Clustering Algorithms

Leonid Peshkin
under the supervision of Prof. Shimon Ullman
Weizmann Institute of Science
Rehovot, ISRAEL

July 1993

Contents

1 Introduction

In this paper we compare several existing clustering algorithms and propose a few improvements. By clustering we understand assigning set membership to points (objects) based on some similarity. We concentrate on grouping points into sets motivated mostly by human perception. Development of appropriate algorithms will be illustrated by examples from a two-dimensional space. However, all considered methods will be applicable to higher dimensional spaces.

The clustering algorithms are generally divided into sequential and parallel. *Parallel*algorithms operate on the whole dataset (of finite size) simultaneously. Such problem will be considered in section 3. *Sequential* algorithms get the data points one by one continuously in time (and in principle, indefinitely). These algorithms analyze current information, learn the set properties from the given subset and try to classify new data points based of the information extracted from all previous ones.

In the problem that is considered in section 2 we assume that there is no possibility to store all the data in memory. It means there is no possibility for post-processing or pre-processing of the whole dataset. In some literature sources such formulation of clustering problem referred to as *online* clustering. Figure 1 illustrates data sets we are going to use in the discussion.

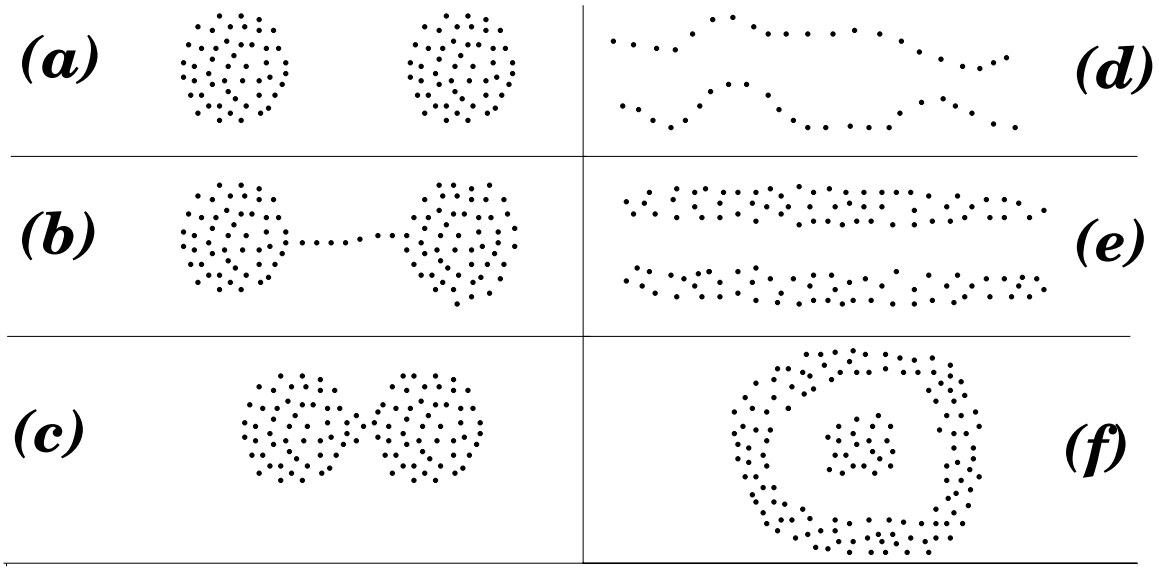

Figure 1. The sample clustering data sets.

We can effortlessly perceive two clusters in each case. Let us now consider how different algorithms deal with these examples.

2 Sequential clustering

In this section we present the empirical study of "Trained cells" algorithm. In addition, the "Two thresholds" algorithm is presented and briefly analyzed. On the basis of these two algorithms the new 'combined' algorithm is proposed and studied.

2.1 *Trained cells* algorithm

This work was initiated to consider sequential clustering algorithm presented in Ambros et al. [1]. Clusters are associated with cells responsible for each cluster representation and recognition respectively. The idea to implement clustering solution by *winner-take-all* network has been presented by Kohonen [5]. This idea is very similar to the one presented in Ambros et al. [1].

2.1.1 Algorithm

Here we use $D(X_1, X_2)$ to denote some kind of distance between points X_1 and X_2. Usually it simply stands for an Euclidean distance.

1. Do steps 2 to 4 for each input X.
2. Identify winning cell C using the measure $D(X, C)$.
3. Assign input X to the appropriate cell.
4. Train winning cell, $C \leftarrow C + \gamma(X - C)$.

Here γ is a so-called learning rate. The authors of the paper suggest that γ should satisfy:

$$\begin{cases} \lim_{t\to\infty} \gamma(t) = 0 \\ \Sigma_t \gamma(t) = \infty \end{cases} \tag{2.1.1}$$

where t is the number of times cluster has been trained. It is easy to show that approximate complexity of this algorithm is $O(N)$. The **main idea** of this algorithm is "cell training". The similar idea for *vector quantization* have been presented by Kohonen [6]. We will consider it more precisely in the section 3.

2.1.2 Drawbacks

Let us analyze the drawbacks:

- The algorithm requires the number of resulting clusters as an input parameter, and makes no suggestions how to choose it.
- The algorithm does not specify the initial cluster's centers positions.
- The algorithm has no mechanism for clusters merging/creating.

2.1.3 Implementation and Experiments

The two modifications of this algorithm were studied with respect to γ.

In the first modification the **mean weighted** learning rate was used.
To find a mean weighted σ_{k+1} for $k+1$ numbers, given a σ_k and x_{k+1}:

$$\sigma_{k+1} = \frac{\sigma_k * k + x_{k+1}}{k+1} = \sigma_k + \frac{1}{k+1} * (x_{k+1} - \sigma_k) \tag{2.1.3}$$

It corresponds to the Step 4 of algorithm with $\gamma = \frac{1}{k+1}$. As we can easily see both requirements holds.

In this case the cluster center's position strongly depends on the first few points and rapidly converges to the weight center of perspective cluster that is not always useful.

In the second modification a **constant** learning rate ($\frac{1}{100} \leq \gamma \leq \frac{1}{20}$ from experiments) was used. In this case the cluster center's position never converges and always strongly depends on the last few points. Indeed in case of simple, well separated, circle-like shape clusters, the first algorithm converges a bit faster, while the results of both algorithms are quite equal.

2.1.4 Input data order dependence

Input data order dependence is a significant criterion of sequential clustering algorithms. The studies of input data order dependence show that with certain assumption the algorithm modification with **constant** learning rate is very efficient.

Assumption: the next input item should appear quite close to one of the existing clusters.
For $2D$ case it means that clusters appear as though they are painted from a spray paint. The following picture illustrates this fact.

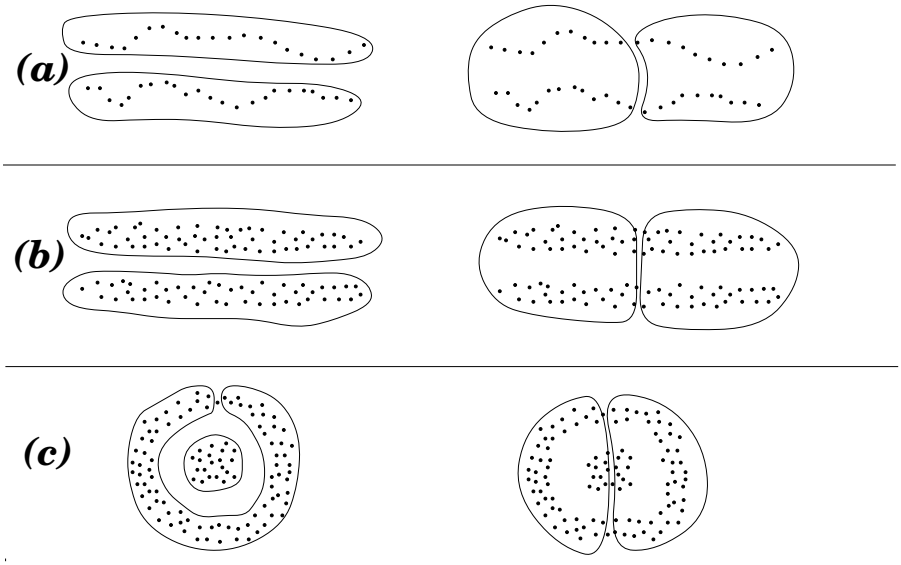

Figure 2. Results for ordered (left) and permuted (right) data.

The right column represents outputs of the constant learning rate algorithm in case of randomly permuted input while the left column represents the case when input corresponds to the assumption. The output of the algorithm with **mean weighted** coefficients is the same as in the right column independently of input data order.

This behavior is quite reasonable because in the modification with **constant** learning rate the cluster centers follow the recent points all the time. Therefore they are able to manage with sophisticated cluster shapes. But in **mean weighted** modification the cluster centers quickly converge to the weight centers so they are able to manage with only circle clusters shape. Figure 3 illustrates another studied problem of input data ordering.

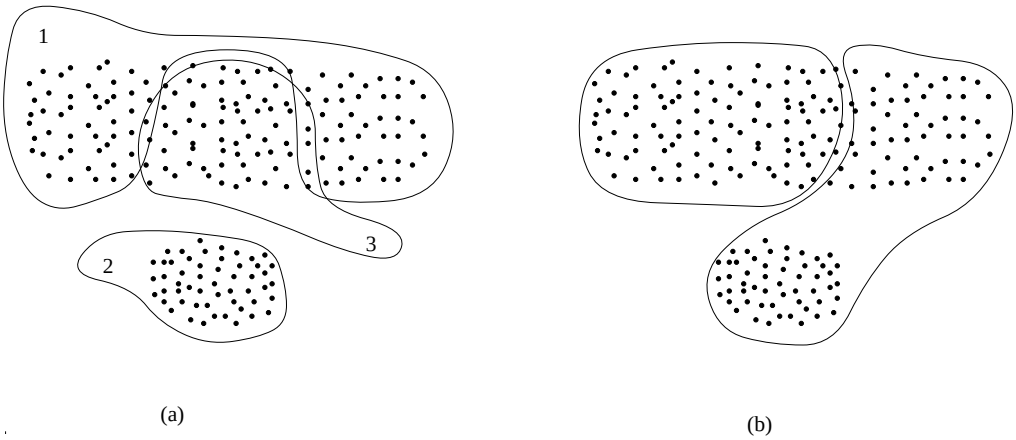

Figure 3. The case of late gap filling.

Figure 3(a) represents the given input set. At the beginning points should lie in area 1, then in area 2, and finally they have to fill the gap between two clouds, the area 3.
Figure 3(b) represents the result of training cells algorithm. It is almost the same for mean weighted and constant learning rates.
The parallel algorithm described in the third section gives two correct clusters.

2.2 *Two thresholds* algorithm

This algorithm was presented in [2]. It contains very simple but original idea that was taken as a basis for a new, 'Combined', algorithm described in the section 2.3 .

2.2.1 Algorithm

1. Assign the first item X_1 to a first cluster center C_1.
2. Do steps 3 to 4 for each input X.
3. Identify winning cluster C' that minimize the measure $D(X, C)$.

4. Analyze $D(X, C')$:
   (a) If $(D(X, C') \leq T_{min})$ assign X to C'
   (b) If $(D(X, C') \geq T_{max})$ create new cluster $C_{new} \leftarrow X$
   (c) If $(T_{min} < D(X, C') < T_{max})$ write X to pull.
5. If pull is empty terminate.
6. Do steps 3 to 4 for each X from pull.
7. If pull is changed goto Step 5. Otherwise make new cluster and assign first point from the pull as its center, then goto Step 5.

The **main idea** of this algorithm is a "gap" between two thresholds. The items lies in that gap have uncertain cluster membership. Thus the appropriate decisions are postponed.

2.2.2 Drawbacks

- The algorithm does not specify the criteria for choosing T_{min}, T_{max}.
- T_{min}, T_{max} remain the same during whole procedure.
- The algorithm has no mechanism for clusters merging.

2.3 *Combined* algorithm

On the basis of two algorithms described above (sect. 2.1, 2.2) I design new one that overcome named drawbacks. It does not require a prior knowledge of the clusters quantity, clusters initial parameters or any other initial data parameters like thresholds, spatial density, etc.

This algorithm works like self-adjusting system. When the general picture converge to the certain clusters set algorithm adjusts to it. If in some point picture tends to change the algorithm adjusts to a new one after a short while. This behavior can be illustrated on the example from Fig. 3. Let us consider it once more. When all the points are in the area 1, my algorithm works as there are two clusters. As soon as points appear in area 2 algorithm create third cluster in right destination. Then at most tricky point, when points start to fill in gab between two clusters (area 3) algorithm creates fourth cluster and then merges three clusters at one. There is a point when algorithm obtains four clusters, but it is just a transitional phase. Such behavior seems to be quite natural.

Another adjustment property is that algorithm is Scale Independent due to the way of thresholds setting and treatment.

2.3.1 Algorithm

Let ES(External source) be the source that gives us objects X (possibly infinitely) and (pull) is a limited size storage. ES can represent the simple data file or arbitrary external 'black box' that generates subjects. Let C_i denote center of i^{th} cluster, $d(X)$ - the distance between object X and closest cluster center, $d(X) = \min_k D(X, C_k)$. The general idea of the suggested algorithm is as following. Use the same policy as in 'two thresholds' to postpone decision in case of uncertainty. Assigning the subject to the certain cluster, train this cluster as in 'Trained cells'. If some cluster was created or even trained check whether there are two relatively close clusters and merge them.

For all points that goes to the pull collect statistical data as:
d_{min} - minimal distance between point in the pull and closest cluster center;
d_{max} - maximal distance between point in the pull and closest cluster center;
d_{mean} - mean distance between points at pull and their closest cluster center;

Thresholds (like those in 'two thresholds' algorithm) are treated on the following levels:
T_{max} equal to the d_{max} or to the maximal pairwise distance between cluster centers.
T_{min} is a weighted sum of d_{min} and d_{max}:

$$T_{min} = \alpha \cdot d_{min} + (1 - \alpha) \cdot d_{mean}; \qquad \alpha \in [0, 1]$$

The experiments have shown that α should lie in the fourth quarter of the interval.

The detailed algorithm:

1. $C_1 \leftarrow X_1$.
2. While ((pull is not full) and (ES is not empty)) $\Longrightarrow$
   Put all X_i from ES besides current X_{max} in pull, meanwhile calculating
   (a) $d_{min} = \min_i\{d(X_i)\}$
   (b) $d_{mean} = \frac{\Sigma_i\{d(X_i)\}}{N}$ (N - number of objects in pull)
   (c) $d_{max} = \max_i\{d(X_i)\}$ (Memorize appropriate X_i as X_{max}.)
3. Set $T_{min} = \frac{1}{2}(d_{min} + d_{mean})$; $T_{max} = d_{max}$; $C_{new} \leftarrow X_{max}$;
4. For all X_i from (pull) :
   (a) If $(\exists j : C_j,\ d(X_i) < T_{min}) \Longrightarrow$ Assign $X_i \in C_j$; Train C_j.
   If $(\exists l, k : D(C_l, C_k) < d_{mean}) \Longrightarrow$ **Merge** $C_l + C_k \rightarrow C_l$
   (b) If $(\exists j : C_j,\ d(X_i) > T_{min}) \Longrightarrow X_i \rightarrow (pull)$
   Note: $d(X_i) < T_{max}$ due to T_{max} choosing.
   Calculate d_{mean},d_{max} (Memorize X_{max}).
   (c) Set $d_{min} = T_{min}$.
5. If (ES is empty) $\Longrightarrow$

(a) If ((pull) is empty) $\Longrightarrow$ **TERMINATE**.

(b) If (Size(pull) was changed during the Step 4) $\Longrightarrow$ goto Step 4
Else goto Step 3.

6. Analyze $d(X_i)$:

(a) If $(\exists j : C_j,\ d(X_i) < T_{min}) \Longrightarrow$ Assign $X_i \in C_j$; Train C_j.
If $(\exists l, k : D(C_l, C_k) < d_{mean}) \Longrightarrow$ **Merge** $C_l + C_k \rightarrow C_l$

(b) If $(d(X_i) > T_{max}) \Longrightarrow$ Create new cluster $C_{new} \leftarrow X_i$;
If $(\exists l, k : D(C_l, C_k) < d_{mean}) \Longrightarrow$ **Merge** $C_l + C_k \rightarrow C_l$
Else set $T_{max} = d(X_i)$;
New-Clust-Created $\equiv$ TRUE;

(c) If $(T_{min} \leq d(X_i) \leq T_{max}) \Longrightarrow X_i \rightarrow (pull)$. Calculate:

i. $d_{min} = \min_i\{d(X_i)\}$

ii. $d_{mean} = \frac{\Sigma_i\{d(X_i)\}}{N}$

iii. $d_{max} = \max_i\{d(X_i)\}$ (Memorize appropriate X_i as X_{max}.)

If ((pull) is full) $\Longrightarrow$

i. If (NOT New-Clust-Created) $\Longrightarrow$ Created new cluster $C_{new} \leftarrow X_{max}$;
If $(\exists l, k : D(C_l, C_k) < d_{mean}) \Longrightarrow$ **Merge** $C_l + C_k \rightarrow C_l$
Else set $T_{max} = d(X_{max})$.

ii. goto Step 4.

7. If (ES is empty) $\Longrightarrow$ goto Step 4
Else goto Step 6

2.3.2 Implementation and Experiments

The complexity of algorithm is quite difficult to calculate analytically, but computer experiments have shown that it is something like $O(N \log N)$. Of course it is very differs for concrete cases with the same number of subjects.

Interesting result gives us situation with algorithms data pull. We can run the algorithm with a different pull size. The output for different pull sizes is almost the same. Of course pull should not be too small (just few points). The size of pull is significant for the learning time. Under learning time here I mean the time when algorithm's output is noisy due to changes in general situation (or at the beginning). Clearly, smaller pull leads to a longer learning. So, as we have seen new algorithm does well with sphere-like shaped clusters. Let us now consider what happens for 'unsolvable puzzles' (e)-(f) from Fig. 1.

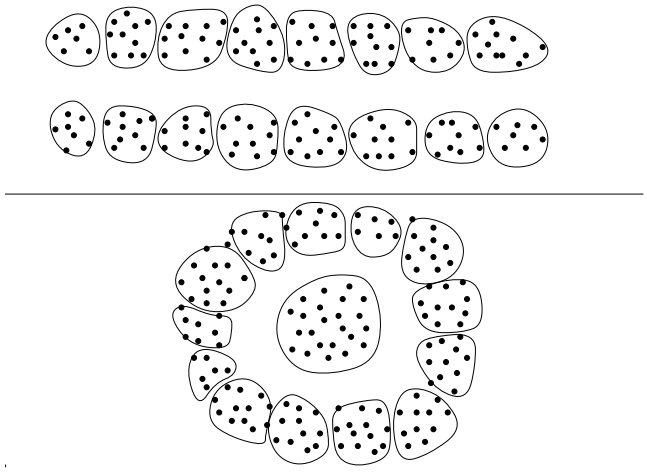

Figure 4. The output of algorithm for 'unsolvable puzzles'.

Although one can think that this results is not better that those received from 'trained cells' algorithm (Fig. 2) it is not true. Such kind of output gives a broad field for ideas and development. For example if we assume that simple post-processing is possible, it is reasonable to think about representation of sophisticated shape clusters by group of elementary clusters. Let call it 'necklace representation'.

Indeed it is not representation with group of spheres. Boundaries of elementary clusters construct Voronoi diagram. There are few possible ways to construct 'enhanced Voronoi diagram' for whole clusters from Voronoi diagram of elementary clusters. Kohonen [6] gives similar idea for *vector quantization*. He claims that a class (cluster) can be represented by several *reference vectors* and then class regions (cluster borders) are defined by a simple nearest-neighbor rule. It might be a good idea to extend this approach for sequential clustering. It is a great opportunity for continuation of this projects but unfortunately it is beyond the scope of this paper.

2.3.3 Drawbacks revised

Let us review the drawbacks of previous algorithms with respect to this one. This algorithm does not need the number of clusters to be given. It has the certain criteria to choose T_{min}, T_{max} and change them adaptively to the input stream. It has the mechanism for clusters creations and merging.

2.3.4 Input data order dependence

The Input data order dependence analysis gives me an interesting results. On the ordered data (accordingly to the Assumption in Sect 2.1.4) algorithm works correctly. It does the same on randomly shuffled data of the same file. But in the second case it works rather fast. The matter is that at the first case the algorithm has a lot of uncertainty about general situation and must postpone a lot of decisions. When in second situation algorithm get the general picture quite fast so it can make effective decisions without postponing them.

3 Vector quantization

3.1 Approaches

A parallel clustering procedure can be viewed as one of finding groupings in a set of events by optimizing some criterion function. Clustering strategies derived from an optimization principle are known as **vector quantization**. The most prominent algorithms are the MacQueen $K - means$ clustering algorithm [8] and the LBG vector quantization algorithm by Linde *et al.* [9].

One of the most widely used criterion functions for clustering analysis is the sum of squared Euclidean distances measured from the cluster centers. The main task is then to seek the grouping that minimize the sum-of-squared errors. Formally it can be expressed as follows:

Given N data points $X_1, X_2, ..., X_N \in R^k$ in a k-dimensional vector space, find M ($M << N$) cluster centers $Y_1, Y_2, ..., Y_M \in R^k$ and $\phi : \{X_1, X_2, ..., X_N\} \rightarrow \{Y_1, Y_2, ..., Y_M\}$ such that the average sum-of-squared errors defined by

$$E = \frac{1}{N}\Sigma||X_i - \phi(X_i)||^2$$

is minimum, where $\phi(X_i) \in \{Y_1, Y_2, ..., Y_M\}$ is the cluster center closest to X_i. Formally:

$$\phi(X_i) = argMin_{1 \leq j \leq M}||X_i - Y_j||^2$$

It is known that the problem of finding the global minimum solution such Equation is NP-complete [10]. For this reason, a number of approximate clustering techniques have been developed.

These techniques can be divided into two categories: iterative optimization and heuristic methods. The algorithm most frequently used for seeking a local minimum solution is the k-means iterative procedure. However, the converging time required by the iterative schemes can easily become unmanageable, particularly for large clustering problems.

Instead of finding local minimum solution, the heuristic approach is designed to reduce the computational complexity, and at the same time produce an acceptable solution. There are two basic heuristic methods: the agglomerative and the divisive techniques.

3.2 Learning vector quantization

Here we will consider approach that was suggested by Kohonen [6]. It is especially interesting because has a direct correspondence with algorithms considered in the section 2.

The consideration will be given in terms of classes(clusters) and their boundaries. If one wants to *classify* the signal sets into a finite number of categories, the decisions at the class boundaries are important. It is possible to define such *effective* values for *codebook* vectors that they directly define *optimal decision boundaries* between the

classes, even in the sense of Bayesian decision theory. These strategies and learning algorithms are called *Learning Vector Quantization* (**LVQ**).

LVQ algorithm places the a fixed number of reference ("codebook") vectors m_i into the pattern space. The initial values should roughly correspond to the statistical density function of inputs $x \in R^n$. The next step is to calibrate the "codebook" vectors, by presenting a number of vectors with known classification. The vector m_i cam *learn* optimal values if they are updated according to the following algorithm. Let c is class such that m_c is closest to x in appropriate metric (e.g. Euclidean); this is then also the classification of x. Apply training vector x with known classification. Update $m_i = m_i(t)$, where t is the discrete-time index ($t = 0, 1, ...$):

$$m_c(t+1) = \begin{cases} m_c(t) + \alpha(t)(x(t) - m_c(t)) & if \;\; x \;\; is \;\; classified \;\; correctly, \\ m_c(t) - \alpha(t)(x(t) - m_c(t)) & if \;\; x \;\; is \;\; classified \;\; incorrectly, \end{cases} \tag{3.2}$$

$$m_i(t+1) = m_i(t) \quad for \;\; i \neq c.$$

Here $\alpha(t)$ is scalar gain ($0 < \alpha(t) < 1$) which is decreasing in time monotonically, e.g., linearly. A proper initial value often be .01 .

Let us now analyze the approach presented above. First let us note that "proper" initial value for $\alpha(t)$ is very close to the analogue suggested in section 2.1 . the learning rule is also very close to that from sections 2.1 and 2.3 . The main difference is that here we consider algorithm for supervised learning in vector quantization. And there is unsupervised self-adjusted algorithm for sequential clustering.

Another strong idea here is *negative learning*. The point of initialization is still weakly described. This is a string argument for a solution proposed in section 2.3 .

3.3 Single linkage algorithm

Below we'll consider one of the most popular agglomerative algorithm - 'Single linkage'.

In this part of the project simple criteria for the variant of 'Single linkage' algorithm [4] termination was suggested and studied.

3.3.1 Algorithm

1. Initially we assume that there are as many clusters as items , each cluster consisting of one item (singleton).

2. Do steps 3 to 4 while we get a single cluster (or somehow preliminary defined number of clusters).

3. Calculate all pairwise distances between clusters using some measure $D(X, C)$.

4. Identify "closest" pair of clusters and merge them using mean weighted value to represent new cluster.

3.3.2 Drawback and improvement

The drawback of the algorithm is that it does not give us the 'reasonable' in some sense number of clusters and their members. Its output is a dendrogram ('merging' tree) that needs additional intellectual analysis. An analysis of those dendrograms gives us a simple criterion for termination :

Terminate when the ratio of minimal distance (for merged clusters) on current iteration to minimal distance on last iteration more than a THRESHOLD. ($1 \leq THRESHOLD \leq 1.6$ *from my experiments.) And when number of clusters less than* log *(number of points). (Last requirement deals with non-homogeneous set cases.)*

The only defect of such criterion is a set of clusters with relatively small number of members that appears beside main 'reasonable' cluster set. We can easily determine such 'noisy' clusters and eliminate them. Figure 5 give us an example of such 'noise'. It is third cluster consisting of the 'neck' points.

3.3.3 Implementation and Experiments

Here the output of 'Single linkage' for a problem set from Fig. 1 presented. First observation from the computer experiments in real time was that disadvantage of parallel algorithms is their inefficiency. Current experiments with 10,000 - 15,000 subjects set gives few seconds for sequential and half an hour for parallel algorithm. It is easy to show that approximate complexity of this algorithm is $O(N^3)$.

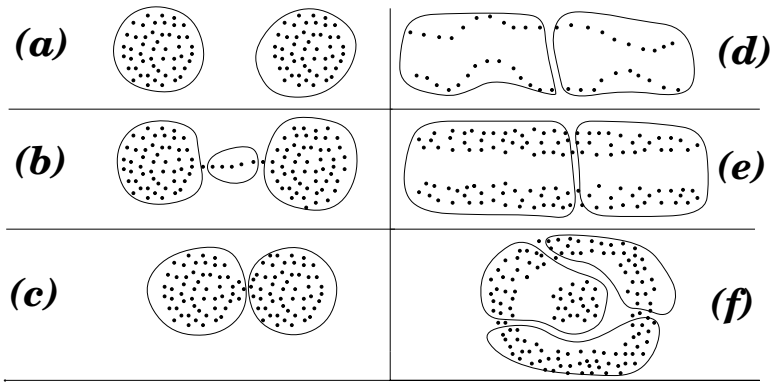

Figure 5. The parallel algorithm results for problems (a) - (f).

The wrong answer in problems (d)-(f) is quite reasonable because these clusters can not be represented by weight centers. But output for the rest of examples proves that suggested termination criterion is correct.

3.3.4 Input data order dependence

It is quite obvious that due to parallel data processing the algorithm is absolutely independent of input data order.

4 Conclusions

We considered various approaches to the clustering problem, focusing on the sequential clustering. This problem relates to a natural model of adaptive system behavior. Building on a few existing algorithms we suggested a new one which seem to be more effective and applicable to various domains. There are still many issues with our algorithm. Especially tricky point is how to treat sophisticated shape clusters. By "treatment" I mean the issue of correct representation. Representation should satisfy certain requirements. It should allow easy solution of the two tasks. Having gotten a new data point we should find the nearest cluster's border. The second task is how to change cluster representation according to a new member.

The results produced by our algorithm suggest a few ideas. The sketch of possible representation called 'necklace' is given here. We would represent a sophisticated shape as an ensemble of elementary sphere-like shapes. It means that each cluster consist from few elementary clusters that are treated separately, yet belong to one hyper-cluster. Almost every shape in R^n can be approximated by such ensemble of elemental spheres. Notice that both "unsolvable" problems mentioned above are quite easy for elementary clusters. The main problem here is to find correct criterion for elementary clusters grouping.

An empirical part of this project consists of the software development. First part is a software implementing 'Training cells' algorithm for different learning rates; 'Two thresholds' algorithm implementation; 'Mixed' algorithm implementation. The second part consists of interface programs for interactive data input and visualization of results. Those programs written for graphics library of Silicon Graphics computer. Beside that the program which randomly shuffles the data set have been written. All the code is written in C programming language.

References

[1] J. Ambros-Ingerson, R. Granger, G. Lynch, *"Simulation of Paleocortex Performs Hierarchical"*, Science, 1990, 247:1344-1348.

[2] P. Trahanias, E. Skordalakis, *"An efficient sequential clustering method"*, Pattern Recognition, 1989, 22, 4:449-453.

[3] Charles T. Zahn, *"Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters"*, IEEE Transactions on computers, 1971, C-20:68-86.

[4] Richard O. Duda,Peter E. Hart, *"Pattern classification and scene analysis"*, A Wilet-Interscience publication, 1986, ch.6.

[5] Kohonen T., *"Self-Organization and Associative Memory"*, Springer-Verlag, 1984.

[6] Kohonen T., *"Learning Vector Quantization"*, Neural Networks, 1988, v.1, Supp 1(INNS Proceedings).

[7] S.J. Wan, S.K.M. WONG, and P. PRUSINKIEWICZ, *"An Algorithm for Multidimensional Data Clustering"*, ACM Transactions on Mathematical Software, June 1988, 14, 2:153-162.

[8] MacQueen, J. ,*"Some methods for classification and analysis of multivariate observations"*, Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, 1967, pp. 281-297.

[9] Linde Y., Buzo A., and Gray R. M., *"An algorithm for vector quantizer design"*, IEEE Transact. Commun. COM, 1980, 28, pp.84-95.

[10] Hyafil L., and Rivest R.L., *"Construction optimal binary decision trees is NP-complete"*, Inf. Process. Lett., May 1976, 5, pp.15-17.