Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

Master thesis

Hierarchical Clustering of Long-term EEG Data

Bc. Matej Murgaš

Supervisor: Ing. Václav Gerla, Ph.D.

May, 2016

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

DIPLOMA THESIS ASSIGNMENT

Student:	Bc. Matej M u r g a š
Study programme:	Biomedical Engineering and Informatics
Specialisation:	Biomedical Engineering
Title of Diploma Thesis:	Hierarchical Clustering of Long-Term EEG Data

Guidelines:

This work deals with hierarchical clustering of long-term EEG data. Such approach allows to discover hidden structures in analyzed signals. The results of clustering can be used as input for the signal segmentation, classification, or building of the EEG database.
The main objectives of the work:

1. Prepare a state-of-the-art of the hierarchical clustering algorithms suitable for large data sets.
2. Design and implement a solution enabling efficient clustering of large data sets (up to hundreds of thousands of objects).
3. Compare the implemented approach with traditional agglomerative hierarchical clustering algorithm.
4. Validate the implemented approach on real clinical EEG recordings.

Bibliography/Sources:

[1] Niedermeyer E, and Lopes da Silva F. Electroencephalography – Basic principles, clinical applications and related field. Philadelphia: Lippincott William & Wilkins, 2005
[2] Loewenstein Y, Portugaly E, Fromer M and Linial M. Efficient algorithms for accurate hierarchical clustering of huge datasets: tackling the entire protein space. Bioinformatics, 24, 13:i41-9, 2008
[3] Cheng-Hsien T, Meng-Feng T, Shan-Hao Ch, Jen-Jung Ch, Wei-Jen W. Shortest-Linkage-Based Parallel Hierarchical Clustering on Main-Belt Moving Objects of the Solar System, Future Generation Computer Systems-The International Journal of Grid Computing and eScience, 34:26-46, 2014
[4] Vijaya P A, Narasimha Murty M, Subramanian D K. Leaders–Subleaders. An efficient hierarchical clustering algorithm for large data sets. Pattern Recognition Letters, 25(4):505-513, 2004

Diploma Thesis Supervisor: Ing. Václav Gerla, Ph.D.

Valid until: the end of the summer semester of academic year 2016/2017

L.S.

prof. Dr. Ing. Jan Kybic	prof. Ing. Pavel Ripka, CSc.
Head of Department	**Dean**

Prague, December 17, 2015

Acknowledgment

I would like to thank my supervisor Ing. Václav Gerla, Ph.D. for helping me with this work and for the given opportunity to participate in creating of PSGlab toolbox.

Author statement for undergraduate thesis

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instruction for observing the ethical principles in the preparation of university theses

Prague, date May 25, 2016 ....

signature

Abstrakt

Táto diplomová práca sa zaoberá hierarchickou zhlukovou analýzou nad dlhodobými záznamami vytvorenými elektroencefalografom. Cieľom práce je nájsť takú hierarchickú zhlukovú analýzu, ktorá bude používať menej pamäte ako klasický prístup a bude pracovať v rozumnom čase s objemnými dátami. V tejto práci porovnávame aglomeratívnu zhlukovú analýzu, ktorá reprezentuje klasický prístup, oproti dvom ďalším metódam, robustné actívne zhlukovanie a hybridné zhlukovanie. Dendrogram, ktorý vzniká ako výsledok hierarchickej zhlukovej analýzy, môže byť použitý na detekciu rôznych zhlukov reprezentujúcich artefakty v reálnom EEG signály. Na porovnanie navrhovaných metód sú použité rôzne dáta. Jedným druhom použitých dát sú umelo vytvorené, ktoré obsahujú tri jasné zhluky a druhým typom dát sú reálne EEG záznamy zastúpené hypnogramami a komatóznymi dátami. Všetky implementácie boli urobené v MATLABe a výsledky tejto práce budú použité pri vývoji PSGlabu.

Kľúčové slová: aglomeratívny, hybridný, actívny, hierarchický, zhlukovanie, k-means++, PCA, EEG

Abstract

This master thesis deals with hierarchical cluster analysis on brain activity recordings created by electroencephalograph. The aim of the work is to find a hierarchical clustering method that use less memory than classical approach in the reasonable time for large datasets. In this work the classical approach, Agglomerative Hierarchical clustering, is compared to the two other approaches, Robust Active clustering and Hybrid cluster analysis. Dendrogram generated by the hierarchical clustering can be used to detect various numbers of cluster that represents artifacts in the real EEG signals. Different datasets are used for the complex comparison of the methods. We use artificial datasets with clear clusters and real EEG datasets represented by hypnograms and comatose data. For the implementations and comparisons we use MATLAB. The results of the work will be used in the PSGlab toolbox.

Keywords: agglomerative, hybrid, active, hierarchical, clustering, k-means++, PCA, EEG

Contents

List of Figures

List of Tables

1 Introduction

One of the most important and most interesting parts of the human body is brain. We can say that the brain is the center of human body, since each organ is controlled by it and it is the center of central nervous system. Based on the previous facts, it is possible to state that the processing and analyzing the signals produced by brains is really important. It is helpful in diagnose illnesses connected with nervous system such as epilepsy, sleep disorders or it helps to monitor a brain activity of patients in coma and many more contributions. The drawback, that makes analyzing of EEG difficult for each neurologist, is that the functionality of human brain is not completely explored. Furthermore it causes that it is not possible to fully automatize processing and analyzing the EEG signal. Another problem comes with various artifacts. In the EEG signal are contained eye movements, heartbeat, muscle activity and others, that makes analyzing of the EEG signal more difficult for the neurologists. In order to get the best possible analysis of the EEG signal, neurologist would spend a few hours with the processing.

Cluster analysis can be helpful in analyzing procedure of EEG signal. We introduce and compare different hierarchical cluster analysis approaches that are used to find different groups of data samples. By creating a hierarchical tree we get a strong tool to find a different number of clusters in the data set. It is an advantage if we need to process EEG signal that contains a various numbers of unwanted elements. These artifacts represents separated clusters in the structure of data. There are numerous methods for performing hierarchical cluster analysis on the data. Basically we can divide them to two different categories divisive and agglomerative. The second one is the most common used and has various techniques such as Complete link, Single link, Ward's method and others.

In this work we focus on finding algorithm that has lower memory consumption and reasonable time consumption, while the detection of clusters is similar to the classical approach. For the purposes of this work we use Agglomerative Hierarchical clustering based on Ward's method as a baseline method. As a new approach we decided to use Robust Active cluster analysis and Hybrid clustering. Results of this work will be used in PSGlab, which is being created by Ing. Václav Gerla, Ph.D.. It is a complex toolbox for reading, pre-processing, analyzing and classifying of EEG record.

2 EEG signal

There is a variety of methods that can be used in order to get information about the human body. Electroencephalography (EEG) is one of the them where the main area of interest is brain. It is basically a measurement of the electrical activity of the brain using electrodes. The first roots of EEG can be traced to 1875 when Richard Canton measured first signals from brains of a rabbit and a dog. In 1929, Hans Berg constructed first EEG device.

Brain is a part of central nervous system, and it consists of various kind of cells and structures. The most important ones are considered neurons and surrounding neural tissue - glias. The cells responsible for electrical activity in the brain are neurons that use this kind of activity for communication between each other. [3]

2.1 Origin of the signal

As mentioned above, the main purpose of EEG is to collect information about electrical activity of the brain. The base of such activity is an influence of chemical transmitters on post-synaptic cortical pyramidal neurons resulting in either localized depolarization - excitatory post-synaptic potential (EPSP), or localized hyperpolarization - inhibitory post-synaptic hyperpolarization (IPSP). Current in EPSP is carried by Na^+ inward flux. In the case of IPSP current is carried inwards by positive ions, such as K^+, and outwards by negative ions Cl^-. The flow of the ions trough the membrane causes the polarization of the neurons, and creation of electrical dipoles.

Pyramidal neurons are located in the most superficial layer of cortex, and are spatially organized, causing synchronization of the neuronal activity in the specific areas. Since the electrical activity of one neuron is so small that EEG would not record it, or it can be the mutually canceled the synchronization and orientation are important in the signal strengthening.

While on the one side of neuron there is a membrane repolarizing back to the resting potential $-70\,mV$, on the other side the neuron is being depolarized to more positive values. This effect causes the creation of dipoles that conduct current. One dipole is usually created of up to thousands of neurons. There are two possible orientations of the dipole in the brain - radial, which is orientated from the center of the brain to the surface and tangential, which is parallel to the brain surface. Layer of the dipoles is responsible for the electric activity that is recorded by the EEG.[16, 13]

2.2 Measuring EEG signals

Previous subsections explain emergence of the electrical signal in the brain. To acquire such signal, we have a recording system. The interface between biological signal and recording system compromises of electrodes and conductive jelly. Right behind the electrodes are located amplifiers with filters that should amplify useful signal and suppress noise. The last part of recording chain is A/D converter and a computer that works as a recording device.

Electrodes

As mentioned above, electrodes are crucial part of the recording system. There are several types of them, such as disposable and reusable disc electrodes, electrode caps (used for multi-chanel recording), and headbands. The common material used for the electrodes is *Ag-AgCl* that also allows recording of the slow changes in potential. To measure potential changes over time we need at least 2 electrodes. One of them is active, and second one is a reference electrode. Sometimes it is possible to use third one, called ground electrode, to get differential voltage. Position of the electrodes is also important, and it is internationally standardized as a $10-20$ system (Figure 2.1).

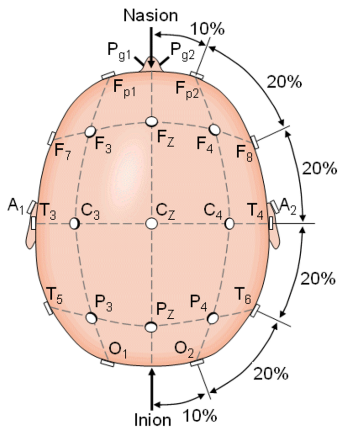


Figure 2.1: 10-20 electrode placement standard

The labels mark a position of the electrode. The letters stand for an area of the head that is being recorded by the specific electrode (F - frontal, C - central, T - temporal, P - posterior, O - occipital, A - earlobe, Pg - nasopharyngeal, Fp - frontal polar). Numbers explain the side of the head where the electrode is placed. Even number means that the electrode is on the right side, and the odd number marks electrodes that are on the left side. In the case if an electrode is in the middle, there is a letter *z* with no number.[19, 13]

Mathematical approach

The mathematical concept used for the explanation of the relation between dipoles and EEG potentials is called *lead-field*. The main idea of the concept is to construct lead-field matrix that expresses relation between discrete locations on the skull and potential measurements at discrete recording sites. Positive and negative electrodes are connected by virtual lines that cross all of the points in the brain. One of the disadvantages of the EEG is decrease in the intensity of acquired signal with increasing distance from the surface of the brain. [20]

2.3 Sleep stages and EEG

EEG signals acquired from the brain nowadays are used mainly for the diagnostic purposes and one application is to measure activity of the brain during the sleep. Using EEG records, it is possible to classify sleep into stages: wakefulness, REM (Rapid Eye Movement) and non-REM. Non-REM sleep is divided into 3 different stages (earlier classifications used 4 stages, but stages 3 and 4 were merged due to their similarities). These stages are also known as deep sleep or slow wave sleep, especially stage 3 where more than 50% of signal consist of delta waves. On the other hand, in REM sleep the EEG signal looks more like wakefulness, but body is completely immobilized. In this stage we usually see dreams. [3, 14]

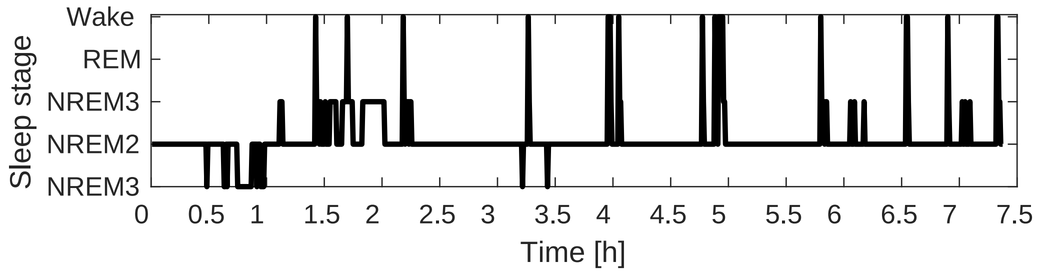


Figure 2.2: Example of Hypnogram

2.4 Coma

Coma data is another kind of signal that can be measured from the brain. Coma is a state of unconsciousness with missing wakefulness. Typical characteristic feature of coma is lack of arousal from internal or external stimuli. This state can be caused by various reasons (such as traumatic head injury, stroke, brain tumor, and drug or alcohol intoxication). Usually patient needs life support and is not able to communicate. Comatose data can be classified into 10 different stages, with $C1 - C10$ describing how deep the coma is. [8] An example of the coma stages can be seen in the following figure.

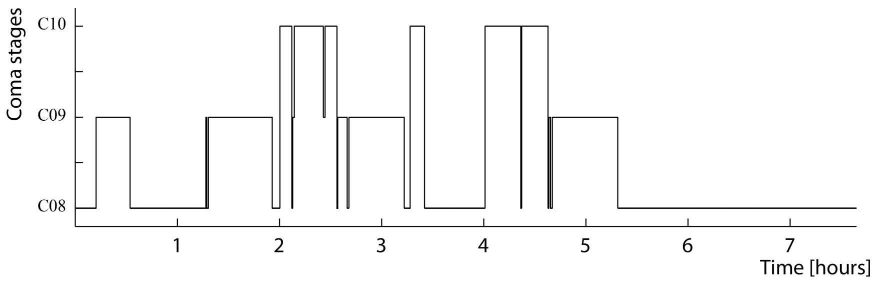


Figure 2.3: Example of Coma Stages

3 EEG signal processing

Even though the signal is already filtered when it is recorded, it has to be processed to obtain reasonable and meaningful data. Signal processing, in general, starts with digital filtration, then the signal is usually segmented and in the end the feature extraction and selection is done. If the information in the features is redundant it is feasible to reduce dimensionality of the data. There are various ways for the digital filtration but this is not in the scope of this work

3.1 Signal segmentation

Real signals are usually non-stationary. The same fact also stands for the EEG signals where statistical values vary over the time. Segmentation means division of the signal into smaller parts. In our case parts are so small that we can say that each segment is almost a stationary signal. Especially this pre-processing is necessary in the task of clustering and feature selection. In general there are two ways how to segment signal. In segments equal in length or various in length. In the case of same length of the segments we are talking about linear segmentation (for example see figure 3.1). The length of the parts is usually set according to empirical observations. In the case of EEG signals, we can consider segments that are approximately 10 seconds long as quasi-stationary signals. In this work we set the length of the segments to 30 seconds.

Another approach is adaptive non-linear segmentation. The advantage of adaptive segmentation is that the segments are more stationary than the ones we obtain from linear segmentation. The signal is divided by its changes. There are several algorithms for adaptive segmentation. The example of adaptive segmentation is in the figure 3.2. In case of this kind of signal separation we most probably get more segments than with linear segmentation. For example, if we take a linear segmentation of 7 hours long EEG signal with segments of 30 seconds, we get 885 segments. With adaptive segmentation this number can be easily few times higher.[8]

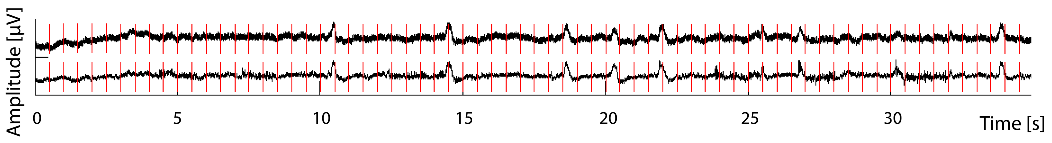


Figure 3.1: Example of linear segmentation

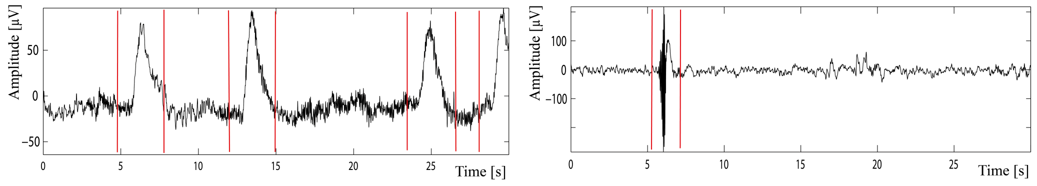


Figure 3.2: Example of adaptive segmentation

3.2 Features

After segmentation, we can represent each segment of the signal by its parameters. The basic and most commonly used features are statistical parameters such as mean, standard deviation, maximum, minimum, kurtosis and skeweness or median. Another type of parameters obtained by interval or period analysis. In these methods, the distribution is situated between maxima and minima. We also know many other kinds of segment representation e.g. Hjorth parameters, Frequency analysis, Wavelet transform etc.

In general, we can we can get many different parameters and features that represents the segment. Each feature represents one dimension of the data. Since the higher dimension brings higher requirements for computation , it is important to select the most important features. There are several ways for choice of a feature selection. One of the possibilities is to use data mining software such as Weka, where you can apply one of the various feature selection algorithms (for example, Info Gain Attribute Evaluation). [8]

3.3 Dimensionality reduction

Another way to reduce a number of features is to reduce dimensionality. In the comparison to the features extraction, where the most useful dimensions are selected and used, here we take all dimensions, and project them to the new dimensions and features, that carry almost the same information, but their amount is significantly lower. One of a such projection methods is a Principal Component Analysis (PCA).

Principal Component Analysis

As mentioned above, we need to find a projection that maps $d-$dimensional space to new $k-$dimensional space where $k < d$, and the loss of information is minimal. In the case of

PCA, we maximize the variance. If $\mathbf{x}$ is data point, and $\mathbf{w}$ is the new direction, then the $\mathbf{z}$ is projection

$$\mathbf{z} = \mathbf{w}^T \mathbf{x} \tag{3.1}$$

If we mark w_1 as a first principal component then after projection of the sample it is most spread out, since the aim of this method is to maximize variance. If we want to get unique solution, then it is necessary to set $\|\mathbf{w}_1\| = 1$. Let $\mathbf{z}_1 = \mathbf{w}_1^T \cdot \mathbf{x}$ and $\Sigma = Cov(\mathbf{x})$, and the variance of $\mathbf{z}_1$ can be written as

$$Var(\mathbf{z}_1) = \mathbf{w}_1^T \cdot \Sigma \cdot \mathbf{w}_1 \tag{3.2}$$

Since we want to maximize variance of z_1 we can take it as a Lagrange problem

$$\underset{\mathbf{w}_1}{max} \mathbf{w}_1^T \cdot \Sigma \cdot \mathbf{w}_1 - \alpha \left(\mathbf{w}_1^T \cdot \mathbf{w}_1 - 1 \right) \tag{3.3}$$

By taking the partial derivative $\frac{\partial}{\partial \mathbf{w}_1}$ of the equation 3.3 and putting it equal to 0 we get following

$$\Sigma \cdot \mathbf{w}_1 = \alpha \cdot \mathbf{w}_1 \tag{3.4}$$

Previous equation holds only in a case when the α is eigenvector of Σ, and $\mathbf{w}_1$ is eigenvector of covariance matrix. Then we can say that the first principal component $\mathbf{w}_1$ is the eigenvector of the covariance matrix Σ with the largest eigenvalue $\lambda_1 = \alpha$. The similar conclusion stands also for the second principal component $\mathbf{w}_2$, which is the eigenvector of the covariance matrix Σ with the second largest eigenvalue $\lambda_2 = \alpha$. By taking k eigenvectors of Σ with k largest eigenvalues we get the new reduced space. As the last step in the projection, to the last step we need to subtract the mean value of original samples. This operation centers the data on the origin.

$$\mathbf{z} = \mathbf{W}^T (\mathbf{x} - \bar{\mathbf{x}}) \tag{3.5}$$

As displayed in the figure 3.3, PCA after centering the sample lines up the axes in the direction of the highest variance. In the case that $\mathbf{z}_2$ is too small it is ignored.

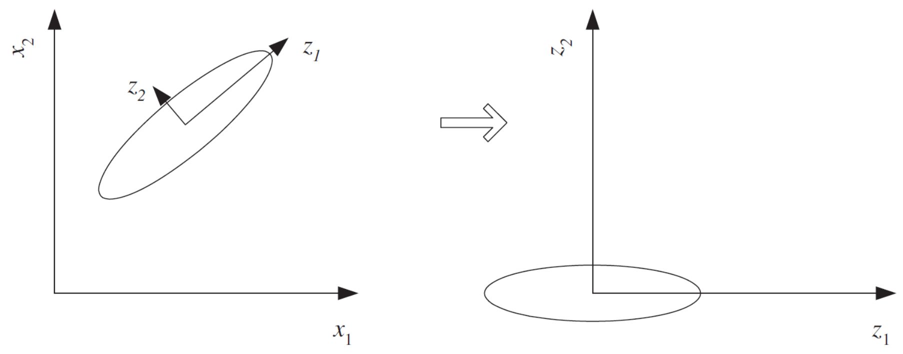


Figure 3.3: PCA geometry

As mentioned in the beginning of this subsection, the aim of this method is to project data to new reduced space with minimal information loss. To check the amount of lost information we use proportion of variance (PoV).

$$PoV = \frac{\lambda_1 + \lambda_2 + \cdots + \lambda_k}{\lambda_1 + \lambda_2 + \cdots + \lambda_k + \cdots + \lambda_d} \tag{3.6}$$

Basically, it is a summation of all used eigenvalues (since the eigenvalue is the variance explained by the eigenvector) divided by the summation of all eigenvalues of the covariance matrix Σ. [1]

Regardless of availability of such function in MATLAB, we are using own implementation of PCA. It is attached in the folder *My_PCA* and it is used before conducting each experiment for dimensionality reduction.

4 Clustering

Cluster analysis is one of the machine learning methods that finds data labels. In general, clustering is used for dividing data into groups (clusters) that are useful, or meaningful, in a desired way. Assigning particular subjects to the groups is called classification, or, as mentioned above, finding the labels for data. The main characteristic of the group is that the subjects within the group have got common or similar properties. In more technical words, it is an abstraction from individual data objects to the clusters where particular data objects belong.

Classifiers and cluster analysis have different approach for data handling. Cluster analysis labels data only based on information gathered from the data, while classifiers are firstly trained with ground truth data. When some information on the groups, such as training dataset, is available, it is possible to use classification. Such situation is called supervised learning problem. Since we have no group information on the given data, we have unsupervised learning problem which is a subject to cluster analysis or shortly clustering. Clustering is the main topic of present thesis therefore we focus only on this labeling method.

As there is not only one method for labeling, there is also not only one kind of clustering. Usually the cluster analysis is divided into two different strategies - hierarchical and partitional. As the names of the cluster analysis methods suggest, the partitional clustering divides objects into clusters that do not overlap. Every object belongs to exactly one group. On the other hand, we have hierarchical clustering that creates clusters with subsets (smaller clusters), so the groups are organized into the hierarchy. Each subset is a node that contains all of the object that are in the lower level branches of the tree that goes from it. Since the goal of the thesis is to find a hierarchical clustering method we discuss only used methods in present work.[1, 17]

4.1 Hierarchical clustering

Previous part mentions two different methods. While the partitional cluster analysis is based on the reconstruction error, hierarchical clustering uses proximity between objects (or as more widely used similarities). There are two classical approaches for hierarchical cluster analysis - agglomerative and divisive. Both of them are represented in the work by one method.

Distance and similarity function

Distance function is crucial tool in the hierarchical clustering. As mentioned above, we need to use similarities that come from the distances. Higher distance means lower similarity and vice versa, which means that for distinct objects, similarity is high. Mathematically, the distance $d(x, y)$ is a function $d : X \times X \to \mathbb{R}$ with the input that consists of two data points x and y from the set of points X that is called *space*. This function satisfies following axioms

1. $d(x, y) \geq 0$, no negative distances
2. $d(x, y) = 0$, only in the case $x = y$
3. $d(x, y) = d(y, x)$
4. Assuming that we have third point z, triangle inequality must hold $d(x, y) \leq d(x, z) + d(y, z)$

Most commonly used distance measure in $n-$dimensional Euclidean space is $L_r - norm$ (Minkowski distance) which is defined as following

$$d([x_1, x_2, \dots x_n], [y_1, y_2, \dots y_n]) = \left(\sum_{i=1}^{n} |x_i - y_i|^r \right)^{\frac{1}{r}} \tag{4.1}$$

Usually used norms are Manhattan distance, where $r = 1$ or Euclidean distance for which $r = 2$. Another generally used norm is L_∞, where the distance is taken as a maximum of $|x_i - y_i|$ over all dimensions i. There are many other types of distance measures but we are not going to discuss them since we use only $L_2 - norm$ in each approach.

Now when we know the distance function, it is easy to define similarity function. For our purposes the similarity function $s : X \times X \to \mathbb{R}$ is defined as

$$s(x, y) = 1 - d(x, y) \tag{4.2}$$

so the similarity $s \in (-\infty, 1]$. [11]

Mathematical definition

Before we proceed to the concrete cluster analysis methods, general mathematical definition needs to be given. Let us assume that we have a dataset $A = \{x_i\}_{i=1}^{N}$, where x_i is a data object and N is number of data objects and the number of clusters is k. A hierarchical clustering $\mathbf{C}$, on the dataset A, is a collection of clusters such that $C_0 \triangleq \{x_i\}_{i=1}^{N} \in \mathbf{C}$

and for each C_i, $C_j \in \mathbf{C}$ either $C_i \subset C_j$, $C_j \subset C_i$ or $C_i \cap C_j = \emptyset$. For any cluster C, if $\exists C'$ with $C' \subset C$, then there exists a set $\{C_i\}_{i=1}^{k}$ of disjoint clusters such that $\bigcup_{i-1}^{k} C_i = C$.[9]

Dendrogram

The beginning of this section mentions a representation of hierarchical clustering inn a form of tree. This tree is called dendrogram. A root node of the tree is whole dataset A. At every level of the tree, the node represents subset of the previous node in the tree (only root node does not have ancestors). The last nodes that have no followers are called leaves. Nodes between levels are connected with branches that usually have a distance between them. In this work we use binomial dendrogram, which is a type of the tree where each node has exactly 2 followers (except of the leaves). The example of the dendrogram is in the figure 4.1. Data for the dendrogram are cut from the classical iris dataset [6]. [15]

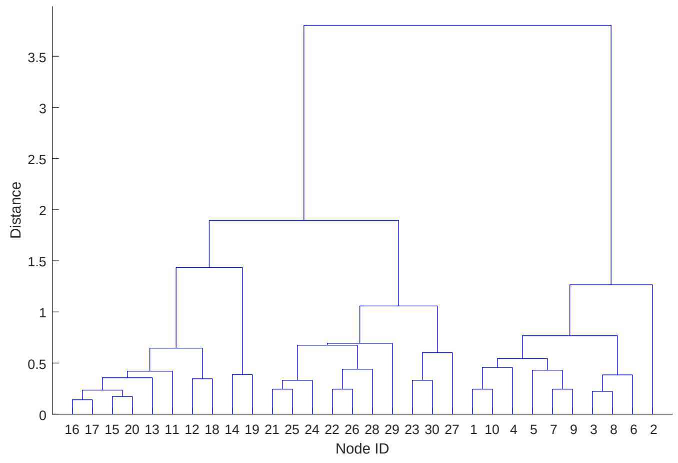


Figure 4.1: Example of dendrogram

4.2 Agglomerative Hierarchical clustering

As mentioned in the previous section, there are two possible approaches to hierarchical clustering with agglomeration being one of them. Basic idea of the Agglomerative Hierarchical clustering is following. In the beginning each data object is in one own cluster. In each step two closest clusters are merged to one cluster until all data points are not in one cluster. In the dendrogram representation, every data point is in one leaf and every cluster that was produced as a combination two clusters is node and branches represents distance between two merged clusters. More formal representation is in Algorithm 1. [17]

Algorithm 1 Basic algorithm for agglomerative hierarchical clustering

```
1: Create similarity (proximity) matrix
2: while there is more than one cluster do
3:     Group together two nearest clusters
4:     Update similarity matrix in a way that it contains new (grouped cluster) and distances between it and already existing ones. New similarity matrix does not contain two clusters that created a new one
5: end while
```

Proximity

The most important part in Agglomerative Hierarchical cluster analysis is to choose correct cluster proximity. Most common used is a distance between two clusters. For example, *Single Link* that takes minimal distance between two objects in two different clusters as a proximity between two clusters or a *Complete Link* that takes a maximal distance between any two points from those two clusters as a distance between two clusters (each point is from other cluster). However, based on the posterior knowledge, this work uses alternative technique called Ward's method.

While classical methods for measuring proximity take distance between points, Ward's method is interested in the error sum of squares (SSE). The definition of SSE for multivariate data is in the equation 4.3. This makes the method similar to k-means since the objective function is the same, but it is still a hierarchical cluster analysis. The method evaluates proximity between two clusters using increase in SSE when it tries to minimize it. So in each step two clusters are merged such the increase in SSE is minimal.

$$SSE = \sum_{i=1}^{K}\sum_{j=1}^{n_i} (y_j - \bar{y_j})^2 \tag{4.3}$$

K is actual number of clusters and n_i is number of data points in $i-th$ cluster. [17, 5]

Time and memory requirements

Before we compare algorithms on the experimental level we need to discuss differences in theoretical level. For this purpose, we use big $\mathcal{O}$ notation which represent the upper bound of complexity i.e. the worst possible case.

Basic Agglomerative Hierarchical clustering on *n* data points needs in the beginning store $\frac{1}{2} \cdot N^2$ of similarities which for the memory in the terms of $\mathcal{O}$ notation means $\mathcal{O}\left(N^2\right)$. Since in the each step the similarity matrix becomes smaller, that is the maximum memory requirement. Since we want to work with large datasets (e.g. $N = 15000$), we need to reduce this requirement in other algorithms.

Another characteristic of the algorithm in a need to track its time consumption. Although the most important parameter for us is memory, time consumption should not be too high. In the case of Agglomerative Hierarchical clustering the time complexity is $\mathcal{O}\left(N^3\right)$ where again *n* is number of data points. [17]

Implementation

Since our motivation is to find an algorithm that has better memory and time requirements for a large dataset than classical approach has, we use Agglomerative Hierarchical clustering as a reference or baseline for comparison. In this work, we use MATLAB implementation *linkage.m* to compare new approaches with classical one. The function settings in MATLAB are $linkage(X(:,1:end-1),'ward','euclidean')$. Settings for the linkage function are based on the knowledge from the paper of V. Gerla et al. [7].

4.3 Hybrid clustering

The second approach tested and compared against classical hierarchical cluster analysis is a Hybrid algorithm. It is not purely hierarchical clustering, but the output is the desired dendrogram that denotes hierarchy in the dataset. The idea comes from the work of Olga Tanaseichuk et al. described in the article *An Efficient Hierarchical Clustering Algorithm for Large Datasets* [18]. However our implementation is different from proposed in the article. The algorithm can also be described by the following picture

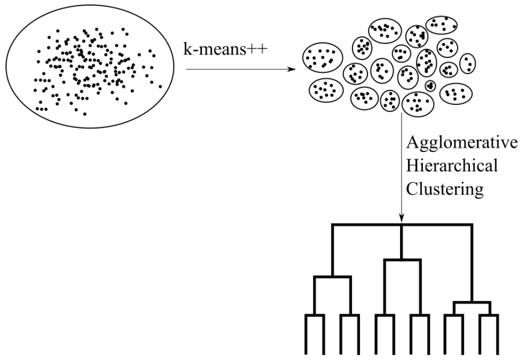


Figure 4.2: Work flow of the hybrid cluster analysis

Under hybrid clustering algorithm, it is possible to imagine method that consists of two levels. In the first step the k-means cluster analysis is done to split dataset into k clusters. In the second step, the clusters from the first step are hierarchically clustered in a way that they represent leaf clusters. For the second step we use the basic hierarchical clustering. In this section we talk only about $k-$means since classical (Agglomerative) hierarchical clustering is discussed in the section 4.2.

$k-$means and $k-$means++

As mentioned in the introduction of this subsection, in the first step of this method we use k-means which has upgraded initialization phase ($k-$means++). Algorithm $k-$means is the simplest and well known example of the partitional cluster analysis. Basically, for given dataset $\mathscr{X}$ consisting of n objects $x_i \in \mathbb{R}^d$, $i = 1, \dots, N$, we try to find k clusters centers that represent by clusters in a such way that the total squared distance ϕ between data objects and their nearest centers is minimized.

Algorithm for solving $k-$means problem was proposed by Lloyd 30 years ago in the paper *Least squares quantization in PCM* [12], and it is still widely used. Essentially, in the beginning the k initial centers are randomly chosen and each data object is assigned to one of it. The total squared distance ϕ is calculated and position of centers is recalculated. Centers represent the mass of points. Assigning of the points to the centers and recalculation of the centers is repeated until the centers stop to change.

Let us have k centers $\mathscr{C} = (c_1, c_2, \dots, c_k)$ that represent cluster $C_1, C_2, \dots, C_k$ with objective function ϕ defined as

$$\phi = \sum_{x \in \mathscr{X}} \min_{c \in \mathscr{C}} \|x - c\|^2 \tag{4.4}$$

that is minimized. Then we can write a formal representation of $k-$means algorithm in a following way

Algorithm 2 Lloyd's algorithm for k-means

```
1: Choose uniformly randomly $k$ centers
2: repeat
3:     For each data object $x_i$, $i$=1,...,$n$, find the nearest center $c_j \in \mathscr{C}$
4:     For each cluster $C_j$, $j$=1,...,$k$, update its center $c_j$: $c_j = \frac{1}{|C_j|} \sum_{x \in C_j} x$
5:     Recalculate objective function $\phi$
6: until objective function $\phi$ convergates
```

In the step 1 of algorithm 2 we can see that the initialization of the centers is done uniformly randomly. If the centers of the clusters are chosen wrongly, k-means does not return proper groups of data objects. Usually, to prevent from wrong initialization, the random choice of the centers is done multiple times. In this work we use other approach of initialization, known as $k-$means++. Its basic idea is to choose first initial center uniformly at random but others are chosen based on squared distance from the centers chosen so far.

If we denote distance between data point x and closest center c as $D(x)$ the $k-$means++ algorithm can be formally written as

Algorithm 3 k-means++ initialization algorithm

```
1: Choose uniformly at random first center $c_1$
2: for i=2 to $k$ do
3:     Choose a new center $c_i$ with probability $\frac{D(x)^2}{\sum_{x \in \mathscr{X}} D(x)^2}$
4: end for
```

When we want to use it with $k-$means, we need to replace initialization in the first step of algorithm 2 with $k-$means++ initialization. [2]

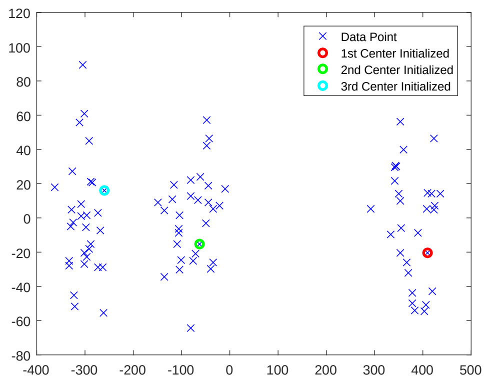


Figure 4.3: Example of k-means++ initialization

The biggest drawback of this method is sensitivity to outliers but it is still more reliable than running initialization more times. An example of successful $k-$means++ initialization is in the figure 4.3, where the first chosen center is red point, second chosen centroid is green point and the last chosen initial center of cluster has cyan color.

Time and memory requirements

As with the previous approach, it is necessary to discuss complexity of time and memory on the theoretical level. We use again big $\mathcal{O}$ notation to characterize Hybrid algorithm. In the first phase of algorithm we use $k-$means++. It is based on classical $k-$means algorithm which means that time and memory complexities have the same basis. Assuming that we have N data points in d dimensions and we want to find k clusters and we use euclidean distance, the complexity analysis is as follows.

In the each iteration we need to evaluate the distance between each centroid and data point which takes $N \cdot k \cdot d$ time, we also need to compare distances which takes $(k-1) \cdot d \cdot N$ of time and in the end we also need to compute new centroids which takes $k \cdot ((d-1)+1) \cdot N$. If we sum it up together, in the $\mathcal{O}$ notation we get following complexity for each iteration. $\mathcal{O}(k \cdot N \cdot d)$. If we assume that $k-$means algorithm finishes up in i iterations it is $\mathcal{O}(k \cdot N \cdot d \cdot i)$.

Memory complexity is strongly dependent on the dimensionality d, number of data points N, and number of clusters k since the only thing that we need to remember are data points and cluster centers in the vectorized format. It makes the memory complexity $\mathcal{O}((k+N)\cdot d)$. In the comparison to the time complexity, number of iterations does not affect memory requirement at all.

In the initialization step we use $k-$means++ initialization. Since in the initialization we evaluate distances between points and centers, and comparing distances between each other as in the clustering part of algorithm, the memory consumption is not higher than in the one with classic k-means. The same stands for the time complexity, which means that the time and memory complexity in the terms of big $\mathcal{O}$ notation does not change with $k-$means++.

In the second phase of the Hybrid algorithm we use Agglomerative Hierarchical clustering in order to get hierarchy of subclusters. Since here the subclusters are represented with k centroids, the time complexity of this step is $\mathcal{O}\left(k^3\right)$ and memory complexity is $\mathcal{O}\left(k^2\right)$. In the terms of the rules of big $\mathcal{O}$ notation and with assumption that $k \ll N$ we can state that time and memory requirements of this algorithm are $\mathcal{O}(k\cdot N\cdot d\cdot i)$, $\mathcal{O}((k+N)\cdot d)$ respectively. For more detailed analysis of Agglomerative Hierarchical clustering see subsection 4.2.

Implementation

The most crucial part of implementation of Hybrid algorithm was to implement $k-$means++. Implemented algorithm for the $k-$means++ is based on the formal algorithm 3. It is an implementation in the MATLAB. In the folder *My_Hybrid_Algorithm* all necessary scripts can be find. As in the previous algorithm, for the second level of this approach we use MATLAB function for Agglomerative Hierarchical cluster analysis implemented as *linkage.m* with the same settings as in the subsection 4.2.

4.4 Robust Active clustering

The last approach proposed in this work is a Robust Active cluster analysis. This algorithm was created and described by Brian Eriksson et al. in *Active Clustering: Robust and Efficient Hierarchical Clustering using Adaptively Selected Similarities* [4]. Unlike methods in the previous chapters, this method does not use all similarities for hierarchical clustering.

Before we proceed to further explanations, we need to define a Tight Clustering (TC) condition. Let us have triple $(A, \mathbf{C}, S)$, where $\mathbf{C}$ is we defined hierarchical clustering, on dataset A and S is similarity matrix and cluster C which is any subset of A and $C \in \mathbf{C}$. We say that the triple holds the TC condition if for every combination of x_i, x_j and x_k, such that $x_i, x_j \in C$ and $x_k \notin C$ the pairwise similarities $s_{i,j}$, $s_{i,k}$ and $s_{j,k}$ satisfies $s_{i,j} > \max\left(s_{i,k}, s_{j,k}\right)$. Consequently all similarities within the cluster are higher than the similarity between any object from the cluster and the object outside of it.

Outliers

In the introduction to this subsection we stated that the main advantage of this method is lower memory consumption achieved by not using all of the pairwise similarities. If we assume that TC condition holds, then the active clustering method that adaptively chooses similarities makes an efficient hierarchical cluster analysis possible. To run such algorithm, we firstly need to define outlier function.

Let us have tree data points as a triple (x_i, x_j, x_k), then with the assumption that TC condition is satisfied, we call data object x_k a leader or outlier, if the path from the root to x_k does not pass nearest common ancestor of x_i and x_j. More formally, the functionality of outlier can be defined as

$$outlier(x_i, x_j, x_k) = \begin{cases} x_i : \max\left(s_{i,j}, s_{i,k}\right) < s_{j,k} \\ x_j : \max\left(s_{i,j}, s_{j,k}\right) < s_{i,k} \\ x_k : \max\left(s_{i,k}, s_{j,k}\right) < s_{i,j} \end{cases} \tag{4.5}$$

Algorithm description

Above, we stated that Robust Active cluster analysis assumes that the Tight Clustering condition is satisfied. With real dataset we cannot assume it, so the assumption needs to be changed. We assume that particular part of the similarities produces correct outlier test. Such similarities are consistent with hierarchy $\mathbf{C}$, and the outlier test on them returns leader of any triple (x_i, x_j, x_k) in $\mathbf{C}$.

Essentially, algorithm quantifies how often two data points agree on outlier test. Data points are chosen from the small subset. If they agree on the outlier test, frequently they are clustered together. Formally, we can characterize Robust Active clustering as follow.

Firstly, we have to define balance factor for non-leaf cluster $C \in \mathbf{C}$ that has subclusters C_L and C_R such $C_L \cap C_R = \emptyset$ and $C_L \cup C_R = C$. Then the balance factor of C is $\eta_C = \frac{min\{|C_L|,|C_R|\}}{|C|}$. Now, let us have $\delta' \in (0,1)$, threshold $\gamma \in (0, 0.5)$ and balance factor $\eta_C \geq \eta$. By assuming that the pairwise similarity are consistent with probability at least $1-q$ (for $q \leq 1 - \frac{1}{\sqrt{2(1-\delta')}}$) and in the same time for q and η hold $\left(1-(1-q)^2\right) < \gamma < (1-q)^2 \cdot \eta$, then with the probability at least $1-\delta'$ we get correct subclusters C_L and C_R.

Previous definition is the most important part of the method. Function is called $split\,(C, m, \gamma)$, and it is defined in the algorithm 4. As we can see there is parameter m on the input of the function *split*. It is used for declaring of the maximum size of the leaf cluster and it is constrained as $m \geq c_0 \log\left(\frac{4n}{\delta'}\right)$, where $c_0 > 0$ and $n > 2m$. Small n stands for the size of the cluster C that is split. Third parameter on the input of *split* is γ that depends strongly on q and η. Based on the source paper, in practice it is user-selected parameter in the range $(0, 0.5)$.

The last function that we need to define is a function for Robust Active clustering itself. It basically uses recursion on the *split* function with initially given the whole dataset A until the subclusters are larger than $2m$. Algorithm 5 formally defines $RAcluster\,(C, m, \gamma)$.

Algorithm 4 *split* (C, m, γ)

```
Input:
    1. A single cluster $C$ consisting of $n$ items
    2. Parameters $m < n/2$ and $\gamma \in (0,1/2)$
Initialize:
    1. Select two subsets $S_V, S_A \subset C$uniformly at random (with replacement) containg $m$ item each
    2. Select a seed item $x_j \in C$ uniformly at random and let $C_j \in C_R, C_L$ denote the subcluster it belongs to.
Split:
    1. For each $x_i \in C$ and $x_k \in S_A \setminus x_i$, compute the outlier fraction of $S_V$:
       $c_{i,j} = \frac{1}{|S_V \setminus \{x_i, x_k\}|} \sum_{x_l \in S_V \setminus \{x_i, x_k\}} \mathbf{1}_{\{outlier(x_i, x_k, x_l) = x_l\}}$
       where $\mathbf{1}$ denotes the indicator function.
    2. Compute the outlier agreement on $S_A$
       $a_{i,j} = \sum_{x_k \in S_A \setminus \{x_i, x_j\}} \left( \mathbf{1}_{\{c_{i,k} > \gamma and\, c_{j,k} > \gamma\}} + \mathbf{1}_{\{c_{i,k} < \gamma and\, c_{j,k} < \gamma\}} \right) / \left| S_A \setminus \{x_i, x_j\} \right|$
    3. Assign item $x_i$ to sublucster according to
       $x_i \in \begin{cases} C_j & : if \quad a_{i,j} \geq 1/2 \\ C/C_j & : if \quad a_{i,j} < 1/2 \end{cases}$
Output:
    subclusters $C_j, C/C_j$
```

Algorithm 5 *RAcluster* (C, m, γ)

```
Given:
    1. $C, n$ items to be hierarchically clustered
    2. parameters $m < n/2$ and $\gamma \in (0,1/2)$
Partitioning:
    1. Find $C_L, C_R = split(C, m, \gamma)$
    2. Evaluate hierarchical subtrees, $\mathbf{C_L}, \mathbf{C_R}$ of cluster $C$ using:
       $\mathbf{C_L} = \begin{cases} RAcluster(C_L, m, \gamma) & : \quad if\ |C_L| > 2m \\ C_L & : \quad otherwise \end{cases}$
       $\mathbf{C_R} = \begin{cases} RAcluster(C_R, m, \gamma) & : \quad if\ |C_R| > 2m \\ C_R & : \quad otherwise \end{cases}$
Output:
    Hierarchical clustering $\mathbf{C} = mathbfC_L, mathbfC_R$ containing sublusters of size $> 2m$
```

Memory requirements

As mentioned in the introduction to this section, Robust Active cluster analysis does not use whole similarity matrix which drastically reduces memory consumption. Unluckily, there is no time complexity analysis in the source paper so we do not provide any other information on it in this section.

In the first step of this algorithm, we use function *split* that needs only $\mathcal{O}(n \cdot \log n)$ similarities where $n = |C|(C$ is cluster that is being split). By assuming that splits are balanced, we get complete cluster tree with depth equal to $\mathcal{O}(\log N)$. During the tree construction function *split* called $\mathcal{O}\left(2^l\right)$times. At level l it involves cluster of size $n = \mathcal{O}\left(N/2^l\right)$. By substituting n to the $\mathcal{O}(n \cdot \log n)$ we get $\mathcal{O}(N \cdot \log^2 N)$ which means that we need $\mathcal{O}(N \cdot \log^2 N)$ similarities in the worst case.

Implementation

Like with the previous methods, MATLAB was used to implement the defined functions. The main function is called *RAcluster.m*, and the support functions are *split.m* and *outlier.m.* Everything is attached in the folder *RAalgorithm*. The output of the *RAclustering.m* is the hierarchical tree saved in the structure. To get comparable and useful results, the function *linkage.m* was used for creating the same hierarchy as MATLAB generates. We include linkage function the comparison of the time and memory consumption .

4.5 Theoretical comparison of proposed algorithms

In the previous section we analyzed proposed algorithms in the terms of time complexity and memory consumption using big $\mathcal{O}$ notation. In the table is overall theoretical comparison of all three defined approaches.

Algorithm	**Memory**	**Time**
Agglomerative Hierarchical clustering	$\mathcal{O}\left(N^2\right)$	$\mathcal{O}\left(N^3\right)$
Hybrid clustering	$\mathcal{O}\left((k+N) \cdot d\right)$	$\mathcal{O}\left(k \cdot N \cdot d \cdot i\right)$
Robust Active clustering	$\mathcal{O}(N \cdot \log^2 N)$	NaN

Table 1: Theoretical comparison

If we assume that for hybrid cluster analysis, we have $k \ll N$ and $d \ll N$ then the memory consumption can be considered as linear in N which means that in the worst case, it is $\mathcal{O}(N)$. In that case, the hybrid clustering has the best theoretical precondition.

Mutual information

The last property of the algorithm that we check is how accurate does it find clusters. Here we use mutual information [10] which tells us how much are two variables dependent on each other. Formally, let us have two discrete variables X and Y. If we denote their joint probability is $P_{XY}(x, y)$ then mutual information $I(X, Y)$ is defined as follow

$$I(X, Y) = \sum_{x,y} P_{XY}(x, y) \cdot log \frac{P_{XY}(x, y)}{P_X(x) \cdot P_Y(y)} \tag{4.6}$$

,where $P_X(x)$ and $P_Y(y)$are marginal probabilities of discrete variables X and Y

$$P_X(x) = \sum_{y} P_{XY}(x, y) \tag{4.7}$$

$$P_Y(y) = \sum_{x} P_{XY}(x, y) \tag{4.8}$$

In this work one of the discrete variables is original labels of samples and the second one is label of cluster where does the sample belongs based on cluster analysis. Higher mutual information means more accurate cluster analysis. For purposes of this thesis MATLAB toolbox *Mutual Information computation* created by Hanchuan Peng was used.

5 Used datasets

To verify and compare all of the approaches to hierarchical cluster analysis, we need to run them over various datasets of different volumes. For the purposes of this work we have decided to use three different sets of data. First one, hypnogram, is obtained from the real EEG signal and represents relatively small dataset. The second one is also gained from real EEG record, and it is comatose data. It represents large real dataset. The last one is a set of data artificially created with clear clusters. Biological data are given in the HDF5 format which is discussed in the following section.

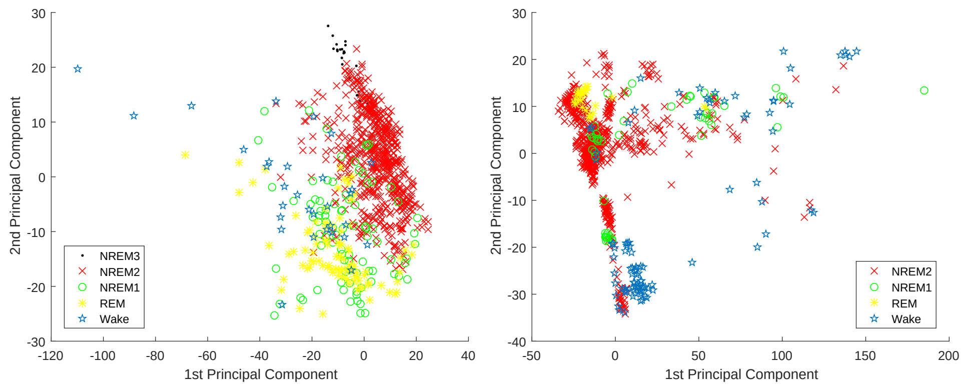
5.1 HDF5 format

Hierarchical Data Format is a file-format developed by HDF group. It is able to store various datatypes, and high volume and complex scientific data. In our case, we use this data-format to store polysomnographyc data, especially hypnogram and attributes of each segment of hypnogram. The advantage of the HDF5 format is the logical organization to data-types, data-spaces, properties and (optional) attributes. As following example shows, the structure of the file is clear and easy to read. Since MATLAB contains all necessary implementations and scripts to read HDF5 format we do not need to create our file reader or use external tool.

```
Group '/hypno-data'
    Attributes:
        'number-of-classes': 5
    Dataset 'hypnogram'
        Size: 885x1
        MaxSize: 885x1
        Datatype: H5T_STD_I16LE (int16)
        ChunkSize: []
        Filters: none
        FillValue: 0
```

5.2 Hypnogram data

In section 2.3 we discussed creation of the hypnogram and the meaning of the stages. This kind of dataset is one of possible types of datasets that we can use in the connection with EEG signals. dataset used in this work is obtained from the BioDat Research Group from CTU in Prague and their partners. Since hypnogram has various features for each channel (e.g. spectrogram or statistical features), we need to use dimensionality reduction to plot data. PCA was explained in the section 3.3, and is used to obtain first two dimensions. For two dimensions, proportion of variance $PoV = 0.50$, and as can be seen in the figure 5.1a the clusters cannot be recognized based on two principal components. In the clustering task we use PCA to reduce dimensionality to 10 for hypnogram data with.



(a) First dataset (b) Second dataset

Figure 5.1: Hypnogram dataset after PCA

In order to test algorithms, two different datasets were used (*Hypnogram_5_Cluster* dataset is in the figure 5.1a and *Hypnogram_4_Cluster* dataset is in the figure 5.1b). In each of them one data point represents 30 seconds long interval of real EEG data. Measurement of the first dataset lasted for approximately 7 hours (25350 seconds), and the second one was recorded for 7.3 hours (26550 seconds). The main difference between those two sets is the number of classes. While the first dataset has all 5 sleeps stages, the second one omits Non-REM3 phase. For the first dataset $PoV = 0.91$, and for the second one it is $PoV = 0.95$. For the visualization purposes PoV was same for both of them.

5.3 Comatose data

Another type of real data obtained from EEG are comatose stages, described also in the previous section 2.4. From our point of view, the main difference between hypnogram data and comatose data is in the volume of dataset. While hypnogram has around 900 data points in 87 dimensions, comatose dataset has 5858 data points in 2044 dimensions.

Most of the attributes (one attribute is one dimension) are statistical characteristics of the segment. Attributes for comatose data are created by combination of 13 different channels. Here the dimensionality reduction for cluster analysis was done to reduce attribute space to 100 with $PoV = 0.77$. In this case, we need to use higher dimensionality to keep information (represented by proportion of variance). As previously, for visualization are used only first two principal components with $PoV = 0.34$ that is reflected in the quality of visualization in the figure 5.2.

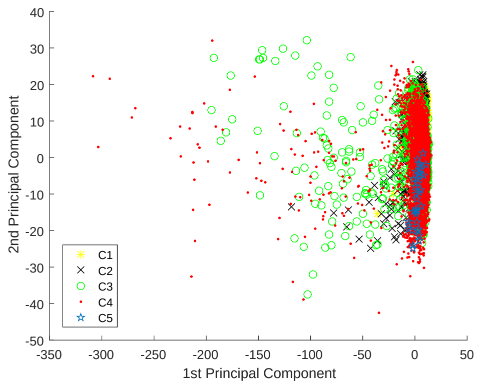


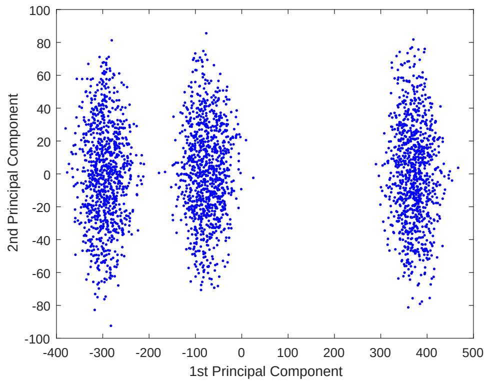
Figure 5.2: Comatose dataset after PCA

In this work we use dataset obtained from the University Hospital Na Bulovce. EEG activity was recorded using $10-20$ system at sampling frequency $128\,Hz$. The length of record is approximately 7.8 hours (28166 seconds). Signal is split using linear segmentation into 5 seconds long intervals. dataset is denoted in this work as *Comatose_Data*. Even though the fact that there are ten different comatose stages, this dataset contains only first five (C1-C5)

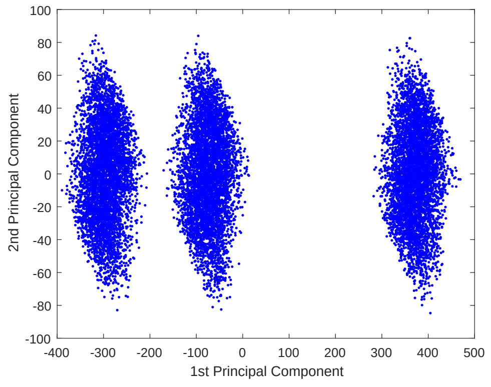
5.4 Created data

Previous sections showed us that real datasets do not have clear clusters. Based on this finding, the artificial dataset was created. As it is possible to see, we have two different datasets and they differ only in the number of data points. It was necessary to test clustering algorithms on various volumes of data since the main goal of this work is to find efficient approach that has feasible memory and time consumption requirements. Each artificial dataset contains 3 clusters that consist of the same amount N of data points in five dimensional space. Objects are generated using three basic points -

$c_1 = (1, 1, 1, 1, 1)$, $c_2 = (100, 100, 100, 100, 100)$ and $c_3 = (300, 300, 300, 300, 300)$. In the beginning N basic points of each kind are created and then the points are moved in all dimension by random number from interval $\langle -100, 0 \rangle$. Again, we used PCA for dimensionality reduction for visualization. However, in this case it was not used for the dimensionality reduction before cluster analysis.



(a) Randomly generated dataset $N = 3000$



(b) Randomly generated dataset $N = 15000$

Figure 5.3: Used datasets

For testing two different datasets were created. First one with $N = 1000$ that means the total number of data points is 3000 (figure 5.3a) is called *Random_Data_3000*. The second one with $N = 5000$ (total number of objects in dataset is 15000) is in figure 5.3b and is named as *Random_Data_15000*. The second one is considered for our purpose as a large dataset since in the reality it could represent dataset consisting of 15000 segments of 2 seconds which is possible to get if the adaptive segmentation is done over data.

6 Experiments and results

For the comparison of proposed methods we use datasets from the previous section. In this section we propose the results of various experiments conducted on the algorithms from section 4. Since the main goal of the work is to find an algorithm that is better in memory consumption and is working in reasonable time and with approximately same performance in the cluster analysis as the classical approach, we measure these properties on various sets.

To acquire memory and the time that algorithm needs, we used built in MATLAB functions. For time it is tic (start timer) and toc (end timer). In the case of memory we used profile viewer that can easily track used memory by each script. It measures peak memory of the whole script, which means that we have peak memory for each function, that is called during the algorithm. Peak memory is the maximum number bytes used by the function.

Each algorithm was run 5 times with each setting, to get correct results. Sometimes profile viewer did not record used memory (result was $0b$), so we had to run it several times and make an average over the recorded memory (zeros were left out). Time consumption was recorded also 5 times and it was averaged over it.

The last characteristic of algorithm used for comparison is mutual information, which is explained in the section 4.5. Mutual information is used to compare how well algorithms split data set to clusters. This property is used only for overall comparing of methods.

Since the real memory and time performance of the algorithm is also dependent on the computer where it runs, we decided to use only one computer. It has installed $16Gb$ of the memory, the processor works at frequency $2.4GHz$ and it has 4 cores. Operating system is $64-bit$ version as well as MATLAB.

Special case are large datasets with high dimensionality. This kind of data is represented by comatose data in this work. We do not use it for general comparison and for parameter settings but we use it to illustrate performance of proposed methods on the high dimensional data. As it is mentioned in the section 5.3, dimension is reduced from 2044 to 100 with very low PoV. We can not go lower with dimensionality because it would loose meaning to make a cluster analysis on such data.

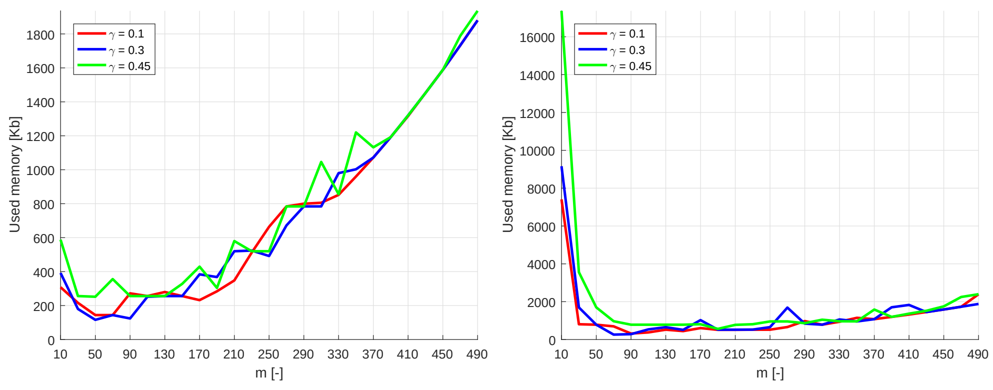
6.1 Robust Active clustering experiments

Most of the experiments were done with Robust Active cluster analysis, because it has more parameters that other two proposed approaches. For these experiments, we used four different datasets (*Random_Data_3000*, *Random_Data_15000*, *Hypnogram_4_Cluster* and *Hypnogram_5_Cluster*), with three different settings of gamma ($\gamma = 0.1$, $\gamma = 0.3$ and $\gamma = 0.45$). The second parameter, that was changed, is m, which affects the maximal size of leaf cluster. For the artificial data it was chosen from the range $\langle 10, 490 \rangle$, starting with $m = 10$ every $20th$ number was taken.

In the experiments involving Robust Active clustering we decided to also take into account the memory and time consumption of the linkage function, since it is used to rebuild the tree.

Memory consumption

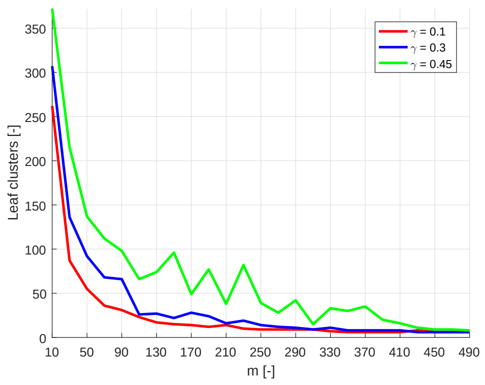
Firstly, we did experiments with artificial data, since they are large and help us to recognize how does the algorithm behaves with bulky dataset. In the figures 6.1a and 6.1b is possible to see, that the memory use for the settings with smaller m (from 10 to 130) is dependent on the γ setting. After $m = 130$ the claim for the memory is approximately same for each γ. The theoretical memory complexity has logarithm inside. By taking a look at the figure 6.1a we can see that the shape of each line representing different γ value has the shape of logarithm. It is also be possible to see the same shape for the random dataset 15000 but it is misrepresented by the higher consumption for the smallest m.



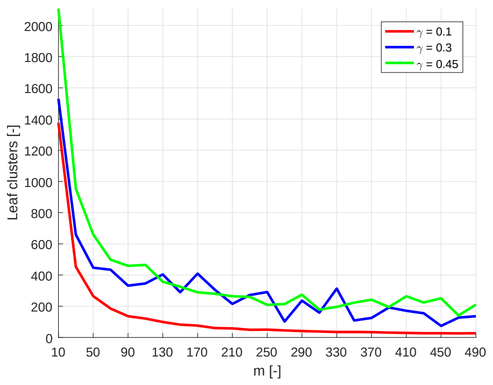
(a) Random data 3000 (b) Random data 15000

Figure 6.1: Memory consumption of Robust Active algorithm for artificial datasets

In the previous figures (figure6.1a and 6.1b) we can see that for the smallest m (from 10 to 50) the memory consumption is decreasing. It is caused by the use of the agglomerative hierarchical cluster algorithm on the leaf centroids to reconstruct the tree. In the figures 6.2a and 6.2b we can see, that for smallest m the number of leaf is quite high ($200-350$ leaf clusters for random data 3000 and $1500-2000$ leaf clusters for random data 15000). On the other hand, it is logical, since m sets the maximum size for the leaf cluster, so with small m we get more leaf clusters. It causes that the memory consumption of the whole algorithm is worse for the smallest m, and that the clear logarithmic shape can not be seen for the random dataset 15000 (figure 6.2b)



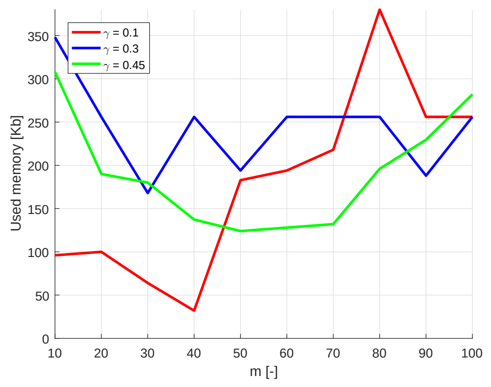
(a) Random data 3000



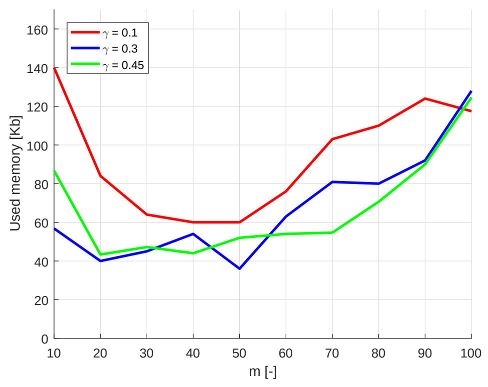
(b) Random data 15000

Figure 6.2: Number of leafs of Robust Active algorithm for artificial datasets

Real datasets are not large enough to test higher m than 100 so we decided to take as a first $m = 10$ and take every $10th$ number until 100. By comparing the results for the different settings of γ we can see, that now the best results we get with $\gamma = 0.3$ for hypnogram with 5 clusters (figure 6.3b). On the other hand, while for the hypnogram with 5 clusters the performance of algorithm is worst for $\gamma = 0.1$, for the hypnogram data with 4 clusters, based on the results in the figure 6.3a, it is the best option.



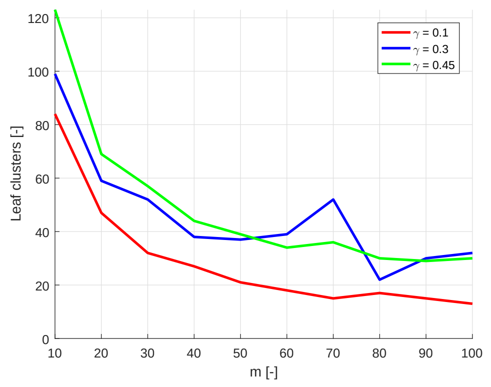
(a) Hypnogram data with 4 clusters



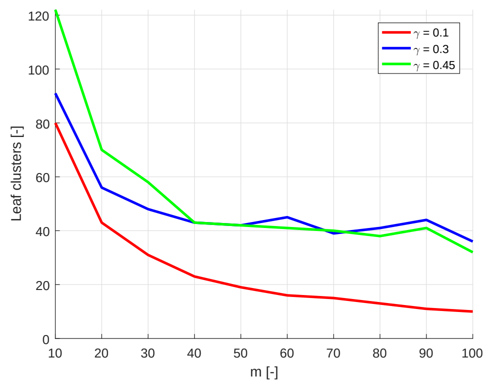
(b) Hypnogram data with 5 clusters

Figure 6.3: Memory consumption of Robust Active algorithm for real datasets

However, if we compare the real data results to the performance of the algorithm on the artificial data, we can see that the behavior of the method does not change. Although the dataset is not as large as the real one, used memory is affected by setting of *m*. The memory consumption for the smallest *m* (from 10 to 40) is higher than it is for other small settings of *m*. Again it is possible to see that number of leafs (figure 6.4) is decreasing with increasing *m*.



(a) Hypnogram data with 4 clusters

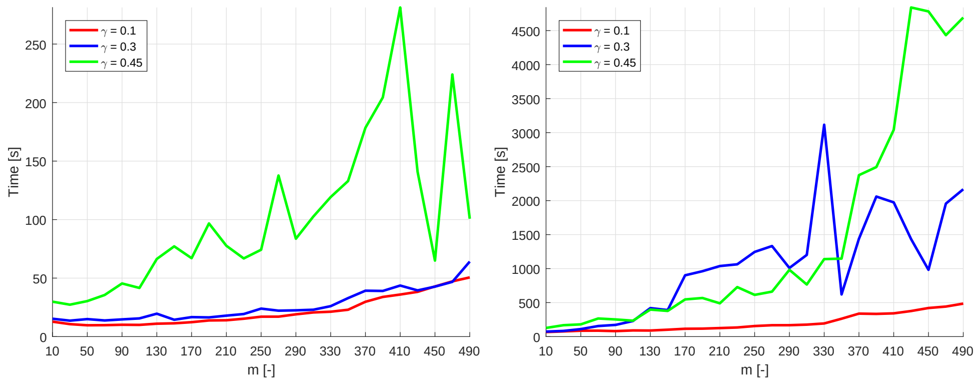


(b) Hypnogram data with 5 clusters

Figure 6.4: Number of leafs of Robust Active algorithm for real datasets

Time consumption

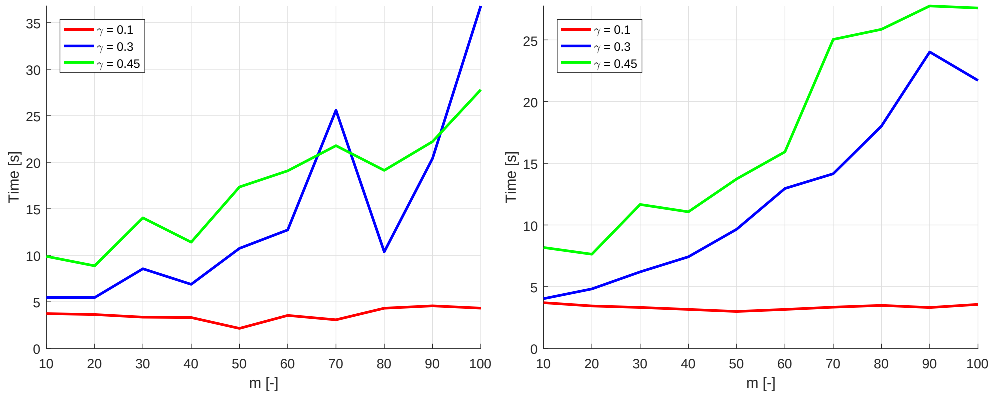
The outcomes are from the same experiments as the memory results. In the figure 6.5 is possible to see that with increasing value of m, the time needed to run the whole algorithm is also increasing. Here, the most important parameter is γ. In the figure 6.5 we can notice that for the artificial dataset with increasing γ parameter also time consumption increases, since γ has effect on the balance factor η in the tree. On the other hand, the plot in the figure 6.5b shows that for certain m (from 150 to 350) we get higher time complexity for $\gamma = 0.30$ than for $\gamma = 0.45$. Unlike memory consumption, time complexity is not affected by the linkage function which means that number of leafs has no effect on the time performance.



(a) Random data 3000 (b) Random data 15000

Figure 6.5: Time consumption of Robust Active algorithm for artificial datasets

As for the artificially generated data the same stands for the real data. With increasing γ, time consumption is increasing. Here, the interesting is running time for the algorithm with $\gamma = 0.1$. For both datasets (figures 6.6a and 6.6b), the time is more or less same of each possible value of m.



(a) Hypnogram data with 4 clusters (b) Hypnogram data with 5 clusters

Figure 6.6: Time consumption of Robust Active algorithm for real datasets

6.2 Hybrid algorithm experiments

Second proposed method was tested on the same datasets as a Robust Active clustering. Here, the only parameter that was changed is k from $k-$means++ algorithm. For *Random_Data_3000* and *Random_Data_15000* we take k from the range $\langle 10, 490 \rangle$. Starting from 10, every $10th$ number was used. For hypnogram datasets we used same range as in the experiments with previous method.

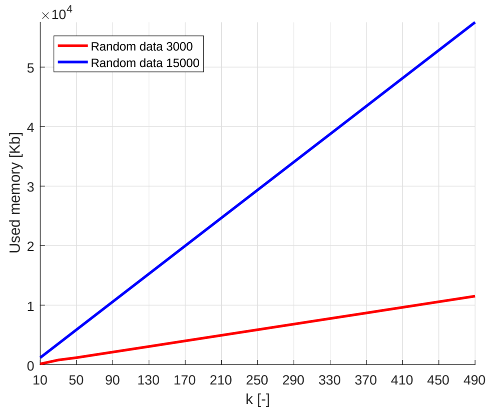
As it is mentioned in the section about implementation of Agglomerative Hybrid algorithm, which is second stage of this method, we use *linkage.m* with following settings $linkage(X(:, 1:end-1), 'ward', 'euclidean')$.

Memory consumption

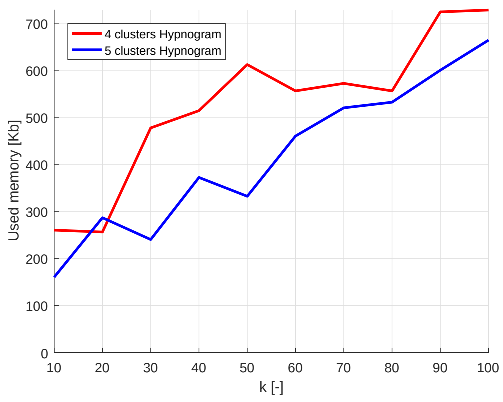
Based on the theory, in the worst case, memory complexity should be $\mathcal{O}(N)$, if we assume that $k \ll N$. Here we have two large datasets (artificially created one) and two relatively small datasets (hypnograms). Since the number of data points N does not change, the next thing that affects memory usage is number of clusters k.

If we focus on the *Random_Data_3000* and *Random_Data_15000* we can see that dependency is clearly linear (with increasing k also used memory is increasing). From the figure 6.7a is also possible to say that the with higher N we get higher memory consumption. Basically, we can say that with smaller k we get better performance, but we have to have in mind, that with k, that is too small accuracy could be bad (if we choose k too small to split dataset into enough initial clusters for the second stage of method)

Figure 6.7b shows the plot with the results from experiments with *Hypnogram_4_Cluster* dataset and *Hypnogram_4_Cluster* dataset. Although it is not as clear as with artificial datasets, we also can say that with higher k the memory consumption is also higher. Here we also can see the influence of number of data points N. While *Hypnogram_4_Cluster* has $N = 885$, *Hypnogram_5_Cluster* $N = 845$, which is small difference, but we can see that for almost all proposed settings for k, memory consumption is higher for *Hypnogram_4_Cluster* than for *Hypnogram_5_Cluster*.



(a) Random data



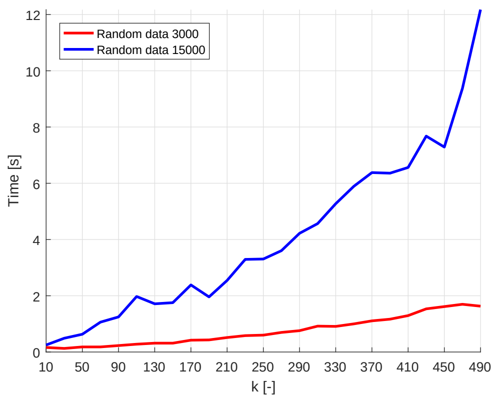
(b) Hypnogram data

Figure 6.7: Memory consumption of Hybrid algorithm

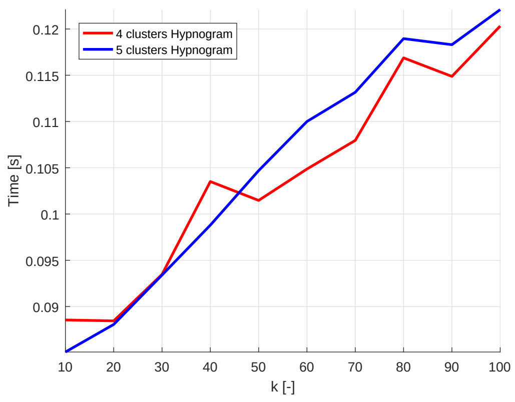
Time consumption

Like with the Robust Active clustering experiments, the same experiments for the time consumption as for the memory usage experiments were used also. Theory says, that in the worst case the memory consumption should be $\mathcal{O}(k \cdot N \cdot d \cdot i)$, which essentially means that if $k \ll N$ and we are working with low dimensional data ($d \ll N$), it depends only on the number of iterations i and number of data points N.

As it is possible to see in the figure 6.8a, for *Random_Data_3000* the memory consumption can be considered as linear and increasing only with k. If we focus on *Random_Data_15000*, we can see that, especially for the higher k, linearity is loosing, and the shape of the curve reminds more quadratic function. It is caused by the rapid increase of iterations of the $k-$means++ algorithm. Like with the memory consumption, here we also can say that lower k means shorter time but it should be again take into account that we should set k too low.



(a) Random data



(b) Hypnogram data

Figure 6.8: Time consumption of Hybrid algorithm

By focusing on the real datasets (see figure 6.8b) we can see, that nonetheless in memory consumption number of data points is reflected, here it does not play any role. In both cases the time consumption is almost the same and it is rising with the increasing k. The curves are approximately linear. Interesting finding is, that time consumption is extremely low for each dataset. The same stands for larger datasets (*Random_Data_3000* and *Random_Data_15000*), although the time is slightly higher. It is caused by larger datasets.

6.3 Agglomerative Hierarchical clustering experiments

The last algorithm that was tested is the baseline method Agglomerative Hierarchical cluster analysis. Here we do not need to set any parameters since the settings for the linkage function are given based on the work mentioned in the section 4.1(V. Gerla et al. [7].) In the table 2 are resulting performances for various datasets. Complete discussion on result can be find in the section 7.3.

Dataset	Used Memory [Kb]	Time [s]	Accuracy [-]
Random_Data_3000	35216	0.15	1.59
Random_Data_15000	880568	2.05	1.59
Hypnogram_4_Cluster	3068	0.05	0.21
Hypnogram_4_Cluster	2792	0.05	0.48

Table 2: Results of experiments with Agglomerative Hierarchical clustering

6.4 High dimensional data experiments

Real dataset might have various number of data objects, from tens to thousands. Memory and time consumption does not depend only on the volume of dataset, but also on the dimension of the data space. For example, if we take a look at the theoretical time and memory complexity of Hybrid cluster analysis, dimensionality has influence on both of them. It is possible to avoid high dimensionality by using dimensionality reduction, but we have to take into account also information carried by data in reduced dimensions.

In this work we use comatose dataset (*Comatose_Data*), that has 5858 data points in 2044 dimensions. Using PCA we reduce number of dimensions to 100 which is still too high, but proportion of variance ($PoV = 0.77$) is becoming too low. There are more convenient methods for dimensionality reduction for cases like this, such as Info Gain attribute evaluation or χ^2 attribute evaluation, but it is not contained in this work. Since the number of data points in the set is 5858 and we know that the number of clusters is 5, based on the previous results we decided to use proposed algorithms with following settings

- Robust Active clustering: $\gamma = 0.1$, $m = 40$
- Hybrid clustering $k = 30$
- Agglomerative Hierarchical clustering $linkage(X(:, 1 : end - 1), 'ward', 'euclidean')$

Recorded memory usage, time consumption and mutual information are listed in the following table

Dataset	**Memory [Kb]**	**Time [s]**	**Mutual Information [-]**
Robust Active clustering	4636	95.69	0.10
Hybrid clustering	4588	0.70	0.13
Agglomerative Hierarchical clustering	134292	1.02	0.04

Table 3: Performance of algorithm with comatose dataset

As it is possible to notice, memory consumption is for Robust Active clustering and Hybrid clustering higher with this dataset as with *Random_Data_15000* where the number of data points is almost three times higher. It is caused right by higher dimensionality which cause higher memory demands for the distance evaluation.

7 Discussion

7.1 Robust Active clustering

Based on the experiments in the previous section, we can choose the optimal settings for the parameters of the Robust Active clustering algorithm. The main parameter, that was used for deciding, is memory. Secondary, real time complexity of algorithm was used. The values are in the table 4.

Since the cluster analysis is intended to find groups of similar objects in the data set, we also need to discuss how well the algorithms find clusters. As it is mentioned in the beginning of this subsection, we use mutual information for this purpose which are listed in the table 4. We show mutual information only for the selected settings of the algorithm. However, since the random datasets have three clear clusters the mutual information is almost the same for each setting that was used.

Dataset	**m [-]**	**γ [-]**	**Mutual Information [-]**
Random data 3000	50	0.3	1.58
Random data 15000	90	0.1	1.59
Hypnogram with 4 clusters	40	0.1	0.22
Hypnogram with 5 clusters	50	0.3	0.38

Table 4: The best settings for parameters of RA clustering

As it is possible to see from the plot in the figure 6.1a, used memory for $m = 50$ and $m = 90$ is lower for this dataset. In the same figure we can see, that for both m values we get best results for two different γ values (0.1 and 0.3). By taking closer to specific numbers we find out that the best results in the terms of memory for $\gamma = 0.3$ and $m = 50$. However, in the terms of time complexity the best setting of parameters is $m = 50$ and $\gamma = 0.1$, we use values for parameters with lowest memory consumption. Difference in time between those two settings is approximately 5 seconds.

For *Random_Data_15000* it is similar as with *Random_Data_3000*. The best performance in the terms of memory usage we get for $\gamma = 0.3$ and $m = 70$. We can see (figure 6.1b), that we also get good memory requirements with $\gamma = 0.1$ for $m = 90$. On the other hand, even though we get for the first setting ($\gamma = 0.3$ and $m = 70$) memory consumption equal to $260Kb$, and for the second setting ($\gamma = 0.1$ and $m = 90$) it is $316Kb$, the time consumption is disproportionately higher for the first setting. While for $\gamma = 0.1$ it takes to run algorithm 80.83 seconds, for $\gamma = 0.3$ it is 155.60 seconds. In this case we have to take into account time complexity, so as the best setting we take $\gamma = 0.1$ and $m = 90$.

Hypnogram_4_Cluster dataset has more straightforward decision making about parameter settings. In the all terms the best results we get for the $m = 40$ and $\gamma = 0.1$. Since for that parameters the memory consumption is the lowest the time complexity did not play a big role in here. Moreover it is approximately the same for each m value for $\gamma = 0.1$ (see figure 6.6a)

The choice of the parameters for the last dataset that was used for testing Robust Active clustering (*Hypnogram_5_Cluster*) is a bit tighter. Although the time consumption is the best again for the $\gamma = 0.1$ for every m setting (see figure 6.6b) the memory usage is highest for every m. Since the memory consumption is main parameter for choosing the parameter setting we use for this dataset $\gamma = 0.3$ and $m = 50$.

7.2 Hybrid clustering

Like with the previous method, we choose the best settings for various datasets. Since the memory usage has linear character (it is linearly increasing with k) it is not the primary characteristic for parameter settings. We also have to consider time consumption and mutual information. All of the settings are listed in the following table

Dataset	**k [-]**	**Mutual Information [-]**
Random data 3000	30	1.58
Random data 15000	30	1.59
Hypnogram with 4 clusters	30	0.14
Hypnogram with 5 clusters	20	0.43

Table 5: The best settings for k parameters of Hybrid clustering algorithm

Here the situation with mutual information is similar as with Robust Active cluster analysis. For *Random_Data_3000* and *Random_Data_15000* it is almost the same for each setting of k.

Experiments in the section 6.2 show that for *Random_Data_3000* memory consumption is linear. Running time of the Hybrid clustering algorithm has linear character also. Here we need to select k with respect to the expected number of clusters. In other words, although we know that number of clusters is 3 we need to set k to higher value to obtain full and correct dendrogram from linkage function. For these reasons we use as the best setting $k = 30$. Also time consumption (0.12 seconds) is slightly lower than for $k = 10$ (0.17 seconds).

Results of experiments for *Random_Data_15000* have same characteristics as the results with *Random_Data_3000*. Memory usage is strictly linear, however time consumption is more quadratic than linear. It means that here we need to necessary choose small k. Even trough the time consumption for $k = 10$ is higher than for $k = 20$, we use it as a best possible setting. We do so for the same reason, as with previous dataset (we cannot take $k = 10$ because it can be too small for finding correct dendrogram).

With real datasets, it is usually harder to set parameters to get best possible performance, since we do not know how many clusters do we have. Here it is more important than in previous two cases to choose higher k. Although, we know that it is hypnogram, where we can have in general 5 clusters there can be a lot of unknown clusters created by different artifacts. For the *Hypnogram_4_Cluster* we decided to use $k = 20$. In the favor of $k = 20$ talks not only higher mutual information between output of cluster analysis and expert classification of hypnogram but memory consumption is also lower than for $k = 10$.

In the case of second real dataset *Hypnogram_5_Cluster* we can observe similar difficulties with the choice of the right k. Again we need to use accuracy as a helping parameter in decision making since time and memory consumption are increasing approximately linearly with increasing k. Here we decided to use $k = 30$. It would seem that $k = 20$ is better since time consumption is lower but if we focus on the figure 6.7b memory consumption higher for $k = 20$ ($240\,Kb$) than for $k = 30$ ($286.4\,Kb$).

7.3 Agglomerative Hierarchical clustering

Baseline experiments were done on the same datasets as the previous experiments with other two methods. Based on the theory memory consumption should be in the worst case $\mathscr{O}\left(N^2\right)$ and time consumption should be in the worst case$\mathscr{O}\left(N^3\right)$. MATLAB implementation of Agglomerative Hierarchical algorithm is optimized for the purposes of MATLAB, so the time consumption is lower as it can be seen in the table 2.

In the same table we also can see that real memory consumption is higher than the theoretical one. Here we can see the main drawback of the MATLAB implementation and the reason why it cannot proceed higher volumes of datasets. As an example of this drawback is possible to show memory consumption for dataset with 30000 data objects. MATLAB function $linkage(X(:,1:end-1),'ward','euclidean')$ used $3522388Kb$ which is approximately $3.4Gb$ of memory.

7.4 Overall comparison of methods

For general comparison of the methods we use the best settings for each dataset. Best settings are selected in the previous subsections. Since the main task is to find an algorithm with lower memory usage, reasonable time consumption and approximately same mutual information measures as classical approach (Agglomerative Hierarchical clustering), we take used memory by algorithm as a main characteristic in the comparison of the methods.

If we take a look at the table 6 we can see that for each dataset, that was used, memory consumption is lowest for Robust Active cluster analysis. The biggest differences are for the *Random_Data_15000* where the Robust Active clustering needs $316Kb$ which is 2787 times less memory than Agglomerative Hierarchical clustering ($880568Kb$). In addition memory used by Robust Active clustering is approximately 11 times less memory than Hybrid clustering. If we take a look at real datasets there is also significant difference in the memory consumption. For *Hypnogram_4_Cluster* dataset Robust Active clustering uses only $32Kb$ while Hybrid clustering consumes $286Kb$ and classical approach needs $2792Kb$.

Although Hybrid clustering algorithm has higher memory requirements than Robust Active clustering, we can see, that it is still better than classical approach. For example if we focus on the largest dataset Hybrid algorithm needs approximately 250 less memory than Agglomerative Hierarchical clustering, which still can be considered as a reasonable result.

Dataset	Robust Active clustering	Hybrid clustering	Agglomerative Hierarchical clustering
Random data 3000	116	784	35216
Random data 15000	316	3524	880568
Hypnogram 4 clusters	32	477	3068
Hypnogram 5 clusters	36	286	2792

Table 6: Memory consumption comparison in Kb

While in the terms of memory consumption is method using Robust Active cluster analysis unbeatable, we can definitely say that in the terms of time complexity it is the worst one. By taking a look on the table 7 we can see that Robust Active clustering needs usually from units of seconds to tens of seconds, while other two approaches need only fractions of seconds to create dendrogram and find clusters. Even through the time consumption of the Robust Active clustering is higher than the time used by other two algorithms, in reality it is still reasonable time.

The most interesting set is again the largest one that was used. If we focus on the running time of algorithms for *Random_Data_15000* we can see, that unlike for the other datasets here Hybrid clustering is the fastest one. Moreover for the *Random_Data_3000* it has almost the same time consumption. If we were using even larger datasets (for example $N = 100000$) the difference would be higher than with the dataset with 15000 data objects.

Dataset	Robust Active clustering	Hybrid clustering	Agglomerative Hierarchical clustering
Random data 3000	14.96	0.13	0.15
Random data 15000	80.83	0.49	2.05
Hypnogram 4 clusters	3.31	0.09	0.05
Hypnogram 5 clusters	9.65	0.09	0.05

Table 7: Time consumption comparison in seconds

The last, but not least parameter, that needs to be discussed is mutual information between the original labels of clusters and the results of cluster analysis. Part of the whole task in the work, is to find algorithm with approximately same performance as the classical approach. In the table 8 is possible to see an overview on mutual information measures of all three algorithms with selected settings.

The most straightforward results we get for *Random_Data_3000* and *Random_Data_15000,* where we have three clear clusters so the cluster analysis task should not be problem for algorithms. As we can see in the table 8, mutual information is for each method almost the same ($1.58 - 1.59$). Based on this, we can say that the results of the proposed methods is almost the same

More interesting are results for the real datasets. For *Hypnogram_4_Cluster* we get higher mutual information using Robust Active algorithm than using classical approach. For this dataset the accuracy of Hybrid cluster analysis was worse than using Agglomerative hierarchical clustering. On the other hand, if we focus on the result obtained from the experiments with *Hypnogram_5_Cluster* dataset, we can see that mutual information is highest for Agglomerative Hierarchical algorithm. Here, if we want to compare Robust Active clustering and Hybrid clustering, the highest mutual information, for results of cluster analysis and real labels, we get for the Hybrid clustering.

Dataset	Robust Active clustering	Hybrid clustering	Agglomerative Hierarchical clustering
Random data 3000	1.58	1.58	1.59
Random data 15000	1.59	1.59	1.59
Hypnogram 4 clusters	0.22	0.14	0.21
Hypnogram 5 clusters	0.38	0.43	0.48

Table 8: Mutual information measurement comparison

8 Conclusion

As mentioned in the begging of this work, we are focused on finding the algorithm that would use less memory than the classical approach in the reasonable time and the cluster analysis itself would be similar. By finding such method we can make an artifact detection easier, moreover it can help in the work of neurologists. We decided to compare Agglomerative Hierarchical clustering to Robust Active clustering and Hybrid method. Firstly, we found a best settings for each algorithm and the we compared them generally on the different datasets.

First discussed algorithm is Robust Active clustering, where we had to find an optimal setting for parameters γ and m for each dataset. As it is possible to see from the results in the table 4, there is nothing like universal setting for this algorithm. While for the smaller data sets (*Random_Data_3000*, *Hypnogram_4_Clusters*, *Hypnogram_5_Clusters*), we set m between 40 and 50, for the higher volume data set it is $m = 90$. It is possible to say, that for datasets with higher volumes of data object we use higher m. On the other hand, we still can say that setting of the m is straightforward and it should not be higher than 100, when the running time of the algorithm started to be too high for large data sets (see figure 6.5b). Finding of optimal setting for γ is not as clear as setting of m. By taking a look at a table 4 we can see, that in two cases it is $\gamma = 0.1$ and for other two datasets it is $\gamma = 0.3$. Here, the mutual information measurement can be helpful in the decision which γ setting is better to use.

Next proposed algorithm in this work was Hybrid algorithm, which was based on the combination of $k-$means++ and Agglomerative Hierarchical clustering. Here, we had to find an optimal setting for number of initialized clusters k. It is more clear than with Robust Active clustering, when in the table 5 we can see, that in the most of the cases is $k = 30$ (only for *Hypnogram_5_Clusters* we have $k = 20$). The most important information obtained from the experiments with this method is that k should not be too high, because with increasing k we have linearly increasing memory consumption. For high volume datasets it means not acceptable amount of used memory.

By comparing the performance of the two proposed algorithms and Agglomerative Hierarchical clustering, we can say from the first sight that both methods are obviously better than classical approach. The largest difference is in the memory consumption, when for the largest used dataset (*Random_Data_15000*) is memory used by Agglomerative Hierarchical cluster analysis 2787 times higher than memory used by Robust Active clustering and 250 times more than Hybrid clustering approach. By taking a look at a time complexity of each algorithm we can see that MATLAB implementation of classical approach is fastest method. On the other hand time used for the both proposed methods is not too high and we can consider it reasonable (see table 7). If we focus on the mutual information measurement, that represents how well methods are dividing data into clusters) we can see in the table 8 that for artificial datasets all of the methods have almost same performance. In the case of real data resulting cluster division is less accurate than Agglomerative Hierarchical clustering, but it is still good enough to say that methods are able to find clusters as good as classical approach.

In general it is possible to state, that we succeed in finding the methods, which has lower memory consumption, especially Robust Active clustering consume significantly lower volume of memory than Agglomerative Hierarchical clustering. On the other hand we get higher time consumption and worse cluster detection. In general, even trough we get lower memory consumption with new methods, each dataset needs slightly different settings to get optimal results. Based on results of this work is possible to conclude that lower memory consumption costs us other performance in other areas.

References

[1] Ethem Alpaydin. *Introduction to Machine Learning, 2nd edition*. The MIT Press, 2009. ISBN 978-0-262-01243-0.

[2] David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2007. ISBN 978-0-898716-24-5. URL `http://dl.acm.org/citation.cfm?id=1283383.1283494`.

[3] Mark F. Bear, Barry W. Connors, and Michael A. Paradiso. *Neuroscience: Exploring the Brain, 4th edition*. Lippincott Williams and Wilkins, Philadelphia, United States, 2015. ISBN 978-0-781-77817-6.

[4] Brian Eriksson, Gautam Dasarathy, Aarti Singh, and Robert D. Nowak. Active clustering: Robust and efficient hierarchical clustering using adaptively selected similarities. *CoRR*, 2011. URL `http://arxiv.org/abs/1102.3887`.

[5] Laura Ferreira and David B. Hitchcock. A comparison of hierarchical methods for clustering functional data. *Communications in Statistics - Simulation and Computation*, 2009. URL `http://dx.doi.org/10.1080/03610910903168603`.

[6] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 1936. URL `http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x`.

[7] V. Gerla, M. Macas, L. Lhotska, V. Djordjevic, V. Krajca, and K. Paul. *World Congress on Medical Physics and Biomedical Engineering, September 7 - 12, 2009, Munich, Germany: Vol. 25/7 Diagnostic and Therapeutic Instrumentation, Clinical Engineering*, chapter Wards Clustering Method for Distinction between Neonatal Sleep Stages. Springer Berlin Heidelberg, 2009. URL `http://dx.doi.org/10.1007/978-3-642-03885-3_218`.

[8] Vaclav Gerla. Automated analysis of long-term EEG signals, 2012. URL `http://bio.felk.cvut.cz/psglab/disertace/disertace-2012-02-29.pdf`.

[9] Akshay Krishnamurthy, Sivaraman Balakrishnan, Min Xu, and Aarti Singh. Efficient active algorithms for hierarchical clustering. In *Proceedings of the 29th International Conference on Machine Learning*, pages 887–894, 2012.

[10] P. E. Latham and Y. Roudi. Mutual information. 2009. URL http://www.scholarpedia.org/article/Mutual_information.

[11] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2014. ISBN 978-1-107-07723-2. URL http://www.mmds.org/.

[12] S. Lloyd. Least squares quantization in pcm. *IEEE Trans. Inf. Theor.*, 2006. URL http://dx.doi.org/10.1109/TIT.1982.1056489.

[13] Jaakko Malmivuo and Robert Plonsey. *Bioelectromagnetism - Principles and Applications of Bioelectric and Biomagnetic Fields*. Oxford University Press, New York, 1995. URL http://www.bem.fi/book/index.htm.

[14] R. W. McCarley and C. M Sinton. Neurobiology of sleep and wakefulness. *Scholarpedia*, 2008. URL http://www.scholarpedia.org/w/index.php?title=Neurobiology_of_sleep_and_wakefulness&action=cite&rev=91567.

[15] Jörg Sander, Xuejie Qin, Zhiyong Lu, Nan Niu, and Alex Kovarsky. Automatic extraction of clusters from hierarchical clustering representations. In *Proceedings of the 7th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, pages 75–87. Springer-Verlag, 2003.

[16] Fernando Lopes Silva. *EEG - fMRI: Physiological Basis, Technique, and Applications*, chapter EEG: Origin and Measurement. Springer Berlin Heidelberg, 2010. ISBN 978-3-540-87919-0. URL http://link.springer.com/chapter/10.1007%2F978-3-540-87919-0_2.

[17] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to data mining, Second Edition*. Boston: Pearson Addison Wesley, 2006. ISBN 978-0-321-32136-7.

[18] Olga Tanaseichuk, Alireza Hadj Khodabakshi, Dimitri Petrov, Jianwei Che, Tao Jiang, Bin Zhou, Andrey Santrosyan, and Yingyao Zhou. An efficient hierarchical clustering algorithm for large datasets. *Austin Journal of Proteomics, Bioinformatics and Genomics*, 2015. URL http://austinpublishinggroup.com/proteomics-bioinformatics-genomics/fulltext/ajpbg-v2-id1008.pdf.

[19] Michal Teplan. Fundamentals of eeg measurement. *Measurement Science Review, Section 2, Vol. 2*, 2002. URL http://www.ccs.fau.edu/eeg/teplan2002.pdf.

[20] David Weinstein, Leonid Zhukov, and Chris Johnson. Lead-field bases for electroencephalography source imaging. *Annals of Biomedical Engineering, Vol. 28*, 2000. URL `http://www.ccs.fau.edu/eeg/teplan2002.pdf`.