if p lies on the boundary of $CirS(q)$.



(a) (b)

Figure 4.1: (a) Voronoi diagram of given points; (b) Voronoi circles

4.2.2 Dunn Validity Index

In this chapter, we use the validity index proposed by Dunn as follows. For a specific number of clusters nc, the Dunn validity index (DUVI) [157] is defined as the following.

$$DUVI = \min_{i=1,\ldots,nc}\left\{\min_{j=i+1,\ldots,nc}\left(\frac{d(c_i,c_j)}{\max_{k=1,\ldots,nc} diam(c_k)}\right)\right\} \tag{4.1}$$

$d(c_i, c_j)$ is the dissimilarity function between two clusters c_i and c_j. It is defined as

$$d(c_i, c_j) = \min_{x \in c_i,\, y \in c_j} d(x, y) \tag{4.2}$$

and the $diam(c_k)$ is the diameter of the kth cluster, which may be considered as a measure of dispersion of the clusters which is defined as follows.

$$diam(c_k) = \max_{x, y \in c_k} d(x, y) \tag{4.3}$$

By plotting this validity index against the cluster number, the optimal number of clusters can be obtained when the Dunn index reaches the maximum as the main goal of this measure is to maximize the inter-cluster distances and minimize the intra-cluster distances. Therefore, the number of clusters that maximizes DUVI is taken as the optimal number of the clusters.

4.3 Proposed Algorithms

Here, we present the proposed algorithms as follows.

4.3.1 Algorithm Based on Voronoi Edges

The main scheme behind this technique is as follows. We initially construct the Voronoi diagram of the given n points. Then all the Voronoi vertices $v_1, v_2, \ldots, v_{2n-5}$ of the constructed Voronoi diagram are identified. We start with a Voronoi vertex v whose circle radius is the smallest one. The cluster formation is done as follows. The first cluster is initially composed of the points that lie on the circumference of v. Further, the points of this cluster are located by finding the directly connected vertices of v (say v' and v'') which are at a distance less than a threshold value μ. Ignore the vertices which are at greater than μ distance from v. Now, the points on the circumference of $v'$and v'' are added to the same cluster. The process is repeated for all the newly identified vertices until no vertex is found within μ distance. It may happen that some visited vertices may be encountered more than once during the iterations. Then, we skip such vertices. When there is no vertex found within μ distance, we check whether all the given points are assigned to the clusters. If all the points do not take part to form the clusters, it indicates the presence of more clusters. Hence, we resume with any non-visited Voronoi vertex and repeat the same procedure. The process is terminated when all the given points are assigned to the clusters.

4.3.2 Problem of Redundancy

It can be noted that the points in the Voronoi diagram are on the circumference of two or more Voronoi circles as shown in the Figure 4.2. Since we use the Voronoi vertices to form the clusters, it leads to the redundancy of points in the clusters as follows. Suppose the points on the circumference of a Voronoi circle $CirS(v_i)$ are assigned to a cluster C_i. If these points are on the circumference of another Voronoi circle $CirS(v_j)$ where the distance $d(v_i, v_j)$ is less than μ, then such points will be assigned to the same cluster again. This way few points may be assigned to some clusters multiple times.

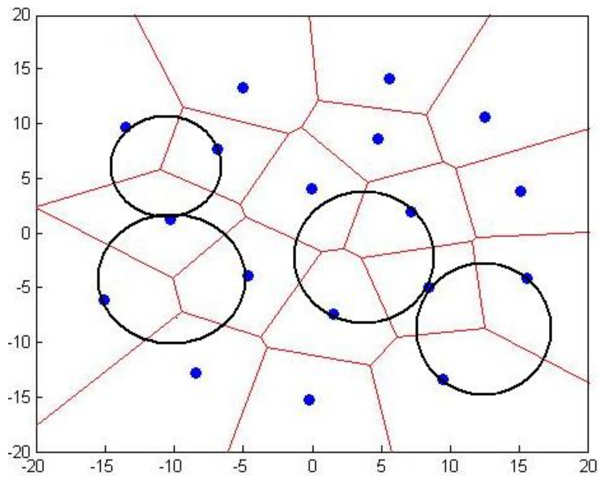


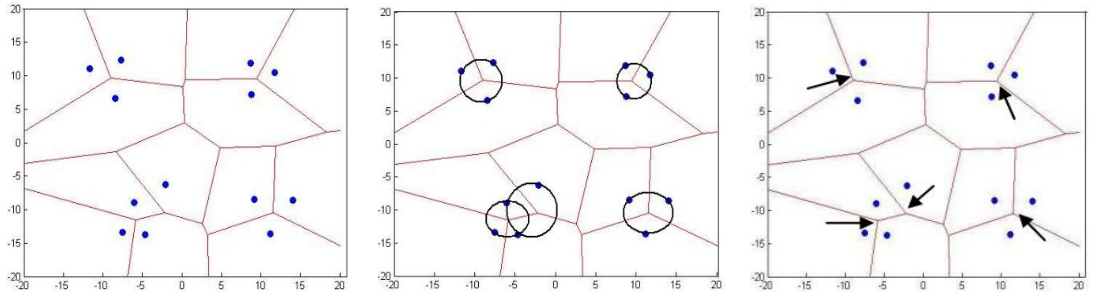
Figure 4.2: Points on the circumference of more than one Voronoi circle

One way to overcome this problem is to find the closest Voronoi vertices for each of the given data points. These vertices are labeled as the representative points of the given data points which are on the circumferences of the corresponding Voroni circles. The efficient way to find the closer Voronoi vertex of each and every given point is to sort the Voronoi vertices with respect to the radius of its largest empty Voronoi circles. Let v_i be the vertex with least radius Voronoi circle $CirS(v_i)$ among $v_1, v_2, \ldots, v_{2n\text{-}5}$ where $1 \leq i \leq 2n\text{-}5$. Then the closer vertex for the points on the circumference of $CirS(v_i)$ is obviously the vertex v_i. The next least radius Voronoi vertex v_j is the closer Voronoi vertex for the points on the circumference of $CirS(v_j)$. Similarly the closer Voronoi vertices for all the other points can be traced out by considering the next Voronoi vertex from the sorted list. It is not necessary to repeat the process for all the vertices of the sorted list. After certain stage, it is easy to observe that all the given n points are covered by few vertices out of $2n$-5 (maximum). For the sake of convenience, we call these vertices as non-empty vertices and the remaining vertices are empty vertices. It is important to note that only non-empty Voronoi vertices hold the points on its boundary where as the empty Voronoi vertices do not represent any of the given points. The empty and non-empty Voronoi vertices are briefly defined as follows.

Definition **(Non-empty Voronoi vertex):** Let V={$v_1, v_2, \ldots, v_{(2n\text{-}5)}$ (max.)} be the set of Voronoi vertices in the Voronoi diagram of n points. A Voronoi vertex v_i is non-empty if the distance of v_i from at least one of the given n data points is less than the distance of such point from all the remaining Voronoi vertices. i.e. a vertex v_i is non-empty iff there exist at least a point p of the given data set $\ni d(p, v_i) < d(p, v_j) \; \forall \; v_j \in V - \{v_i\}$, where V is the set of all Voronoi vertices.

Definition **(Empty Voronoi vertex):** Let the set $V = \{v_1, v_2, \ldots, v_{(2n\text{-}5)}$ (max.)} be the set of all Voronoi vertices in the Voronoi diagram of n points and NE be the set of non-empty Voronoi vertices defined as above. Then the vertices belong to the set $V - NE$ are the empty vertices.

The empty and non-empty Voronoi vertices are described in the Figures 4.3(a-c). Here, all the given points are represented by closer Voronoi vertices with respect to the radii of the Voronoi circles. We call these vertices (shown by arrows) as non-empty Voronoi vertices and all the other vertices are empty Voronoi vertices.



(a) (b) (c)

Figure 4.3: Illustration of empty and non-empty vertices in a Voronoi diagram of 13 points. (a) *Vor*(*S*); (b) Voronoi circles of closer Voronoi vertices for all the points; (c) Non-Empty Voronoi vertices shown by arrows

The main task in the proposed method is to group the points on the circumferences of non-empty Voronoi vertices which are directly connected to either empty or non-empty vertices by a Voronoi edge at a distance less than or equal to μ. We store the already visited Voronoi vertices in a temporary set to avoid the problem of redundancy. The Proposed algorithm is terminated when all the points on the circumferences of non-empty Voronoi vertices are distributed to the clusters.

4.3.3 Problem of Infinite Edges

The important task in the proposed method is to find the vertices connected by an edge. In general, a Voronoi vertex is the intersection point of the Voronoi edges. When a vertex is made by the intersection of one or more infinite Voronoi edges, it is complex to find out the vertices directly connected by such an edge. In order to solve the problem of infinite edges, initially, we locate the intersected area of the Voronoi diagram with a d – dimensional cube whose extrema are defined as the minimum and the maximum attribute values in each dimension as shown in the Figure 4.4.

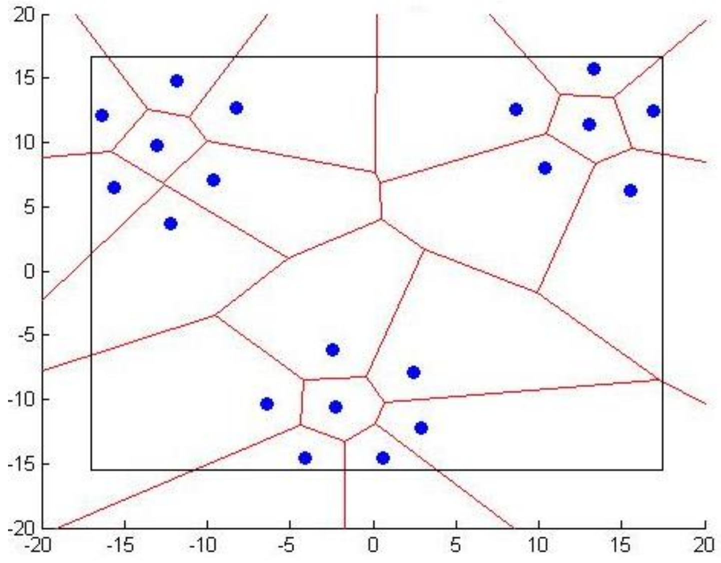
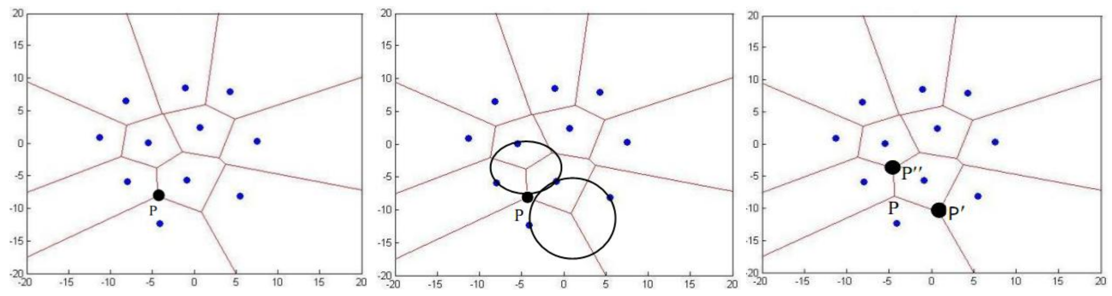


Figure 4.4: Intersection of the Voronoi diagram of given data with a cube

Now, the directly connected vertices can be found using the Voronoi edges bounded by this intersected region. Let v be the Voronoi vertex which is created by the intersection of one or more infinite edges. To find the vertices v_i that are connected to v by the direct Voronoi edge, we initially find the points (say p_1, p_2 and p_3) that are on the boundary of $CirS(v)$. We now find the Voronoi vertices on whose circle boundary any two of the points p_1, p_2 and p_3 are located. All such vertices are nothing but the required vertices which are directly connected to v by a Voronoi edge. This way of finding the vertices directly connected to a vertex is depicted by the Figures 4.5(a–c). Here, we find the Voronoi vertices that are directly connected to the vertex P (shown in Figure 4.5(a)) by an edge. For that, the points on the boundary of the Voronoi circle of P are traced out and examined the circles which contain any two of these points on their circumference. The centers of those circles are nothing but the Voronoi vertices that are directly connected to P by an edge. From the Figure 4.5(b), we can observe that the Voronoi circles of the vertices P' and P'' have two of the points of p_1, p_2 and p_3 on their boundaries. Therefore the desired Voronoi vertices connected to P by a direct Voronoi edge are P' and P'' (shown in Figure 4.5(c)).



(a) (b) (c)

Figure 4.5: Illustration of finding directly connected vertices (a) a vertex *P* to find the directly connected vertices; (b) Voronoi circles of the desired vertices; (c) Located vertices *P′* and *P″* connected to *P*

We now present the pseudo code of the proposed algorithm as follows.

Algorithm-GCVD (*S*, *μ*)

Input: A set *S* of *n* data points and a threshold value *μ*

Output: A set of clusters

Functions and variables used:

Vor (*S*): A function when called constructs the Voronoi diagram for the given data set *S* of *n* points.

connected_vertices(*v*): A function to find all the Voronoi vertices directly connected to the Voronoi vertex *v*.

V[]: An array to represent all the Voronoi vertices.

NEV[]: An array to represent non-empty Voronoi vertices.

EV[]: An array to represent empty Voronoi vertices.

VV[]: An array to store the visited Voronoi vertices (It is empty initially).

C_l[]: An array to represent the cluster *i*.

G_l[]: An array to represent *i*th group of Voronoi vertices.

Temp[]: A temporary array.

i, *j*, *k*, *l*: Temporary variables. /* Initially *i*, *j*, *k* are NULL & *l* =1 */

```
Algorithm_GCVD($S$, $\mu$)
{
Step 1: Call Vor($S$);
Step 2: Sort all the Voronoi vertices $v_i$ in ascending order with respect to the radius of
        their largest empty Voronoi circle's $CirS(v_i)$, $i = 1, 2, \ldots, 2n-5$ (max.) and store
        them in an array $V$.
                /* Phase to find the empty & non-empty vertices */
Step 3: While ( $EV = \phi$ )
        {
          Locate the points $P_k$, where $k = 1, 2, \ldots, j$ ($j \geq 3$) on the boundary of $CirS(v_i)$
          where $v_i$ is the $i$ th element of the array $V$.
          $NEV \leftarrow NEV \cup \{v_i\}$;
          For $k = 1$ to $j$
              {
                If ($P_k \notin$ Temp)
                Temp $\leftarrow$ Temp $\cup$ { $P_k$ };
              }
          If (($|Temp|$) == $n$) then
              {
                $EV \leftarrow V - NEV$;
                Break;
              }
          Else
                $i \leftarrow i + 1$;
        }
Step 4: $i, j, k \leftarrow \phi$
Step 5: Select any one vertex $v$ from the set $NEV$ and store it in the group $G_l$ and the
        set $VV$;
                /* Formation of $K$ (say) groups of non-empty vertices*/
Step 6: Call connected_vertices($v$);
Step 7: If $|VV| \neq |NEV|$ then
            {
              $l = l + 1$;
              Go to step 6 with a vertex $v \in NEV - VV$;
            }
Step 8:  $l \leftarrow 1$;
            /* Formation of $K$ clusters from the above $K$ groups */
Step 9: While ($l \leq K$)
                {
                  Find the points $P_i$ (for some $i$) that are on the circumferences
                  of $CirS(v_j)$ where $v_j \in G_l$.
                  Store the points in the cluster $C_l$ i.e. $C_l \leftarrow P_i$;
                  $l \leftarrow l + 1$;
                }
Step 10: Exit ();
}
```

□

```
Function connected_vertices(v)
{
Step 1: Find the points $P_1$, $P_2$ and $P_3$ that are on the boundary of $CirS(v)$;
Step 2: Locate the vertices $v_i$, for some $i = 1, 2,\ldots, m \ni$ any two of $P_1$, $P_2$ and $P_3$ lie on
        the boundary of $CirS(v_i)$;
Step 3: Find $d(v, v_i)\ \forall\ i = 1, 2,\ldots, m$;
Step 4: For $i = 1, 2,\ldots, m$
        {
            If $d(v, v_i) \leq \mu$ then
                {
                    If ($v_i \notin VV[]$ && $v_i \in NEV$) then
                        {
```

```
                        $G_l \leftarrow G_l \cup \{v_i\}$;
                        $VV \leftarrow VV \cup \{v_i\}$;
                        Call **connected_vertices($v_i$);**
                    }
            }
    }
}
```

Time Complexity: The Voronoi diagram is constructed in $O(n \log n)$ time. Then it requires $O(n \log n)$ time to sort all the $2n$-5 Voronoi vertices with respect to their radii. The vertices in the Voronoi diagram are examined for the redundancy condition. This condition is verified till all the points are distributed to the clusters. This phase runs in $O(pn)$ time as there are $2n$-5 (maximum) Voronoi vertices, where p is the number of vertices examined for the redundancy. It can be noted that $p < 2n$-5 as we skip the vertices which are at greater than the threshold distance. We stop checking the redundancy condition when all the clusters are fully formed. Hence, the time complexity of the proposed algorithm is maximum of $\{O(n \log n), O(pn)\}$.

4.3.4 Outlier Detection

The outlier detection phase is performed after formation of the clusters by the algorithm GCVD(S, μ). The proposed algorithm locates the outliers in the form of separate clusters depending on the chosen threshold value μ. Then the clusters with considerably small number of points are declared as outliers by imposing a limit on the cardinality of the clusters produced by GCVD. For example, in the Figure 4.6(a), the outliers (shown in the boxes) are represented by two non-empty Voronoi vertices (i.e., these outlier points lie on the circumference of the circles of two non-empty vertices) which are not connected to any of the other non-empty vertex. Hence, when chosen an efficient value of μ, the two outlier regions are separated from the other regions to represent the clusters. Then we put a limit on the cardinality of each cluster to identify the outliers hidden in the form of clusters. We tune the limit on the cardinality of the clusters as follows. We first find the *min_limit* for any cluster cardinality as $n/(2*k)$, where k is the number of clusters produced by the GCVD and n is the given number of points. Then we find all the clusters whose cardinality $|C_k|$ is < *min_limit.* Now, the points of such cluster are declared as the outliers. Based on this idea, the two small clusters shown in the Figure 4.6(b) are declared as outliers.

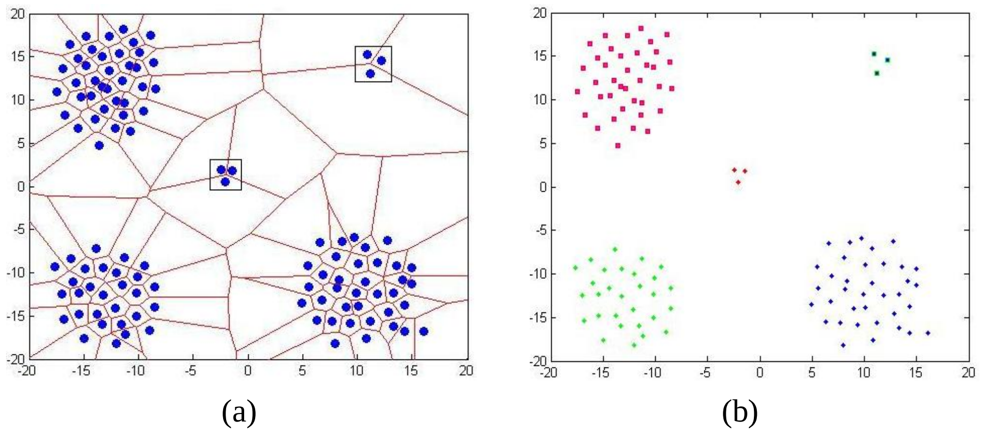


Figure 4.6: Outlier detection with *min_limit* (a) outliers (shown in boxes) represented by the non-empty Voronoi vertices far away from other non-empty Voronoi vertices; (b) outliers after separated using *min_limit*

4.3.5 Experimental Results

Initially, the proposed method was experimented on six artificially generated data sets namely, crescent and full moon, cluster inside cluster, five-group, L-shape, four-group and spiral. At the same time the existing methods *K*-means [26], FCM [158], CTVN [76] and SC [152] are also applied on the same data sets. The comparison results of the proposed method with the existing techniques are as follows. Initially the proposed GCVD method was applied on moon data of size 1500. It has produced the most wanted clusters in the form of two full moons and a half moon as shown in the Figure 4.7(f). At the same time none of the other methods produced the same result which can easily be seen from the Figures 4.7(b-e). We then applied the proposed GCVD algorithm on the cluster-inside-cluster data of size 1469. As it is clearly observed from the Figure 4.8(f), our method detects three clusters where two of them lie inside the third one. When applied, the other methods on the same data, it has resulted overlapped clusters as illustrated by the Figures 4.8(b-e). Then the proposed method was tested on the five-group data which has four non-convex clusters and one convex cluster. It successfully produced five clusters in the desired fashion as visible from the Figure 4.9(f). However, when applied, none of the methods of *K*-means, FCM, CTVN and SC results the similar clusters which is depicted from the Figures 4.9(b-e). After this, the proposed algorithm was run on L-shaped data of 1216 points. This data has few outlier points. The proposed method is able to group the points of each L-shaped class into a cluster. All the four clusters are produced and the outliers are also located as discussed in section 4.3.4. It can be seen from the Figure 4.10(f). However, the other methods are unable to produce the required result as it can easily be seen from the Figures 4.10(b-e). Next, the algorithm GCVD was experimented on four-group data of size 2036. The GCVD successfully produced four proper clusters as observed from the Figure 4.11(f). On the other side, none of the methods *K*-means, FCM, CTVN and SC produced the similar result which is clearly shown in the Figures 4.11(b-e). Finally the proposed GCVD method was tested on the spiral data with outliers. It produced three clusters where each cluster represents a spiral as shown in the Figure 4.12(f). Here, the outlier points are also located as shown from the same Figure. Clearly the GCVD outperforms *K*-means, FCM, CTVN and SC as visible from the Figures 4.12(b-e).

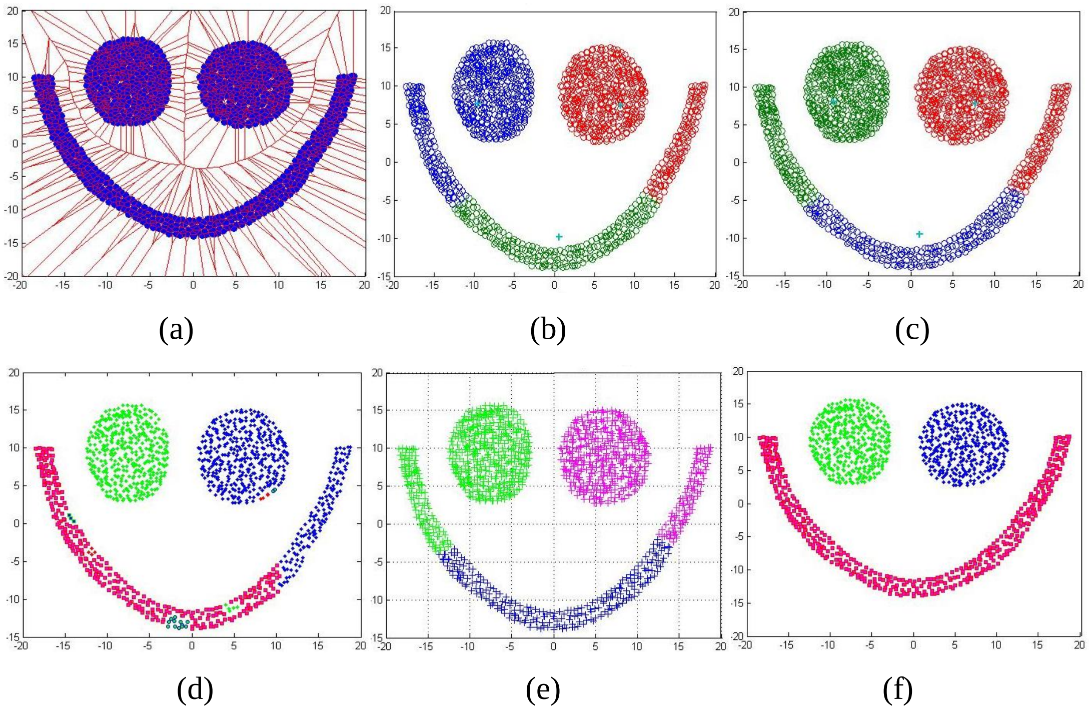


Figure 4.7: Result of crescent and full moon data of 1500 points. (a) initial Voronoi diagram; (b) *K*-means; (c) FCM; (d) CTVN; (e) SC; (f) proposed method

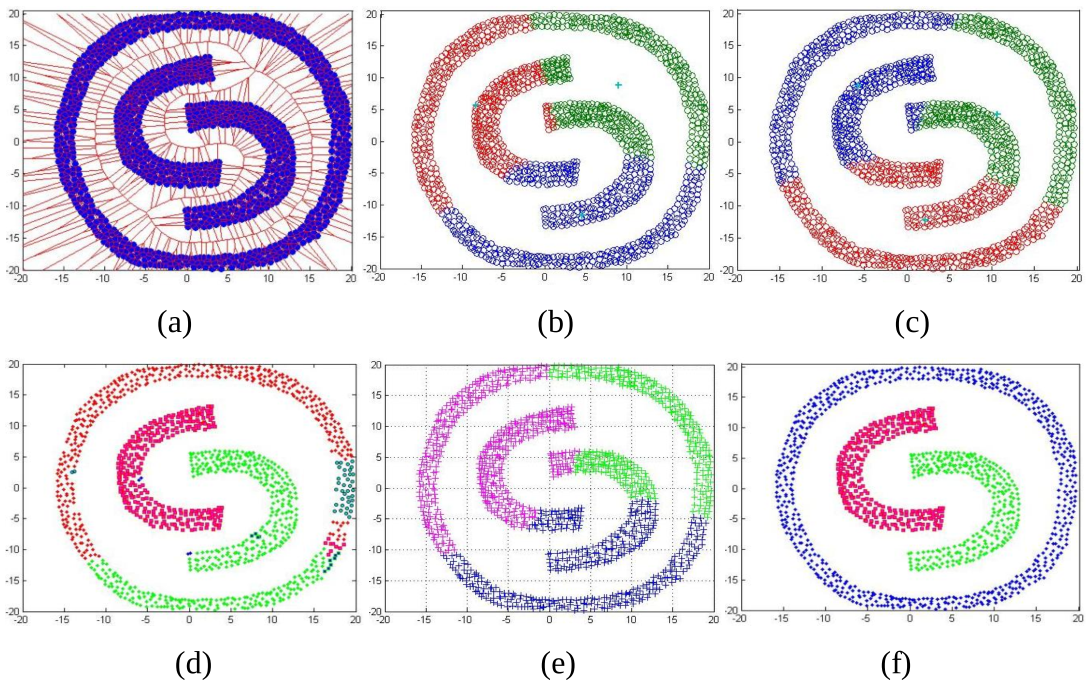


Figure 4.8: Result of cluster-inside-cluster data of 1469 points. (a) initial Voronoi diagram; (b) *K*-means; (c) FCM; (d) CTVN; (e) SC; (f) proposed method

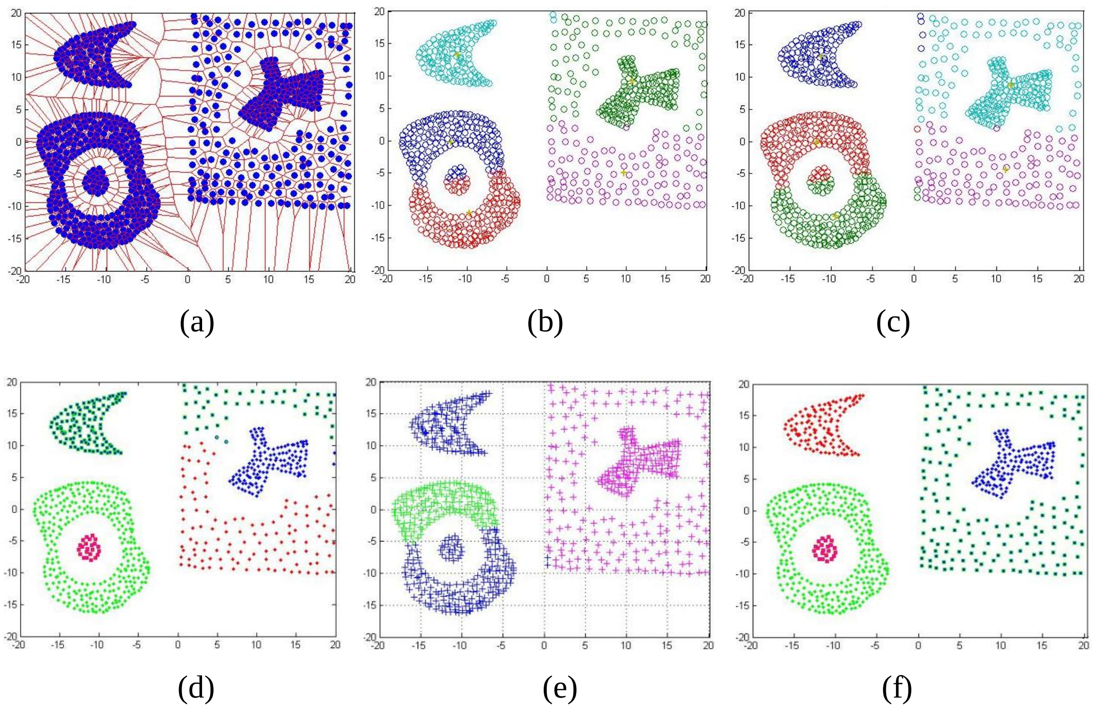


Figure 4.9: Result of five-group data of 760 points. (a) initial Voronoi diagram; (b) *K*-means; (c) FCM; (d) CTVN; (e) SC; (f) proposed method

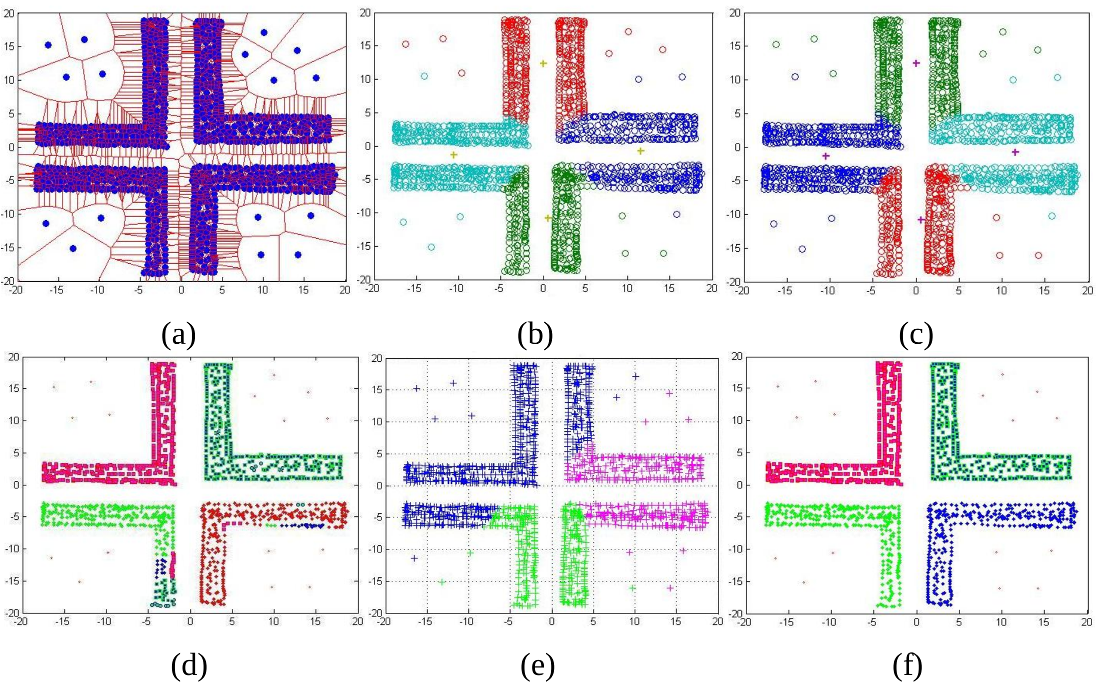


Figure 4.10: Result of L-shape data of 1216 points. (a) initial Voronoi diagram; (b) *K*-means; (c) FCM; (d) CTVN; (e) SC; (f) proposed method

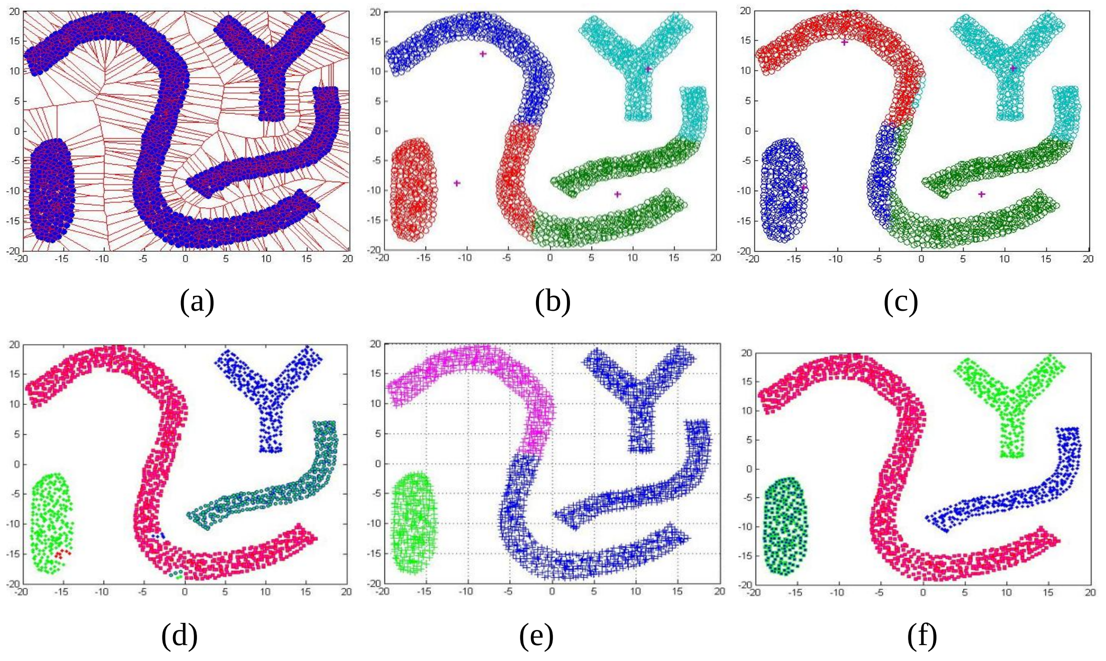


Figure 4.11: Result of four-group data of 2036 points. (a) initial Voronoi diagram; (b) *K*-means; (c) FCM; (d) CTVN; (e) SC; (f) proposed method

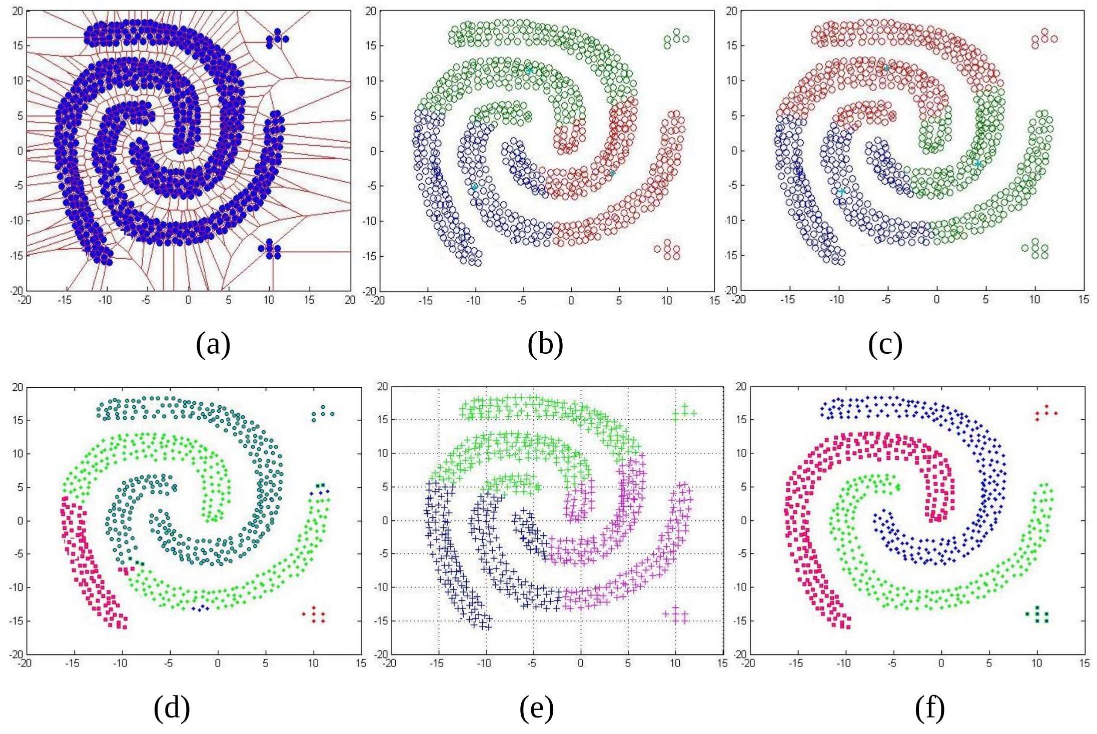


Figure 4.12: Result of spiral data of 611 points. (a) initial Voronoi diagram; (b) *K*-means; (c) FCM; (d) CTVN; (e) SC; (f) proposed method

Then, the proposed algorithm was applied on five gene expression data sets, namely, Yeast cell-cycle, Sporulation, Lymphoma, Diauxic and Fibroblasts [33]. We used Dunn validity index (DUVI) to observe the quality of clusters of gene expression data. For the fair evaluation, without any loss of generality, the number of clusters was fixed at 256 for all the data sets similar to that of Du et al., [87]. The comparison results of the proposed scheme with *K*-means, FCM, CTVN and SC by means of DUVI are as follows. The values of DUVI (discussed in section 4.2.2) were calculated for the clusters generated by the proposed method, *K*-means, FCM, CTVN and SC for all the gene expression data sets. We can notice from the definition that high value of the DUVI indicates more quality of the clusters. It can easily be noted from the Table 4.1 that the proposed method attains superior values over the existing methods.

Table 4.1: Results of gene expression data using Dunn validity index (DUVI)

Gene Expression Data	**No. of Attributes**	**Data Size**	**DUVI**				
			***K*-means**	**FCM**	**CTVN**	**SC**	**Proposed**
Yeast cell-cycle	77	5571	234.5363	198.8474	249.0473	280.7656	**310.2743**
Sporulation	7	6039	176.9273	210.6453	245.9383	198.2889	**279.7354**
Lymphoma	96	4026	145.8545	121.3242	80.2645	214.4185	**315.5536**
Diauxic	7	6100	243.9752	165.0534	221.4276	341.5634	**389.0756**
Fibroblasts	18	501	120.4561	167.9363	135.9835	132.5648	**276.9474**

4.3.6 Algorithm Based on Voronoi Circles

The central theme of this algorithm is as follows. Given a set S of n data points and a value k of required number of clusters. Initially, the Voronoi diagram of the given n points is constructed. We assume that all the given points represent separate sub-clusters. These sub-clusters are further merged as follows. We define a function f which maps from S to R. The value of $f(P)$ where $P \in S$, is defined as the sum of radii of all the circles which contain P on their circumferences. All the given points are sorted in monotonically increasing order with respect to this function. Let $SL = \{p_1, p_2, \ldots, p_n\}$ be the sorted list of the given points using the above function. It can easily be noted that the Voronoi cell of the point p_i is smaller compared to the Voronoi cells of the points $p_{(i+j)}$ ($i = 1, 2, \ldots, n$-1, $j = 1, 2, \ldots, n$). The merging process begins with p_i (i is one initially). Then we find all the Voronoi circles which contain p_i on their circumference. Next, the points (say, p_l for some $l = 1, 2, \ldots, m$, $m \geq 2$ if we exclude p_i) on the smallest Voronoi circle that has p_i on their circumference are considered. If any of such points already assigned to a sub-cluster, then we add the point p_i to the same sub-cluster. If the points p_l are not assigned to any sub-cluster, we add p_i to a new sub-cluster. After dealing with a point of SL, we count the number of sub-clusters generated till this stage. If k clusters are formed, the process is terminated. Otherwise, i is incremented by one and repeat the same procedure with p_i, i.e., the next point of the sorted list SL. The process is continued for all the points of the sorted list SL. Once the sorted list SL is completely exhausted, then the process is restarted from the point p_i (i is initiated to one) of the sorted list SL. It is necessary to note that the sub clusters at this stage may contain multiple points unlike the sub-clusters in case of the previous scan of the sorted list SL. So, the only difference of merging the clusters from this stage is as follows. When a point of a sub-cluster is assigned to another sub-cluster, then the points of both of these sub-clusters are merged together. In this way we restart the procedure from the first point of the sorted list SL until the termination condition is encountered while scanning the points of SL. The unbounded cells in the Voronoi diagram are bounded by the intersection of the whole Voronoi diagram with a *d*-dimensional cube whose extrema are defined by the minimum and maximum attribute values in each dimension. The pseudo code of the proposed algorithm is as follows.

Algorithm-Voronoi-Cluster (*S*, *k*)

***Input*:** A set *S* of *n* data points, a value *k* of the required number of clusters

***Output*:** A set of clusters

Functions and variables used:

Vor (*S*): A function when called constructs the Voronoi diagram for the given data set *S* of *n* points.

V (*p*): Voronoi cell of the point *p*.

CirS(*v*): Voronoi circle of the vertex *v*.

R (*v*): A function to find the radius of the Voronoi circle *CirS*(*v*) of the vertex *v*.

Vcell (*p*): A function to find the error of the Voronoi cell of the point *p*.

Sort (*S*): A function to sort the points of *S* in monotonically increasing order as per the function *Vcell* ().

SL []: An array to store the sorted points.

Sum, *i*, *j*, *h*, *m*, *r*, *a*, *b*, *p*, *q*: Temporary variables.

```
Step 1: Call Vor (S) to construct the Voronoi diagram of the given set S.
Step 2: Call Vcell ($p_i$) to find the error of the Voronoi cells of the points $p_i$.
Step 3: Call Sort ($p_i$) to sort the points $p_i$ of S in monotonically increasing order of the
        values of Vcell ($p_i$).
Step 4: Store the sorted points $p_i$, $i = 1, 2, \ldots, n$ in the array of sorted list SL.
Step 5: Start with SL[j], find the Voronoi vertices $v_m$ (for some $m = 1, 2, \ldots, r$) whose
        circles CirS($v_m$) contain SL[j] on their circumferences. /* initially $j \leftarrow 0$ */
Step 6: Call R ($v_m$) to find the radius of the Voronoi circles CirS($v_m$).
Step 7: Find the points on the least radius Voronoi circle (say CirS(q)) which contain
        SL[j] on its circumference.
```

```
Step 8: If $SL[j]$ is already assigned to a sub-cluster $C_a$ then
        {
         If the points on the circumference of $CirS(q)$ are assigned to any sub-
             cluster $C_b$ then
                  Merge $SL[j]$ and all the other points of sub-
                  cluster $C_a$ to the sub-cluster $C_b$.
                  i.e., $C_b \leftarrow C_b \cup C_a$;
                          (or)
                  Merge all the points of sub-cluster $C_b$ to sub-
                  cluster $C_a$.
                  i.e., $C_a \leftarrow C_a \cup C_b$;
        }
      Else If the points on the circumference of $CirS(q)$ are assigned to any cluster
             $C_h$ then
                  Add $SL[j]$ to the sub-cluster $C_h$.
                  i.e., $C_h \leftarrow C_h \cup SL[j]$;
Step 9: If $k$ number of clusters are produced then Exit ();
         Else If ($j == |(SL-1)|$) then go to step 10;
          /* This indicates that the sorted list $SL$ is exhausted */
             Else
               {
                 $j \leftarrow j + 1$;
                 Go to step 5 to continue with the other points of the sorted
                  list $SL$;
               }
Step 10: $j \leftarrow 0$;
Step 11: Go to step 5 to repeat the same procedure from the beginning of the sorted
          list $SL$ for the next least radius Voronoi circles.
```

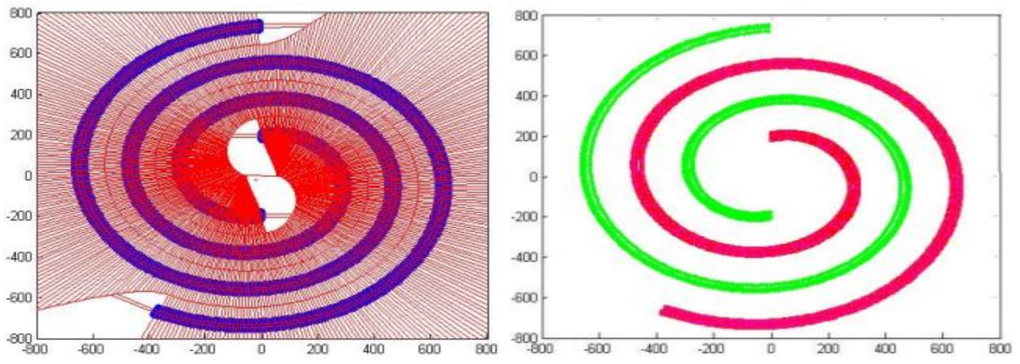
□

```
Function Vcell ($p_i$)
    {
      Step 1: Find the Voronoi vertices on whose circles (say $CirS(v_l)$ for some $l$ =
              1, 2,…, $t$) circumferences the point $p_i$ is located.
      Step 2: Call $R(v_l)$ to find the radiuses $r_1, r_2,…, r_t$ of $CirS(v_l)$.
      Step 3: $Sum = (r_1 + r_2 +…+ r_t)$;
      Step 4: Return ($Sum$);
    }
```

Time Complexity: The construction of the Voronoi diagram of given n points require $O(n \log n)$ time. The values of the function *Vcell* are computed for each of the given point in $O(n \log n)$ time. The points are sorted in $O(n \log n)$ time. Therefore, the overall time complexity of the proposed algorithm is $O(n \log n)$.

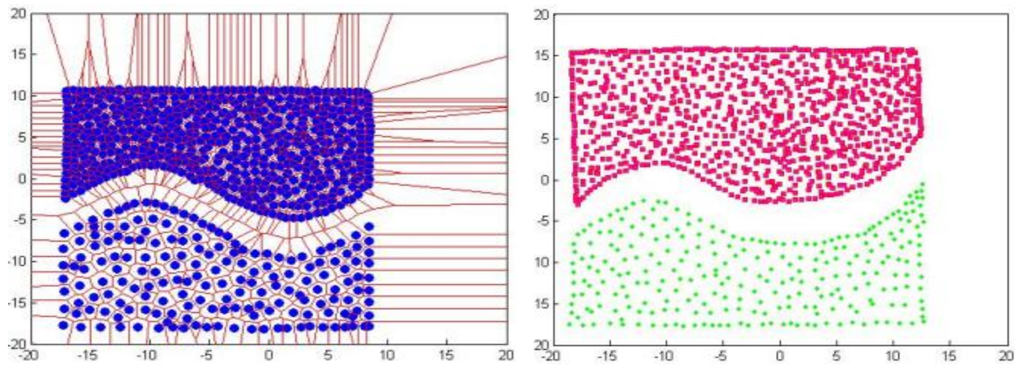
4.3.7 Experimental Analysis

In order to evaluate the performance, the proposed Voronoi diagram based method was applied on various two-dimensional synthetic and multi-dimensional biological data sets taken from UCI machine learning repository [32]. For the sake of visualization, initially, the proposed method was applied on four synthetic data sets, namely, 2-spiral, density separated, kernel and 5-group. The initial Voronoi diagrams of all these data sets are shown in the Figures 4.13(a), 4.14(a), 4.15(a) and 4.16(a). The results of the proposed algorithm on these four data sets are shown by the Figures 4.13(b), 4.14(b), 4.15(b) and 4.16(b) respectively. In case of 2-spiral data the proposed algorithm produced two clusters in spiral fashion as shown in Figure 4.13(b). In case of density separated data, the proposed method resulted two clusters separated by the densities as visible from the Figure 4.14(b). Similarly, the proposed algorithm produced the desired clusters in case of kernel and five-group data as depicted in the Figures 4.15(b) and 4.16(b).



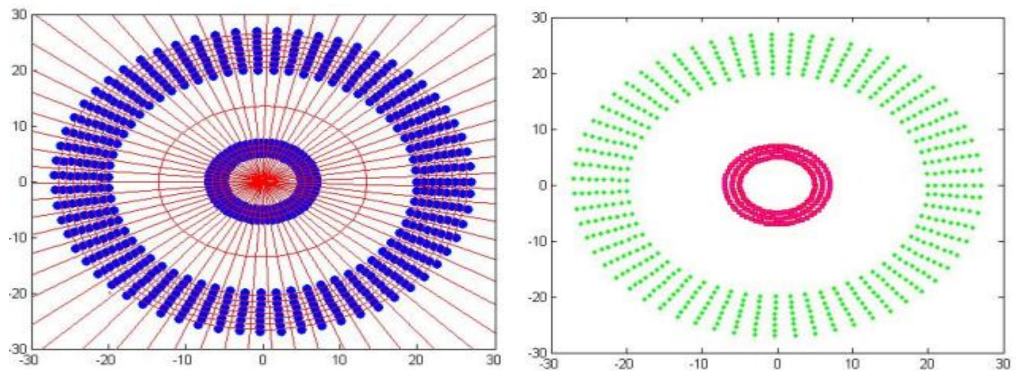
(a) (b)

Figure 4.13: Result of 2-spiral data (a) initial Voronoi diagram; (b) clusters produced by the proposed method



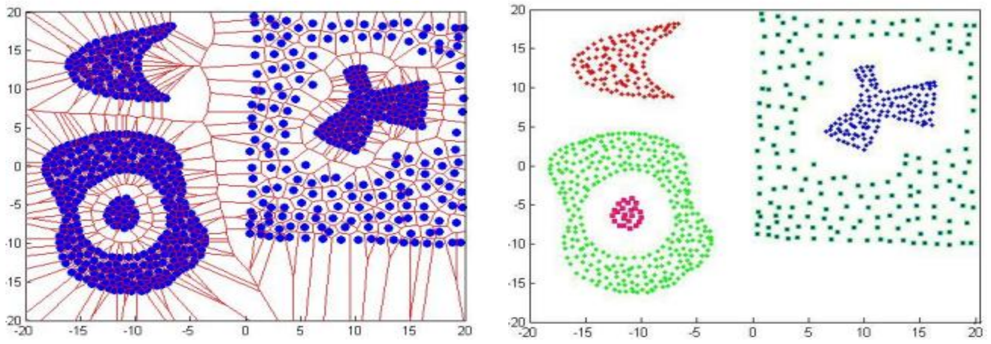
(a) (b)

Figure 4.14: Result of density-separated data (a) initial Voronoi diagram; (b) clusters produced by the proposed method



(a) (b)

Figure 4.15: Result of kernel data (a) initial Voronoi diagram; (b) clusters produced by the proposed method



(a) (b)

Figure 4.16: Result of 5-group data (a) initial Voronoi diagram; (b) clusters produced by the proposed method

Next, the proposed and the existing methods were applied on eight biological data sets [32], namely, iris, wine, statlog heart, breast tissue, Pima Indians diabetes, cloud, blood transfusion and yeast. The results were evaluated with the help of Dunn validity index (DUVI) discussed in section 4.2.2. For the sake of comparison, we used three clustering techniques, namely, *K*-means [26], VDAC [75] and SC [152]. The comparison results shown by Table 4.2 depict the performance of the proposed method over the existing techniques. As per the definition, the higher values of DUVI indicate more quality of the clusters. It is observed from the values of DUVI presented in the Table 4.2 that the proposed method outperforms the existing techniques *K*-means, VDAC and SC in case of all the biological data.

Table 4.2: Results of biological data using Dunn validity index (DUVI)

Data	**No. of Attributes**	**Data Size**	**Cluster No.**	**DUVI**			
				***K*-means**	**VDAC**	**SC**	**Proposed**
Iris	4	150	3	45.6453	80.9112	69.1209	**98.7262**
Spect Heart	22	187	2	34.4524	67.8278	55.4632	**102.4564**
Wine	13	178	3	27.6575	44.2755	39.4599	**88.3445**
Ecoli	7	336	8	46.5648	66.2727	77.3833	**139.2727**
Statlog Heart	13	270	2	25.9267	39.6354	48.2637	**78.2719**
Pima Indians Diabetes	8	768	2	65.3782	88.9326	76.3001	**145.6149**
Soybean	35	47	4	12.8038	13.2967	17.9263	**21.7279**
Breast Tissue	9	106	2	16.3928	23.1355	22.6782	**33.5623**

4.3.8 Algorithm Based on Voronoi Vertices

The basic idea of this approach is as follows. Given the set of *n* data points, say *S*, the Voronoi diagram *Vor*(*S*) is constructed initially. We then find the minimum number of Voronoi vertices (i.e., prototypes) to cover all the given points. To find such prototypes, the Voronoi circles are traced out. Then the Voronoi circles are sorted with respect to the radius of their largest empty circle. We consider the Voronoi circle with least radius and use the points on its circumference to form what is called initial cluster. We then consider the next least radius circle to form the next initial cluster. After the formation of all the initial clusters, we need to merge them efficiently to form the desired clusters. This is accomplished by constructing the new Voronoi diagram by inputting the Voronoi vertices as the initial points, where each vertex stands for a prototype. We repeat the same procedure until the Voronoi diagrams in two successive iterations are same. We now formally present the pseudo code of the proposed method as follows.

Algorithm Build-Voronoi (*S*, *μ*, *M*)

Input: $S[n][d]$, μ, M

Output: C_1, C_2,…, C_k

Notations used:

S: Given set of *n* points and *d* dimension

Vor(*S*): Voronoi diagram of the given set *S*

v_i : Voronoi vertex i =1, 2,…, $2n–5$ (max.)

CirS(*v*): Largest empty circle of vertex *v*

R(*v*): Radius of the Voronoi circle *CirS*(*v*).

r: A temporary variable.

μ: Threshold value on the radius of the Voronoi circle to separate the clusters.

M: Threshold value on the density of clusters to detect the outliers.

4.3.8 Algorithm Based on Voronoi Vertices

```
Step 1: Given a set $S$ of $n$ points, construct the Voronoi diagram $Vor(S)$.
Step 2: Sort all the Voronoi vertices $v_i$ in ascending order with respect to the radius of
        their largest empty Voronoi circle's $CirS(v_i)$, $i = 1, 2, \ldots, 2n-5$ (max.) and
        store them in an array $V[]$.
Step 3: Repeat steps 4 through 6 for $i = 1, 2, \ldots, 2n-5$
Step 4: Assign the radius of $CirS(V[i])$ to $r$, i.e., $r = R(V[i])$
Step 5: If $r \leq \mu$ then locate all the points lying on the boundary of $CirS(V[i])$. These
        are the points with respect to the $i^{th}$ prototype $P_i$. If any point is already
        covered by a circle then ignore that point.
Step 6: If $r > \mu$ then the uncovered points (if any) are left for the separate prototypes
        and hence exit the loop.
Step 7: Store all the vertices $V[i]$ generated in step 5 and the uncovered points from
        step 6 in the set $S'$.
Step 8: If $S = S'$ then go to step 9
        Else construct the Voronoi diagram Vor($S'$) for the set $S'$ and go to step 2.
Step 9: Merge the points of $S'$ based on $\mu$ distance to form $k$ groups $g_1, g_2, \ldots, g_k$.
Step 10: Explore the $k$ groups to obtain the set of clusters, say $\{C_1, C_2, \ldots, C_k\}$.
Step 11: For $i = 1$ to $k$
             If $|C_i| < M$ then declare $C_i$ as the outlier.
Step 12: Stop.
```

Time Complexity: Step 1 requires $O(n \log n)$ time for the construction of the Voronoi diagram of the n data points. Step 2 also requires $O(n \log n)$ time for sorting. Steps 4 through 6 are repeated at most $2n - 5$ time in which each of the steps 4, 5, and 6 requires constant time and thus they require $O(n)$ time in total. Step 7 requires linear time. Therefore steps 2 through 7 require $O(n \log n)$ time in total. However, steps 2 through 8 are repeated a finite number of times, say q times in which construction of the Voronoi diagram is the dominating computation. As q is significantly smaller compared to n, the total time complexity of the proposed algorithm is $O(n \log n)$.

4.3.9 Experimental Analysis

In order to prove the efficiency of the proposed method, we considered various multi-dimensional biological data [32] such as iris, wine, statlog heart, breast tissue, Pima Indians diabetes, cloud, blood transfusion and yeast. The proposed algorithm was experimented on all these data sets. The results were compared with the existing techniques namely, *K*-means [26], FCM [158] and CTVN [76]. The comparison results are shown by means of Dunn validity index (DUVI) in Table 4.3. The DUVI values of the proposed method are higher compared to the values of the existing methods *K*-means, FCM and CTVN as depicted in the Table 4.3. Therefore, as per the definition of DUVI, it is observed that the proposed method produces better results than that of the existing.

Table 4.3: Comparison of the proposed scheme with *K*-means, FCM and CTVN using Dunn validity index (DUVI)

Data	**No. of Attributes**	**Data Size**	**Cluster No.**	**DUVI**			
				***K*-means**	**FCM**	**CTVN**	**Proposed**
Iris	4	150	3	45.6453	56.2683	77.6561	**81.7431**
Wine	13	178	3	27.6575	44.2159	49.5128	**55.3587**
Statlog Heart	13	270	2	25.9267	21.2575	82.5998	**121.6244**
Breast Tissue	9	106	2	16.3928	17.1104	19.2815	**19.8939**
Pima Indians Diabetes	8	768	2	65.3782	50.0237	47.4635	**177.8923**
Cloud	10	1024	2	120.3771	103.6622	134.1253	**156.0308**
Blood Transfusion	5	748	2	148.4003	167.1972	161.2036	**131.3672**
Yeast	8	1484	10	85.6354	91.3289	143.6734	**191.3015**

4.4 Conclusion

In this chapter, we proposed three clustering algorithms with the help of Voronoi diagram. The algorithm GCVD has been designed using the edges in the Voronoi diagram. This method tested on various synthetic and gene expression data namely, Yeast cell-cycle, Sporulation, Lymphoma, Diauxic and Fibroblasts. The results are compared with *K*-means, FCM, CTVN and SC in terms of Dunn validity index. All the comparisons clearly proved the efficiency of the proposed method than the existing. We also proposed two algorithms namely, Voronoi-Cluster and Build-Voronoi using the Voronoi circles and vertices respectively. The notion of the algorithm Voronoi-Cluster is directed by a real valued function based on the Voronoi circles. This algorithm runs in O (n log n) time. Similarly, the algorithm Build-Voronoi is designed by reconstructing the Voronoi diagram using the vertices. Both Voronoi-Cluster and Build-Voronoi are experimented on different artificial and biological data. It is observed that the proposed algorithms outperform the *K*-means, VDAC, SC, FCM and CTVN algorithms.