



Enhancing hierarchical clustering with local search

By:

Kjetil Monge

Olav Jensen

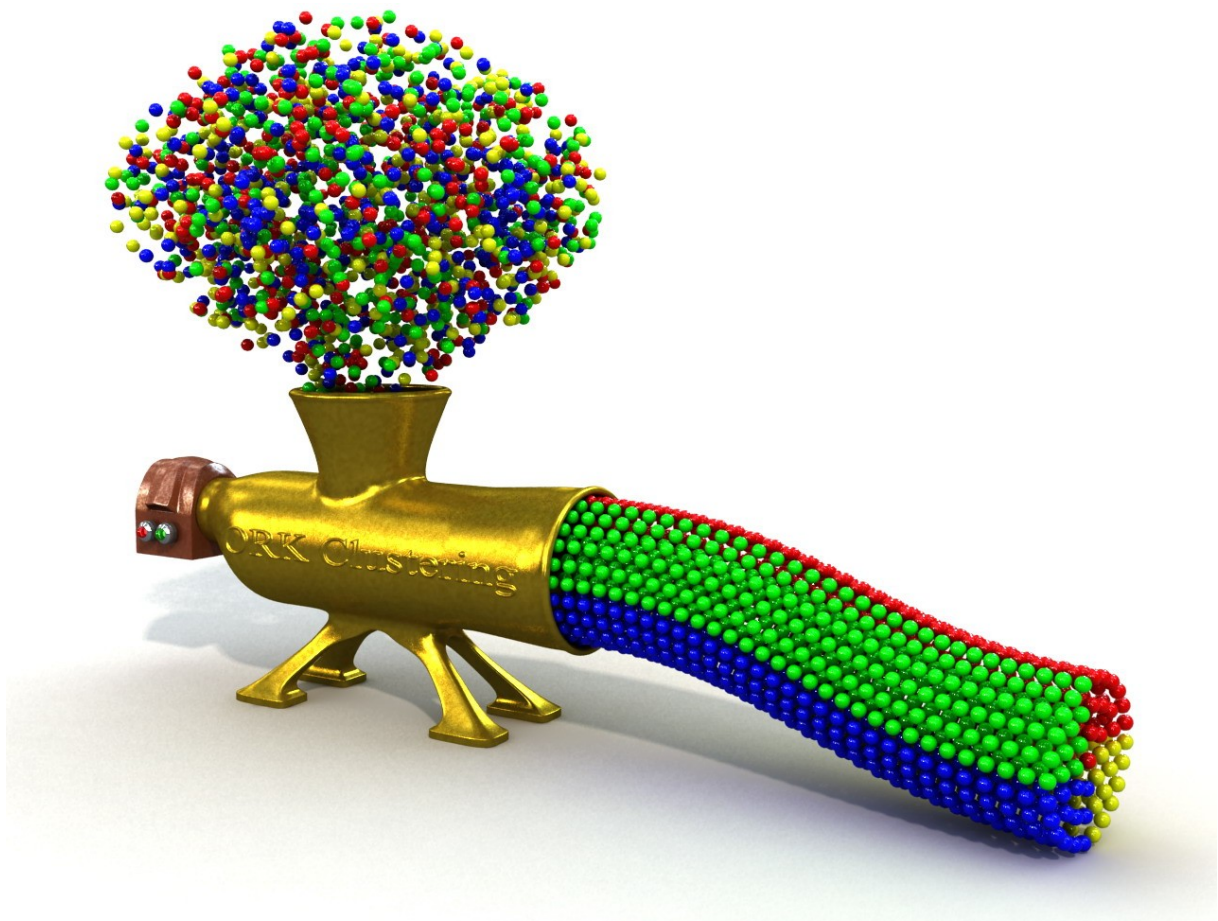
Raymond Koteng

**Thesis in partial fulfilment of the degree of
Master in Technology in
Information and Communication Technology**

**Agder University College
Faculty of Engineering and Science**

**Grimstad
Norway**

May 2007



Abstract:

Data Clustering is defined as grouping together objects which share similar properties. These properties can be anything as long as it is possible to measure and compare them. Clustering can be an important tool in many different settings varying from medical use to data mining. In this work we distinguish between two different types of clustering. The simplest one, called partitional clustering, tries to create one solution by comparing objects and partitioning them into non-overlapping groups. The second one, called hierarchical clustering, is a bit more complex and will try to generate a multi-layered solution instead. In this multi-layered structure, groups actually consist of more than one sub-group from a lower layer. The structure is often and probably best thought of as a tree-structure. The only way to get, and ensure, a “perfect” solution is to use a brute force clustering algorithm which works by comparing all possible solutions against each other. We do not implement a brute force method; instead we use another algorithm which generates very high quality solutions.

The goal of this study was to develop an algorithm for hierarchical clustering with the ability to solve large problems quicker than the alternative existing solution. The approach to this was to make the algorithm work with only parts of the problem area at any time. In this report this is called: “local search”.

Some other important concepts also explained and used in this report are local/global optimum, online/offline clustering and partitional/hierarchical clustering.

The already existing algorithm briefly mentioned serves as a “template” which we use to compare and test our developments against. This testing has been performed using an application we have developed specifically for this purpose.

The results from our experiments chapter hints that the approach we have taken when developing the algorithms is very much on the right track. Our algorithms perform high quality clustering within decent time. Compared to the “state of the art” algorithm we have tested them against they provide, on large problems, somewhat less quality solutions, but at a much better time. That being said we still think there may be room for improvements in any future work.

.

Preface

This master thesis is our final work at the Master of Science program at Agder University College, Faculty of Engineering and Science. Finishing this thesis gives us the degree Master of Science in Information and Communication Technology. This master thesis is done during the spring semester and has a weight of 30 ECTS.

Writing this thesis proved to be a bit of a challenge. A lot of unforeseen problems got revealed during the work, but this in itself was expected and we did manage to stay reasonable close to our work schedule. Even if there were some unexpected problems and results we feel we have reached a good result, why this will hopefully be answered in this report.

We have a small number of people which we want to thank for helping us during the project work.

First of all we want to thank our problem holder and supervisor Ole-Christoffer Granmo for excellent guidance and interesting discussions during this entire thesis.

Second we want to thank Andreas Häber for providing us with a fast and stable server to test all our experiments and for helping us with Linux support.

Last we want to thank the school administration who helped us with a lot of practical issues like report guidelines, paper submission, printing etc.

Olav Jensen
Kjetil Monge
Raymond Koteng

Version Control

Version¹	Status²	Date³	Change⁴	Author⁵
0.1	Draft	16.01.2007	Initial start	Raymond
0.1	Draft	16.01.2007	Started merging reports for other projects and added new chapters.	Raymond
0.1	Draft	22.01.2007	Added more info to chapters	Raymond
0.1	Draft	02.02.2007	Completed chapter one and two	Raymond
0.2	Draft	19.02.2007	Merged chapters written by Raymond and Kjetil into one document.	Raymond and Kjetil
0.2	Draft	28.02.2007	Chapter 1 and 2 as good as finished	Raymond
0.3	Draft	01.03.2007	Chapter 3 and 5 as good as finished	Olav and Kjetil
0.3	Draft	12.03.2007	Rewrote chapter 4	Raymond
0.4	Draft	01.04.2007	All chapters has "final" elements	Olav, Kjetil and Raymond
0.5	Draft	15.04.2007	Results from experiments are now in the report	Olav and Raymond
0.7	Draft	24.04.2007	Description of algorithm completed	Kjetil
0.8	Review	08.05.2007	Sending a review to supervisor	Raymond
0.9	Review	16.05.2007	Changed text after feedback from supervisor	Olav, Kjetil, Raymond
0.9	Review	18.05.2007	Sending a last review to supervisor	Raymond
1.0	FINAL	29.05.2007	Status final	Olav, Kjetil and Raymond

-
- 1 **Version** indicates the version number starting at 0.1 for the first draft and 1.0 for the first review version.
 - 2 **Status** is DRAFT, REVIEW or FINAL
 - 3 **Date** is given in ISO format: dd-mm-yyyy
 - 4 **Change** describes the changes carried out since the previous version
 - 5 **Author** is the one who did the change

Table of Contents

1	Introduction	10
1.1	Background	10
1.2	Thesis definition.....	14
1.3	The importance of the project	15
1.4	Problem statement.....	15
1.5	Sub problems	15
1.6	Delimitations and assumptions	16
1.7	Hypothesis.....	17
1.8	Risks for the project	17
1.9	Structure of the thesis.....	17
2	Review of literature.....	18
2.1	Machine learning	18
2.2	Dendrogram.....	18
2.3	Mean and SSR	19
2.4	Clustering	21
2.5	Partitional clustering	22
2.6	Hierarchical clustering.....	23
2.7	Offline cluster algorithms	24
2.8	Online cluster algorithms	24
2.9	Fuzzy clustering.....	24
2.10	Agglomerative cluster.....	25
2.11	Different Clustering algorithms	26
2.11.1	Popular Clustering algorithms	26
3	Research.....	28
3.1	Problem classification	28
3.2	Hypothesis approach	28
3.3	Application functionality	29
3.4	Programming platform	29
3.5	Programming	30
3.6	Design	30
3.6.1	Framework.....	30
3.6.2	Validation.....	31
3.7	Organization for our master thesis	32
3.7.1	Work process.....	32
3.7.2	Time schedule	33
3.7.3	Critical resources	34
4	Overview of our application	35
4.1	Graphical User Interface	35
4.1.1	Main interface	35
4.1.2	Settings panel.....	36
4.1.3	Representing of the clusters	40
4.1.4	Tree structure for hierarchical clustering	41
4.1.5	GUI for creating custom clusters.....	42
4.1.6	Visual representing of the clustering results	43
4.2	XML.....	44
5	Approach for partitional clustering	46
5.1	Partitional clustering algorithm.....	46
5.2	Different partitional cluster algorithms approaches.....	47
5.3	Experiments.....	48
5.3.1	Setup.....	48
5.3.2	Results	49
5.4	Local optimum experiment.....	54

5.5	Experiments summary	56
6	Approach hierarchical clustering.....	58
6.1	Online hierarchical clustering algorithm	58
6.2	Different hierarchical cluster algorithms	60
6.2.1	Algorithms for initializing the tree	62
6.2.2	Clustering algorithms	66
6.3	Description of algorithms	68
6.3.1	Clusters switcher	68
6.3.2	Cluster mover	70
6.4	Experiments hierarchical clustering.....	74
6.4.1	Setup	74
6.5	Empirical results	77
6.5.1	Results from 4 clusters 20 points without noise	78
6.5.2	Results from 4 clusters 20 points with noise	80
6.5.3	Results from 4 clusters 100 points without noise.....	82
6.5.4	Results from 4 clusters 100 points with noise	84
6.5.5	Results from 4 clusters 1000 points without noise.....	86
6.5.6	Results from 4 clusters 1000 points with noise	88
6.5.7	Results from 4 clusters 4000 points without noise.....	90
6.5.8	Results from 4 clusters 4000 points with noise	92
6.5.9	Results 400 points, different levels	94
6.6	Experiments summary	96
6.7	Scaling of the algorithms.....	97
7	Conclusion	99
8	Appendix.....	100
8.1	References	100
8.2	Attachments.....	102

Table of figures

Figure 1 Example of clustering	10
Figure 2 Example of data tree	12
Figure 3 Example of divisive clustering.....	13
Figure 4 Example of agglomerative clustering	14
Figure 5 Example of a dendrogram	19
Figure 6 The calculation of mean.....	19
Figure 7 The calculation of clusters SSR.....	19
Figure 8 The calculation of total SSR	19
Figure 9 Update SSR when adding a point.....	20
Figure 10 Update mean when getting a point	20
Figure 11 Update mean when losing a point.....	20
Figure 12 Update SSR when losing a point	20
Figure 13 Explanations of the letters in the formulas	20
Figure 14 Hierarchical clustering	23
Figure 15 Agglomerative clustering	25
Figure 16 Closest agglomerative and lowest SSR tree comparison.....	27
Figure 16 Shows our time schedule.....	33
Figure 17 Shows the main interface for our application	35
Figure 18 Shows the settings panel.....	39
Figure 19 Shows 400 points clustered into 9 different cluster levels	40
Figure 20 Shows tree structure for 100 points	41
Figure 21 GUI for creating custom clusters.....	42
Figure 22 Shows the result from clustering.....	43
Figure 23 Shows the XML structure of two clusters	44
Figure 24 Shows the basic XML structure of two clusters.....	45
Figure 25 Pseudo algorithm.....	46
Figure 26 Compares results of 3 algorithms clustering 4 clusters and 400 points	49
Figure 27 Compares results of 3 algorithms clustering 4 clusters and 2000 points	50
Figure 28 Compares results of 3 algorithms clustering 10 clusters and 1000 points	51
Figure 29 Compares results of 3 algorithms clustering 10 clusters and 5000 points	52
Figure 30 Compares results of 3 algorithms clustering 60 clusters and 6000 points	53
Figure 31 The local optimum problem	54
Figure 32 Settings to solve local optimum	54
Figure 33 Local optimum problem solved	55
Figure 34 Online hierarchical clustering algorithm	59
Figure 35 Example of a Hierarchical tree.....	61
Figure 36 Dendrogram example	61
Figure 37 Cluster switcher.....	68
Figure 38 Example of a dendrogram switch.....	69
Figure 39 Cluster mover	70
Figure 40 Move to different sub tree.....	71
Figure 41 Move to same sub tree.....	72
Figure 42 Example of experiment with 4 clusters.....	74
Figure 43 Example of experiment with 12 clusters.....	75
Figure 44 20 points, without noise - the natural clusters	78
Figure 45 20 points, without noise - hierarchical tree before and after clustering	79
Figure 46 20 points, without noise - graph showing the clustering progress	79
Figure 47 20 points, with noise - the natural clusters	80
Figure 48 20 points, with noise - hierarchical tree before and after clustering.....	81
Figure 49 20 points, with noise - graph showing the clustering progress	81
Figure 50 100 points, without noise - the natural clusters	82
Figure 51 100 points, without noise - hierarchical tree before and after clustering	83

Figure 52 100 points, without noise - Graph showing the clustering progress.....	83
Figure 53 100 points, with noise - the natural clusters	84
Figure 54 100 points, with noise - hierarchical tree before and after clustering.....	85
Figure 55 100 points, with noise - Graph showing the clustering progress.....	85
Figure 56 1000 points, without noise - the natural clusters	86
Figure 57 1000 points, without noise - hierarchical tree before and after clustering	87
Figure 58 1000 points, without noise - Graph showing the clustering progress.....	87
Figure 59 1000 points, with noise - the natural clusters	88
Figure 60 1000 points, with noise - hierarchical tree before and after clustering.....	89
Figure 61 1000 points, with noise - Graph showing the clustering progress.....	89
Figure 62 4000 points, without noise - the natural clusters	90
Figure 63 4000 points, without noise - hierarchical tree before and after clustering	91
Figure 64 4000 points, without noise - Graph showing the clustering progress.....	91
Figure 65 4000 points, with noise - the natural clusters	92
Figure 66 4000 points, with noise - hierarchical tree before and after clustering.....	93
Figure 67 4000 points, with noise - Graph showing the clustering progress.....	93
Figure 68 400 points, different levels - the natural clusters	94
Figure 69 400 points, different levels - hierarchical tree before and after clustering.....	95
Figure 70 400 points, different levels - Graph showing the clustering progress.....	95
Figure 71 Two different dendrograms.....	96
Figure 72 Graph showing the scaling of the algorithms	98

1 Introduction

The introduction presents the background for the report and links this to our thesis definition and our problem statement. The background also briefly explains some information needed to understand the other chapters.

From the problem statement we derive our sub problems and definitions of delimitations and assumptions. These definitions are explained with an attempt to justify them. Finally this leads us to hypotheses and research question.

1.1 Background

Data clustering

Data clustering (or simply clustering) is the process of grouping together a collection of unlabeled objects so that the objects that is most similar or has the most in common will be placed in the same group. This could for instance group together any number of points so the generated clusters contain only points which lie close to one another. The similarity does not need to be distance but can be any properties of an object which can be measured and compared. This has to be defined based on what the goal of the clustering is. For instance, there would be no problem to use color as a measurement of similarity, as long as we also define which colors lies close to each other and which lies farther apart.

Below is an example of clustering on a set of 100 points. The left side of the figure shows the points randomly distributed between two clusters. The results after clustering are shown at the right. Here we can see that the algorithm has found the structure of the clusters and correctly organized the data by putting the points in the correct groups

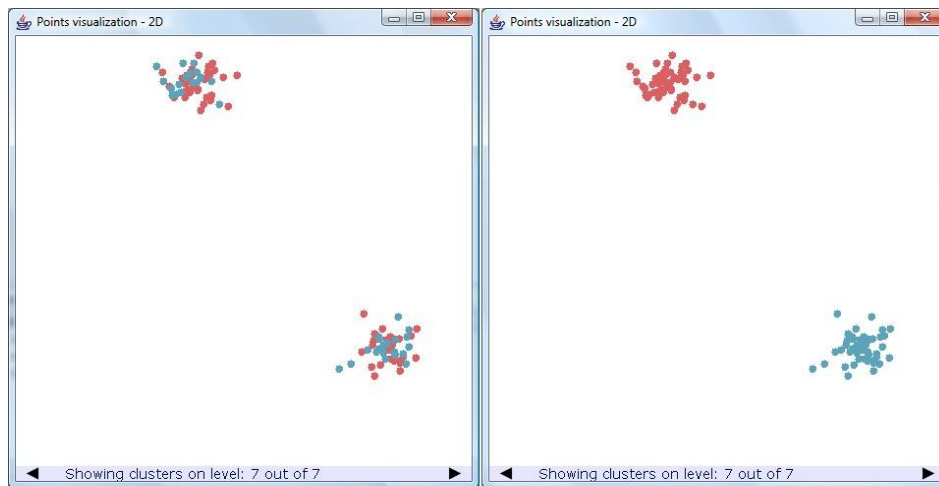


Figure 1 Example of clustering

Usage of clustering

Data clustering is an important tool in many different areas. It is a part of computer science often called machine learning, which is closely related to the study of artificial intelligence. In machine learning, data clustering helps organize the data and eases the process of looking for similarities. This makes clustering usable, amongst other, in the field of data mining and search engines. One example of a search engine which relies heavily on clustering is Clusty [21]. Clusty uses clustering in such a way that the results delivered are put into groups (clusters) based on how similar they are. As a test we took a search for “Drinks”. And we were given topics like “Soft drinks”, “Wine”, “Alcoholic Drinks”, and “Energy Drinks” etc. as groups of search hits.

Clustering can also be of use in computer image recognition tools and medical science. If you have x-ray images of skulls, for instance, those images could be clustered based on color. This could help a computer to recognize familiar patterns such as early stages of tumors, which again could help locate such things even if the x-ray was taken simply to check out if the skull had fractures or not.

Types of clustering

The two kinds of clustering important for this study are partitional and hierarchical clustering.

In partitional clustering the goal is to get one partition of objects containing a static number of clusters

In hierarchical clustering the goal is to create a structure containing more information than a simple partitional solution. The result is often displayed as a tree structure. Both rough and fine grained solutions will be available through this tree structure. To reach finer granularity of any part of the tree, one will only need to examine the child nodes placed further down the structure. The search engine “Clusty” for example, will give results in groups based on similarities instead of just a list of hits like “regular” search engines usually does. A group of hits might also contain sub-groups which in a tree structure will be the children of a cluster.

While hierarchical clustering in most cases would require much more work than partitional algorithms, the aim is that in the end, an analysis of the tree will give better overall understanding, and therefore make it possible to generate solutions of better quality which also can also be used as a solution for a broader set of areas.

Online and offline clustering

We also distinguish between online and offline clustering. In an offline algorithm the entire input set will be examined, and used for each step of the clustering. This could in some cases be the only step, while in other algorithms it might be one in a series of steps while working towards a final solution. An example of an offline algorithm is the well known K-means. K-means is a partitional algorithm where all the objects are examined to calculate new clusters which are then used as starting point for the next step, improving it more and more for each step.

Online algorithms perform changes while focusing on the ability to have a currently available solution even during the work process. An example of an online partitional algorithm is the “one point algorithm” which we have been studying as an introduction to clustering (described in detail in chapter 5).

Online algorithms typically cluster by iteratively performing small changes to an already existing cluster structure. These changes should be kept so small that one step will not majorly disrupt the current solution already contained in the cluster structure. Large changes are made as the result of many of these small steps.

Local and global search

Sometimes, especially when dealing with hierarchical clustering, one small change can affect large parts of the structure. In these cases, assessing the effect of a change may be computationally costly, limiting scalability. The more data the algorithm has to take into consideration, the worse the algorithm would scale as the problems grow in size.

By scaling we mean how well the algorithm handles additional objects and/or clusters in the input set. Local and global search describes if the algorithm can perform changes without having to look at the entire, “global”, structure. Ideally all local search algorithms should have only a very small sub-structure to worry about during clustering, but how much of the structure it will have to work with is dependent on the actual algorithm. Global algorithms will not try to limit its work in this way.

Data tree

In computer science a data tree is a structure used to store data. A tree structure will have nodes containing some sort of data with relations between nodes based on relations in the data. The top of the tree will usually contain a single node called the top-node or the root of the tree. All nodes have relations to other nodes. Nodes with direct relation to a lower node will be the parent of that child. All child nodes will have one or more parent, but usually one and only one. The lowest nodes in the structure are called leaf-nodes. These nodes will have no children, only parent(s). Any top nodes will have no parent, only children.

A layer in a data tree is the set of all the nodes which are the children of the nodes on the previous layer. The first layer will consist of only the top node(s). The second will be the children of the top node. The third will be the children of the nodes on layer two etc. With this definition of a layer each layer will contain all the data already in the root node but distributed amongst different amounts of nodes. So in a way when it comes to clustering, each layer will represent one partitional solution. To get solutions not represented on a layer a combination of parts of layers have to be used.

Figure 2 shows an example of a general data tree.

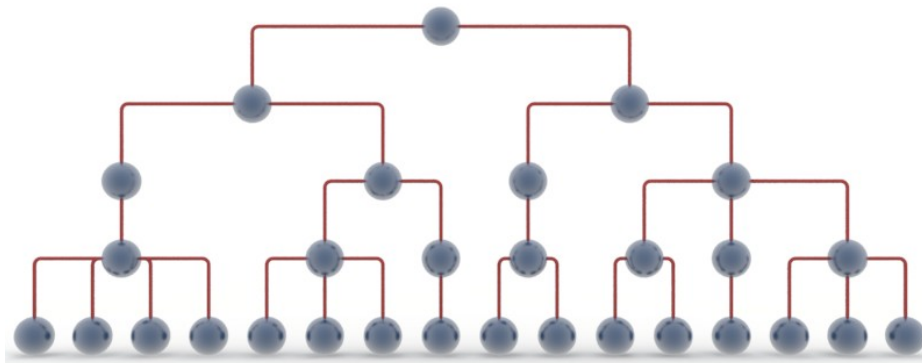


Figure 2 Example of data tree

Divisive clustering

When building a hierarchical structure there are two main ways of doing this. A divisive method, also called top-down clustering, will start with objects in few groups. Often all data will initially be one single cluster. The algorithm will then divide these clusters into two or more clusters each, which again will be divided into smaller clusters and so on until there is one cluster for each object. Divisive clustering is rarely used because it is hard to define and implement a way of dividing clusters so that the end result would be a clustered tree. To generate a random tree using the divisive method is simple though.

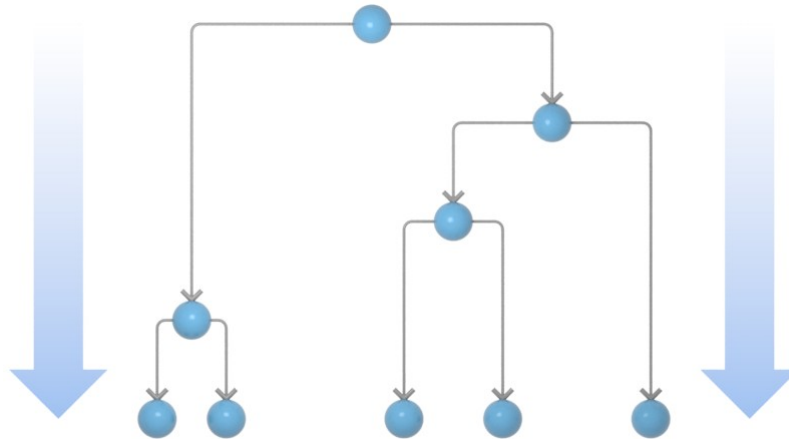


Figure 3 Example of divisive clustering

Agglomerative clustering

Agglomerative clustering, also known as bottom-up clustering, will have start with cluster for each object in the input set. To create new clusters these clusters will then be merged two by two until the top is reached. At the top the clusters are finally merged into one single cluster. If the two clusters which are merged each step of the algorithm are chosen based on similarities, the structure will be clustered when the tree is finished being built. The standard algorithm which we test our own algorithms against in the experiments section is an agglomerative clustering algorithm. This algorithm clusters the input set by always selecting the two clusters which are closest to one another when choosing clusters for merging. In this report we call this method the “closest agglomerative” clustering method. The figure below shows five clusters clustered with closest agglomerative clustering. Each of the nodes in the tree represents a cluster. The distance between the nodes is not the actual distance between the clusters, but is a representation of the distance projected to one dimension. The first two nodes merged are the ones with the shortest distance between them; the second are the clusters which then have the shortest distance etc.

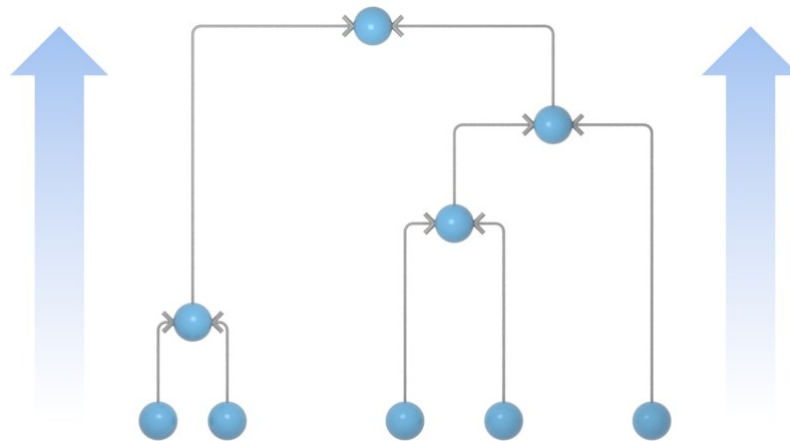


Figure 4 Example of agglomerative clustering

1.2 Thesis definition

This thesis will investigate the possibility of hierarchical online cluster algorithms with a local search approach. Our final definition of our thesis is formulated as this:

“The purpose of this thesis is to develop a scheme for hierarchical online clustering. The goal is to combine the main benefits of local search (e.g., clustering speed) with the benefits of a hierarchical approach (e.g., different levels of cluster granularity). The resulting scheme will be evaluated with a particular emphasis on clustering accuracy, processing speed, and ability to find global optima. A benchmark framework will be developed for evaluation purposes.” [17].

1.3 The importance of the project

The “closest agglomerative” algorithm gives high quality solutions. But its uses are a bit limited because of its inability to scale. Some areas where clustering can be of great use requires clustering of very large amounts of data, so a quicker solution might be required. Also many of the available hierarchical clustering algorithms are offline algorithms. This can be a limitation in many areas. In fields where additional information might become available after clustering has commenced, this information can be added as an object to a random cluster and an online algorithm can keep clustering without having to stop or restart at all. An offline algorithm would not have this luxury. Therefore a highly scalable online algorithm is a very diverse algorithm with a very wide are of use.

1.4 Problem statement

The purpose of this thesis defined in our thesis description is to develop a scheme for hierarchical online clustering. The goal is to combine the benefits of local search with the benefits of hierarchical approaches.

The scheme will be evaluated with a particular emphasis on clustering accuracy, processing speed, and ability to find global optima. One of the main benefits of partitional clustering compared to hierarchical is speed. There are several algorithms available which gives good results and does so in decent time. One of the major problems with partitional clustering is that they only provide one solution possessing a fixed numbers of clusters. If finer or rougher granularity is desired, a completely new round of clustering has to be done. Hierarchical clustering, on the other hand, generates a set of solutions with easy access to different layers of solutions.

The benefits of both partitional and hierarchical clustering combined without the disadvantages could probably be very useful in several areas of machine learning. To actually test and verify anything we do we need to have a testing framework available.

1.5 Sub problems

From our problem statement we have defined our sub problems.

Limited visibility when working with a “local search solution”

Since we want the algorithm to be an online algorithm, a small operation step will be performed a high number of times. Because of this even a small decrease in workload for each step may decrease the total work time substantially. In a hierarchical system, changes made will automatically affect potentially large parts of the structure. We have called part of the structure affected the “minimal local area”. Our first sub problem will be to find this minimal local area so that time will not be wasted on unnecessary changes to areas outside this.

We will have to have a definition of quality of a cluster.

The algorithm should perform small changes which will bring the problem closer to optimal solution. But to perform these steps we need to have a way of identifying which changes will contribute to a better solution and which will not. To do this we want to define quality for a cluster and measure that quality before and after a change. Changes which contribute positively to the solution will then be identifiable by looking at the change in quality. How to define this quality measurement, so that the calculation of it remains within the minimal local area of the structure, will be an important part of our research.

Finding global optimum

Global optimum is the single solution which cannot be improved any further. Which solution is the global optimum is dependent on the quality definition.

Since the thesis definition states that focus should be on finding a solution close to global optimum we will need to have a way of identifying the global optimum solution or a solution as close as possible to the global optimum.

Defining quality for a hierarchical solution

After finding the optimal solution we should also be able to compare the quality of our clustering against the quality of the global optimum state. When comparing solutions it is important to see how similar they are and not only if one is better than the other. So we need to define quality for a hierarchical structure.

A suitable framework for testing environment is needed

We need a tool which enables us to extract research data and to compare the different algorithms. The requirements of this application are:

- Generate problems, either random or let the user manually create them.
- Store and reopen problems.
- Should support several different cluster algorithms.
- Should have an interface to make implementing of new algorithms easy and effortless.
- Provide a visual representation of the problem both before and after clustering.
- Store statistical data while running an algorithm.
- Show the user graphs and/or tables of the statistical data.
- The framework should be as intuitive and simple as possible.

1.6 Delimitations and assumptions

To focus our work only on important areas we have a few delimitations and assumptions.

- Limit our tests to two or three different implementations of both hierarchical and partitional cluster algorithms.
- Define the type of tree structure for hierarchical solutions.
- Input data for test should only be points in a two dimensional space.
- Distance between clusters should be the Euclidian distance between the clusters middle points.
- Input data should have no apparent structure prior to clustering other than each points location.
- SSR should always be used when comparing different algorithms..
- In order to have a controlled environment to test the algorithms in, we will use “artificial” data, generated randomly.

1.7 Hypothesis

From our problem statement and sub problems we have derivate the following hypothesis:

Hypothesis 1:

“Quick selection of random changes will result in better scalable algorithms than spending time on finding a “best change” scenario.”

Hypothesis 2:

“Minimizing evaluation and updates to only the vital parts of the structure will improve scalability significantly.”

Hypothesis 3:

“An online hierarchical algorithm with local search will scale better than the closest agglomerative approach.”

The approaches for assessing the hypotheses are discussed in the research chapter 3.2.

1.8 Risks for the project

There are some things we need to make sure of to get good and proper results.

- Need to make sure we follow our own limitations to assure that we are only spending time on the core of the problem.
- Perform our work and do our writing so that the results and conclusion is valid however the experiments turn out.
- To get a scientific approach we have to make sure we properly document and describe the entire work process with focus on making it reconstructable.

1.9 Structure of the thesis

- Chapter 2
 - Repetition of the definition of clustering
 - Discussion of important properties for clustering
 - Description of different approaches for clustering
 - Description of some popular clustering algorithms.
- Chapter 3
 - Presentation our early research.
 - Discussion and answers to hypotheses.
- Chapter 4
 - Overview of the testing application.
 - Discussion of important attributes.
- Chapter 5
 - Description of online partitional algorithms with local search.
 - Results of experiments on partitional algorithms.
 - Conclusion to the experiments on partitional algorithms.
- Chapter 6
 - Description of online hierarchical algorithms with local search.
 - Results of experiments on partitional algorithms.
 - Conclusion to the experiments on partitional algorithms.
- Chapter 7
 - Conclusion to thesis.

2 Review of literature

The review of literature chapter covers a small discussion of how clustering can be used in machine learning. We will also give information about clustering, including fields of use and important properties of a good clustering algorithm.

Finally, we explain different approaches for different clustering algorithms and explain their strength and weaknesses.

2.1 Machine learning

The field of machine learning is based on trying to create algorithms that can “learn” what actions to perform. Machine learning is split into several groups; amongst them is supervised and unsupervised machine learning.

In supervised machine learning the machine already have some notion of what its output should be. Its task will then be to chose actions which results in those outputs. In unsupervised machine learning on the other hand, the algorithm have no notion of what the output should be nor does it get any response to its actions from an external environment. When these machines gather information, the results should be organized into optimized categories. Clustering is an important approach of archiving this [3] [4].

2.2 Dendrogram

Because of the algorithm we used as a starting point for our work we did some early experimenting with a tree structure where nodes could have n children (where $n > 0$) but decided to change it so the work focused on the dendrogram structure instead. Dendrogram is a tree where all nodes, except for the top and leaf nodes, have exactly two children and one parent. The main reason for this decision is that sub clusters are easier to locate using a dendrogram.

Since the decision of using dendrogram structure came a little while into the project we decided to make changes to the layer concept as well. Layers are actually not used at all in our algorithms when clustering a dendrogram structure (but are still a part of the non-dendrogram algorithms). The change from non-dendrogram to dendrogram algorithms also opened for the possibility of performing changes to the tree structure. The only purpose of dendrogram-layers in our work is to output the tree structure in a clean way to the monitor. It is important to note that in the dendrogram figures throughout the report, a layer does not contain all objects in the input set as it does in the non-dendrogram trees.

The figure bellow shows an example of a dendrogram.

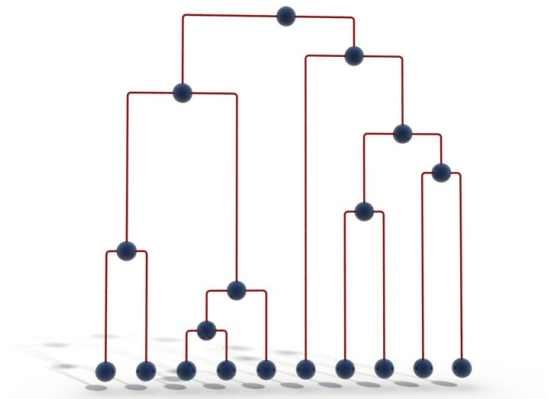


Figure 5 Example of a dendrogram

2.3 Mean and SSR

Each cluster will have a mean that lies in the middle of the cluster (sometimes called the center point). This mean is calculated by adding all x values of the points in the cluster together, then divide that number on the amount of points in the cluster. The same is done for the y values (and any other values if it is working in more than two dimensions). This result in the x and y (referred to as X in the formula) coordinates of the mean. The formula for this is:

$$M_i = \frac{1}{N_i} \sum_{x \in D_i} x$$

Figure 6 The calculation of mean

For each cluster a value called “the sum of square roots” (from now on just called SSR) is then calculated. This is the sum of the Euclidian distances from each point in a cluster to the clusters mean. The SSR is calculated with the following formula:

$$J_i = \sum_{x \in D_i} ||x - M_i||^2$$

Figure 7 The calculation of clusters SSR

The total SSR for all the clusters is then calculated, simply by adding each of the clusters SSR. The goal of the clustering is to minimize this value as much as possible. This value is calculated by the formula:

$$J_e = \sum_{i=1}^c J_i$$

Figure 8 The calculation of total SSR

Since this calculation is CPU demanding, some optimized formulas for updating the mean and SSR values can be used. The formulas are used when a point is moved from one cluster to another. Using this saves the algorithm from having to iterate through every point in the clusters affected. This formula has to be used to update the values each time one point is moved. If two points are going to be swapped between two clusters, one point has to be moved and the values updated, then the second point can be moved and the values updated again. Also, the way this formula works it is important that the SSR value gets updated before the new mean is calculated.

When a cluster loses a point, the algorithm calculates a new mean and SSR for the cluster with the formulas:

$$J_i^* = J_i - \frac{N_i}{N_i - 1} \| \hat{X} - M_i \|^2$$

Figure 9 Update SSR when adding a point

$$M_i^* = M_i - \frac{\hat{X} - M_i}{N_i - 1}$$

Figure 10 Update mean when getting a point

When a cluster gets a new point the algorithm calculates a new mean and SSR for the cluster with the formulas:

$$M_j^* = M_j + \frac{\hat{X} - M_j}{N_j + 1}$$

Figure 11 Update mean when losing a point

$$J_j^* = J_j + \frac{N_j}{N_j + 1} \| \hat{X} - M_j \|^2$$

Figure 12 Update SSR when losing a point

An explanation of the letters used in the formula is described here:

- M_i = cluster mean for cluster i
- N_i = number of points in cluster i
- X = points X value (also used for the Y value, and every other dimension the point has)
- J_i = SSR value for cluster i
- J_e = total SSR values for all the clusters
- X^{\wedge} = X mean

Figure 13 Explanations of the letters in the formulas

2.4 Clustering

Data clustering is a common technique for statistical data analysis, which is used in many fields, including unsupervised machine learning, data mining, pattern recognition, image analysis and bioinformatics. Wikipedia, the free online encyclopedia defines clustering as follows: “*Clustering is the classification of similar objects into different groups, or more precisely, the partitioning of a data set into subsets (clusters), so that the data in each subset (ideally) share some common trait - often proximity according to some defined distance measure.*”[7].

Further Wikipedia defines classification as: “*the act of placing an object or concept into a set or sets of categories (...), based on the properties of the object or concept*” [8].

A simpler way of saying this is that one should put objects which share traits or similarities into the same cluster, while objects with different traits should be put into other clusters. One example of similarity measurement could be Euclidian distance between objects.

For a clustering algorithm the following properties are desirable:

- **Scalability**
If an algorithm is not scalable, the time required to solve a problem will increase with more than a linear factor. This will make the algorithm less useful as the complexity of the problem increases.
- **Dealing with different types of attributes**
The algorithm has to work whether it is two-dimensional points or other types of attributes such as the classification of plants, roadmaps etc.
- **Discovering clusters with arbitrary shape**
No matter what shape the clusters have, the algorithm should be able to recognize the optimal solution. What that optimal solution is had to be defined based on the purpose of the clustering. For example, if an algorithm always produces clusters with rectangular shapes, the algorithm is probably not satisfying this property.
- **Ability to deal with noise and outliers**
In some cases the input data may have elements breaking from the regular pattern. This may contribute to trapping some algorithms in what is called a local optimum, fooling them to believe they have found the best solution, while they have not.
- **Insensitivity to order of input records**
The order of how the data sets are arranged when starting the algorithm should not affect the final result of the clustering.
- **High dimensionality**
If an algorithm works with n -dimensions, it should also work for $(n+1)$ dimensions under the assumption that $n > 0$.

- **Interpretability and usability**

The algorithm has to be flexible, making it easy to adopt the algorithm to different areas of use.

These properties are not to be considered as binary properties either supported or not supported. They can be supported to a lesser or greater degree [10].

2.5 Partitional clustering

In partitional clustering objects gets divided into a fixed amount of clusters. Before the clustering starts, the number of clusters can either be specified by the user, or it will be found dynamically by the algorithm during runtime.

This may be a good method when it is easy to predetermine how many clusters are optimal, but can be a somewhat less useful in cases where the numbers of clusters are harder to predict. The main advantage of partitional clustering is speed.

2.6 Hierarchical clustering

Hierarchical clustering will instead of focusing on a single solution focus on finding solutions on several hierarchical levels. For instance: If there are two clusters which themselves contains three sub clusters each. A partitional solution could be to either try to find six clusters or two clusters. A hierarchical solution would on the other hand generate a structure where both the solution for six and for two clusters is easily available. If the structure is traversed even combinations of sub clusters and main clusters are available. Also internally in a cluster, with no clear structure the objects would be organized based on proximity.

Basically there are two types of hierarchical clustering, divisive and agglomerative clustering. The difference is that divisive clustering will start with all objects in a single cluster which it will split into smaller clusters. These clusters will also split, and so on, until there is only one point in each cluster. The other approach is called agglomerative clustering. Here objects start out alone, in each of their very own cluster. The algorithm will then select clusters, usually the closest or within a limited threshold, and merge them into another cluster. This will go on until there are no more valid merges available. Agglomerative clustering is used much more often than divisive. This is because it is harder to define how to split a cluster than how to merge them.

The “closest agglomerative” algorithm we will be using as a standard will always choose the two closest clusters to merge. This will always provide the same result when run on the same problem, and will provide a very high quality solution, which in many cases also will be the best possible solution. The only way to ensure a best possible solution is to use a brute force algorithm which will check all possible hierarchical solutions against each other. The closest agglomerative method does not scale well. Even one extra object means that that object will have to be tested against all the other clusters during all the steps of the algorithm.

The figure bellow shows how a hierarchical tree can be created. The figure shows the structure of the hierarchical tree. The top node is the solution where all objects are stored in a single cluster. This node then has two children which contains these objects distributed between them etc.

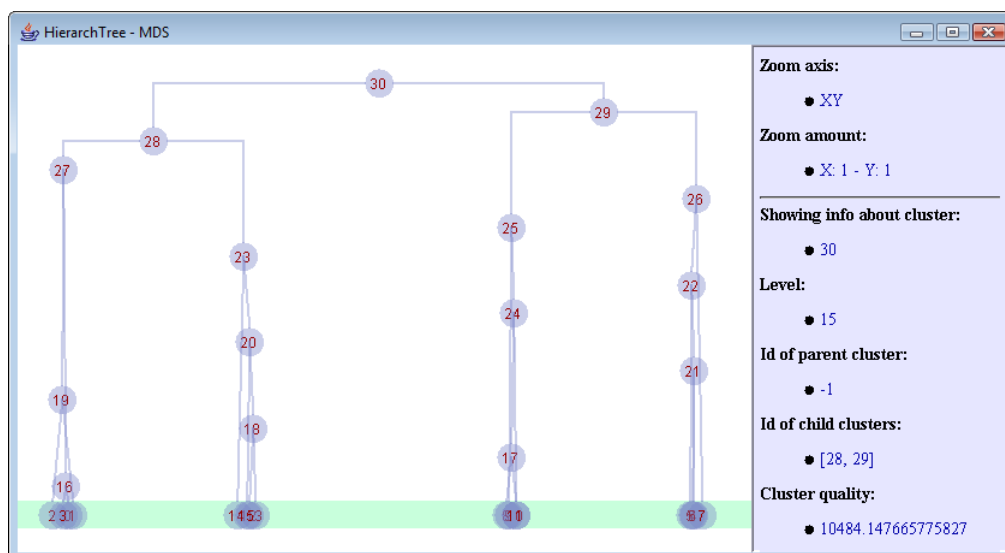


Figure 14 Hierarchical clustering

2.7 Offline cluster algorithms

An offline clustering algorithm will perform clustering steps that involves all or nearly all of the objects. Each step generally performs large changes, and therefore each step also requires more work than a step in an online clustering algorithm.

2.8 Online cluster algorithms

Online clustering performs only small changes to the cluster structure each step. For instance the “two point” algorithm described in chapter 5.2 performs changes by swapping two points between clusters, effectively limiting changes to two clusters instead of affecting the entire structure.

The idea behind these small steps on few objects at a time is that there should at any time be a solution available, even while the algorithm is running. Of course at the early stage of clustering this solution would be of low quality, but it should be improved in many small steps as the algorithm continues to work.

When having a large set of data, this drastically reduces the required computational time for each step, but the number of steps needed to get an adequate result may be somewhat higher than an offline algorithm. Both online and offline algorithms can be stopped any time, and both will have to either complete the step they are currently performing or to do a rollback to the result of the previous step. An online algorithm would in this case either perform a very small step and finish or rollback a minor change then stop. An offline algorithm has the choice of completing a potentially time consuming step or performing a rollback which might cause it to miss or undo major changes.

Even if online algorithms will require more steps to finish, the change in workload for each step will probably make online algorithms scale better for larger problems

2.9 Fuzzy clustering

In fuzzy clustering objects can belong to more than one cluster at the same time. If an object lies close to the border of more than one cluster it can be said to be partly present in all of them, for example by defining one point to belong 45 percent in cluster A and 55 percent in cluster B. None of the algorithms described in this report supports fuzzy clusters.

2.10 Agglomerative cluster

Agglomerative clustering, also known as bottom-up clustering, will initially have all points distributed amongst a number of clusters equal to the number of points. It will usually find two clusters that are close to each other and merge this in to a new cluster. This process will be continued until all clusters are merged to into one large single cluster. This large cluster will be the root and each of the clusters with only one point in them will be the leaf nodes. A tree structure where all nodes, except the leaf nodes, contain two children and all nodes except the top node contains one parent is often referred to as a dendrogram. [15] [16].

The figure below shows 10 points clustered with agglomerative clustering. The clusters are merged together at different levels. Note that in this picture the distance between the nodes are not related to the distance between the actual objects or clusters.

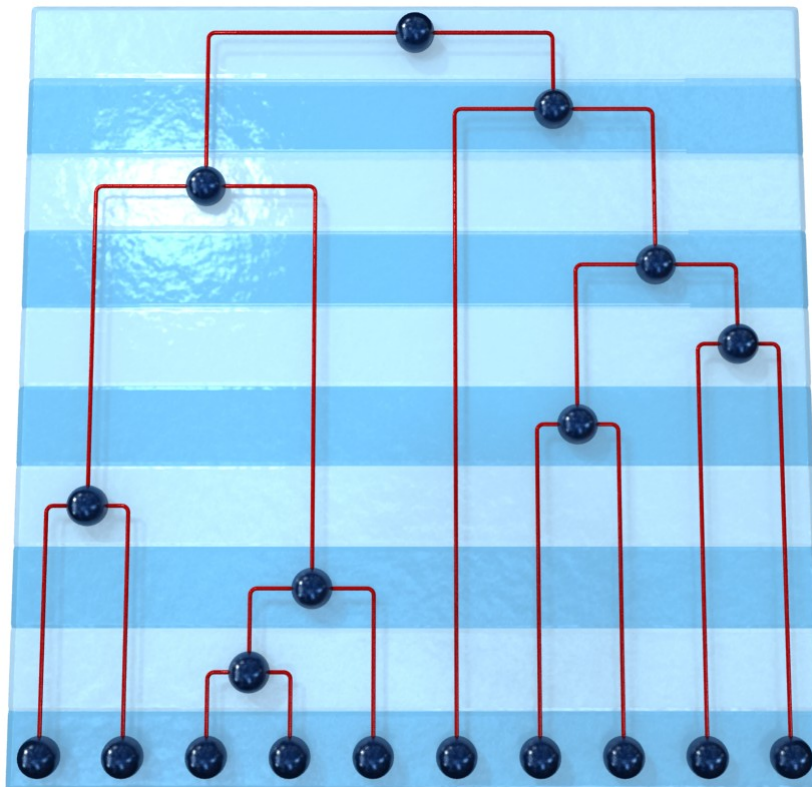


Figure 15 Agglomerative clustering

2.11 Different Clustering algorithms

As stated earlier there are already many different techniques for clustering available. Most of the algorithms used today have both strengths and weaknesses. We will try to explain the most common algorithms and a version of the algorithms we have worked with earlier. All of these are partitional clustering methods, but served as a introductory study of clustering before starting the work on the hierarchical clustering. The intention of this is to try to compare the currently available algorithms against each others as well as against the attributes described earlier in chapter 2.4. Most of the popular algorithms come in different variants.

2.11.1 Popular Clustering algorithms

K-Means

K-Means is an offline partitional algorithm. It starts out with points distributed into a given number of clusters; this number has to be set before the algorithm can start. Distribution could either be randomly or non-random. In some cases the centers are placed at random instead of objects being divided between clusters. In the case where points are distributed to clusters, centers for each cluster will be calculated. In the next phase all objects will be associated with the center closest to itself (in other words regrouped into the cluster with center closest to its own location) before the centers is recalculated. This is done until no object switches cluster or alternatively until no center point moves.

Compared to the online algorithm we have worked with and discussed later in this report, k-means works very fast. By reassigning many points between each recalculation of centers it will in almost all cases require less work than the algorithm which moves only one or two points at a time.

Because of its performance k-means is a very popular algorithm when it can be used. It supports high dimensionality and it is insensitive to order of input. Although it is very quick it does have some limitations. Since it is an offline algorithm it is not very well suited in for example distributed systems. It handles noise and outliers badly and it is stated on Wikipedia: *“Recently, however, David Arthur and Sergei Vassilvitskii showed that there exist certain point sets on which k-means takes super polynomial time: $2^{\Omega(\sqrt{n})}$ to converge.”* [13]. K-means will generate different clusters dependent on the initial clusters.

Quality Threshold

The quality threshold algorithm is more dynamic than the previously discussed algorithm. It is a partitional offline cluster algorithm, but does not need the numbers of clusters to be predefined. Instead a maximum diameter for a cluster has to be defined. The algorithm will, for each object available, generate a cluster containing every other point within the maximum diameter specified. The cluster with the highest count of objects is selected as a “finished cluster” and all its objects will be ignored at the next steps of the algorithm. The algorithm performs the same step all over again on the reduced set of objects until there are no objects left to work with. [14].

The quality threshold is an offline partitional clustering algorithm. It is somewhat more computer intensive than for example the k-means algorithm, but it will always deliver the same clusters as result no matter how many times it is run on the same problem. This can in some cases be an advantage. It handles noise and outliers by putting them into separate clusters instead of forcing them into a poorly suited cluster. It also supports high dimensionality.

Closest agglomerative

Closest agglomerative is what we have called the agglomerative algorithm for systematically creating a clustered tree. This is done by finding the two closest clusters without any parents, and these will be children of a new cluster. When there is created a cluster containing all the points, the algorithm has completed.

When we first started using this method, we assumed it created the perfect tree, with the lowest possible SSR value. But later as a result of our research we learned that this is not always the case, at least not in terms of getting the lowest SSR value. In Figure 16 we have created an example of this. The tree at the left shows the tree created by the closest agglomerative algorithm, while the tree at the right shows the solution with the lowest possible SSR value. But even though the closest agglomerative has a higher SSR value, this solution might still be a better solution. In many cases it may be more important to group together the closest objects, as the closest agglomerative does. This is the main reason of why we can use this for a comparison to our own algorithm. The alternative would be to use a brute force algorithm, which often would be unfeasible because of the tremendous computing power it would need in experiments of a certain magnitude. And since we believe that the structure with the lowest SSR value may not always be the best solution, we found the closest agglomerative to be the most appropriate choice.

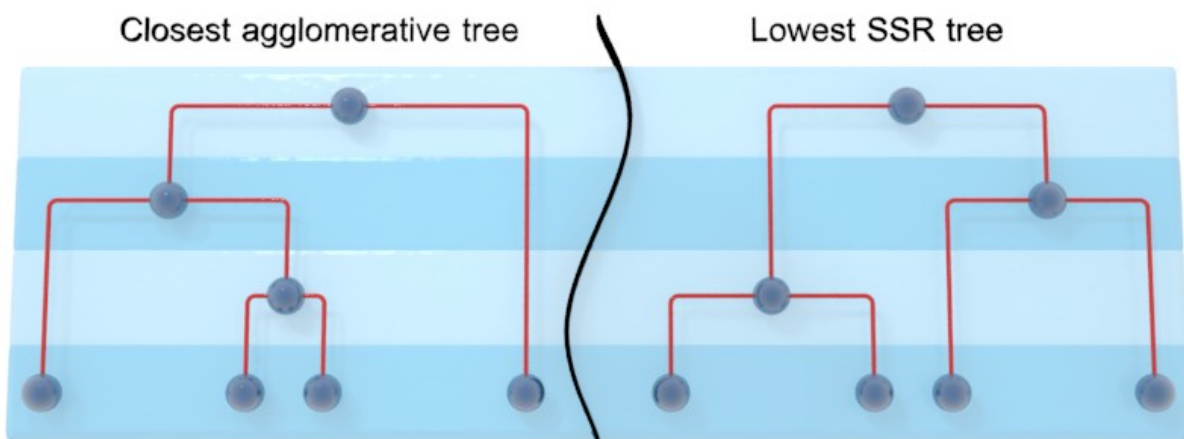


Figure 16 Closest agglomerative and lowest SSR tree comparison

3 Research

The research chapter covers the results from the research study we did before this thesis started. We did a classification of our problem, discussed sub problem approaches and gave an overview of our programming platform, the design of our framework and information about our organisation of this mater thesis.

3.1 Problem classification

Research design is divided into three different categories; exploratory, descriptive and explanatory.

Exploratory design is often in the scope of gathering relevant literature and case studies. The first four of our sub problems are of this category. For these problems we will need to gather all relevant literature in order to pinpoint our problems and make a plan for avoiding any of these problems.

Explanatory design is the classic experiments part and are a part of our last sub problems. This sub problem is typical programming tasks which needs experiments.

3.2 Hypothesis approach

Approach to test hypothesis 1

To test whether the systematic algorithms scale well compared to the totally random ones we will have to run experiments on both types and examine the results to see how well they scale.

Approach to test hypothesis 2

Many algorithms have to update the entire structure when performing some kind of action. If we can create an approach witch only needs to update to a little part of the tree we hope to drastically reduce the workload and hopefully increase the speed of the algorithms.

Approach to test hypothesis 3

The closest agglomerative approach requires a full update of the entire structure every time a new object is introduced to the problem. Therefore this approach will have a scaling problem as the workload is increasing. With an online hierarchical algorithm we hope to reduce the workload and hopefully this result in an algorithm which is able to scale lot better than the closest agglomerative approach.

3.3 Application functionality

Our application has different levels of functionality. At a basic level the application should be able to randomly generate problems. The algorithms should be able to use this point and run all of the algorithms on. At this level every algorithm should have been tested for all bugs we are able to think of. A big change in this basic design can be hard to do on a later stage.

When we have implemented the basic functionality, we can start to work on the more advanced functionality. In this section we are going to build a framework which gives the users a GUI to test and evaluate the problems that is solved with the basic application. This advanced functionality should be able to give a visual representing of the initial and end results, a way to manipulate the number of points and the number of clusters.

3.4 Programming platform

In order to implement our algorithm and the framework we need, we have chosen the Java development platform and Eclipse SDK as developer tool.

Java

Java is an object-oriented programming language developed by James Gosling and his team at Sun Microsystems in the early 1990s. Java does not compile to native code; it's compiled to a byte code which is then run by a Java virtual machine. Because of this virtual machine, a Java program is platform independent. Therefore Java is used at many platforms like computers, mobile phones, smart card etc [1].

Eclipse SDK

Eclipse is an open source software development kit, or simply SDK, which gives a good platform for software development. Eclipse platform can be downloaded for free and have a lot of plug-in that supports different technology [2].

3.5 Programming

The programming platform for this application is done in Java 2 SE version 1.5. To make it easier to maintain the code, every part of the source code must be stored in different packages. Every class that belongs to the GUI must be stored in a GUI package; different algorithms must be stored in different packages etc.

To avoid problems that can arise when several people are working on different part of the application, every packages need to have an interface which the programmers can work against. All source code must be stored at a CVS. This way it is easy for everyone to have a complete overview of the application.

The style of the source code syntax must follow the standard code conventions for the Java programming language [9].

3.6 Design

3.6.1 Framework

From Wikipedia, the free encyclopedia design is defined as this: *“design is a visual look and/or a shape given to a certain object, in order to make it more attractive, make it more comfortable or to improve another characteristic. Designers use tools from geometry and art. Design is divided to some sub-categories: Graphic design, Buildings and nature design, Consumer goods design”* [11].

Our framework needs a graphic design and we need to design this as intuitive as possible. This means that there should not be use for any manual in order to use this.

We need to design a setting panel which gives the user a series of setting which can help him with the solving parameters.

The framework will consist of the following parts:

- **Main interface**
Our main interface will list the points we are going to work with and which cluster the points belongs to. This is just to give the possibility to show the points with their values and which clusters they belong to. This can be useful information when inspecting the results. This is also where we find the menu for all the functions of the program.
- **Settings panel.**
The settings panel will provide the main settings for the solution. With all these settings, we can easily change the parameters for the algorithm and also change the settings for the visual settings. E.g. it is important to switch off all GUI updates when solving a problem if speed is important. And it can be a good thing to turn on all GUI settings when we want to look at the visual results.

- **Visual representation of the clusters**

Because it is easier for a human being to evaluate the result of the clustering by visually inspection we have created an application for exactly this. The clustering application have been implemented with the possibility to solve problems in two dimensions The viewing application can be run during the clustering process, which gives the users a visual simulation of the actual moves done during the whole solving process. Since the drawing process is CPU demanding some settings are available for the user to set. The settings that can be changed are update rate; toggle on and off drawing of the lines between points and toggle on and off drawing the middle points.

- **GUI for creating custom clusters**

Sometimes it can be useful to manually create our own clusters. To test out special case problems this can be useful way to pinpoint these problems. We can e.g. simulate “real world” problems by generating noise. This can be useful when trying to produce the case of a solution lock caused by a local optimum or to see if the algorithms can find global optima when the clusters have arbitrary shape.

3.6.2 Validation

In order to validate our work, we will rely on several methods. One important method is to have the ability to visually inspect the result of the clustering. This will give us an intuitive way of verifying that the behavior of our application is appearing to be reasonable. When running the application for a longer period of time on different problems, any errors in the application are likely to produce strange results and in this way a lot of problems may be detected.

When running the application on problems that have clearly separated clusters of elements, it is especially easy to check if the algorithm can recognize these clusters. Since one type of problems we are working with is two dimensional points, one way of achieving this visualization is to draw all the elements, colored by the clusters they currently belong to. But we will also implement the functionality to view the progress of the clustering by time. Then we will display a graph, where the total distances between every element and their corresponding cluster will be plotted as a function of time.

An example of this can be seen in Figure 23, which was made on a previous course we had. This can also be used to compare multiple executions of the algorithm on the problems and use this to discover critical sections. Smooth curves, decreasing fast in the first phases and flattening out as time goes by is expected, while errors may cause these curves to be more irregular.

A more accurate way of validating the algorithm will be to execute it step by step and manually checking that the output is correct. Since the algorithm is heavily based on math, it is easy to check the correctness of these calculation regarding accuracy, processing speed, and ability to find global optima.

3.7 Organization for our master thesis

3.7.1 Work process

Research phase

In the first phase of our master thesis, we had to get a good understanding of how the algorithm is supposed to work. We look into existing clustering algorithms in order to get more familiar with these types of problems and how they can be approached. We also did some research to find out more about the fields where these algorithms can be used. This is important to get a better understanding of the problem and what the essential criteria's are. While working with the algorithm, having enough knowledge of the field may be used to make improvements and optimizations. As the algorithm we are working with has not been tested before, it is important to always look for areas of improvement.

Development and analysis phase

At this phase we needed to have a complete understanding of how the algorithm should work and a good insight to the problem area. During this period, the requirements was refined and updated as our knowledge increases. We were focusing on having a good and well defined requirement specification, which will help us with the development of the application.

We relied on a kind of agile software development for the implementation. Wikipedia states that: *"Agile methods emphasize real-time communication, preferably face-to-face, over written documents. Most agile teams are located in a bullpen and include all the people necessary to finish software"* [18]. This method suits us well, as we usually work together and have group discussions about many parts of the project. A prototype of the application will be developed, with the most important properties built in. As new requirements and features are implemented, these should be tested thoroughly.

When the application reached a certain point in the development, we used this to analyze and experiment with the algorithm. This helped to see what improvements can be made and evaluate the results of these. The evaluation and analyzing of the algorithm are one of the most important elements of this project, and therefore given a great part of the time schedule. It was also very import to document the progress regularly.

These two phases was completed at two weeks before the final deadline. When working with projects there is always a chance of having some unforeseen issues, in that case it is important to have the time to deal with it

When we had completed these two parts, we started to prepare the presentation. It consists of an oral presentation, a video presentation and a poster. This was completed about one week before the deadline, in order to have some time to do any potential additional changes.

3.7.2 Time schedule

The group consists of three people, and we need to organize the work in such a way that everybody always have something to work with. From other projects we have experienced that it has several advantages to have more than one person working with the implementation. This makes it easy to come up with new ideas and asserts that the source code will be readable and as general as possible. During the implementation phase, two of us were scheduled to participate with the programming. Whenever there was a need to do some programming in the other phases, there will probably be sufficient with one person taking care of this.

The person(s) that is not working with the programming part was responsible for working on the report. We had a focus on writing a good report from week one. This way the report was always be up to date and did not experience any “panic work” the last week of this master thesis.

The group had a close collaboration with the problem owner with status meeting every week and e-mail or phone when an urgent problem occurs.

A big part of this project was to research and get familiar with the field. But most importantly we had to reserve a great amount of time to use for experiments and evaluation of the algorithm, as the application gets to a stage where it can be used to this. The time schedule we have created had some small changes during the project, as some parts required more or less time, but some of the dates are final delivery dates and cannot be changed. This schedule served as a guideline and we tried to keep this track unless we had a good reason to change something.

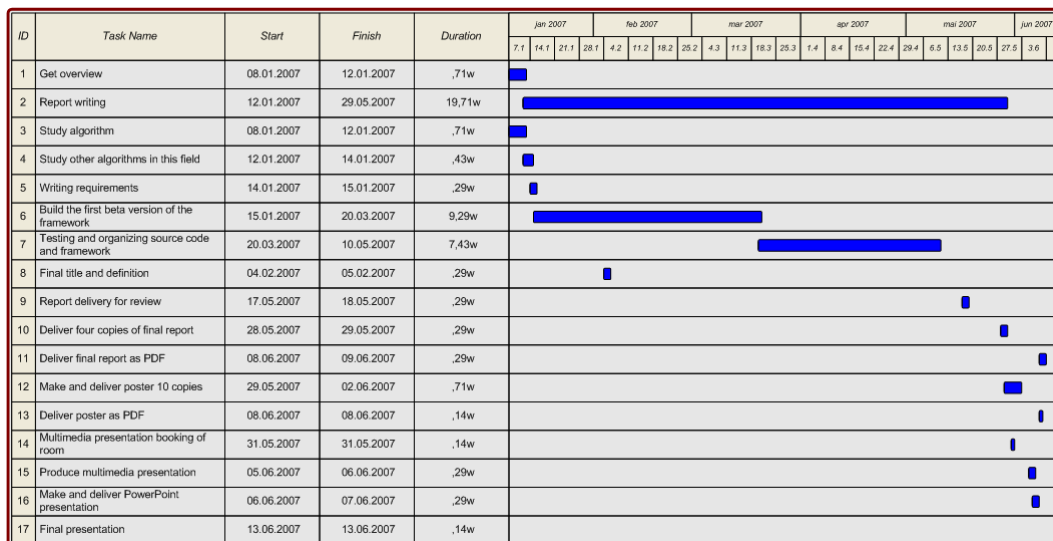


Figure 17 Shows our time schedule

3.7.3 Critical resources

Importance of research

When we started with the implementation, we needed to have the full and absolute correct understanding of how the algorithm worked. This made the research phase critical, all the requirements had to be accurate and reflect the desired behavior of the application. Any errors here will probably give an incorrect result and can be hard to discover or trace back to the source of the problem. Valuable time may be wasted if we search our code for programming errors, while the error is located in a requirement. Because of this, we had to take the time to ensure the quality of our research and preparations.

Evaluation of the algorithm

The most important phase of this project is the evaluation and analysis of the algorithm; hence the tests done with our application is critical. It is very important that these tests are accurate and not influenced by any errors. This makes it necessary to thoroughly validate the results of our experiments and make sure that the behavior of the application is flawless.

4 Overview of our application

This chapter will give an overview of our application and we will explain the purpose of the different parts of application.

4.1 Graphical User Interface

Our application is divided into six different parts which we discuss in the next six sub chapters.

- Main interface
- Settings panel
- Representation of the clusters
- GUI for creating custom clusters
- Show the hierarchical tree structure
- Visual representing of the clustering results

4.1.1 Main interface

Our main interface is the start point for the application. Here the user can choose different settings from the menu bar. This main interface will also give important information about the clustering. The screenshot below shows an example of this.

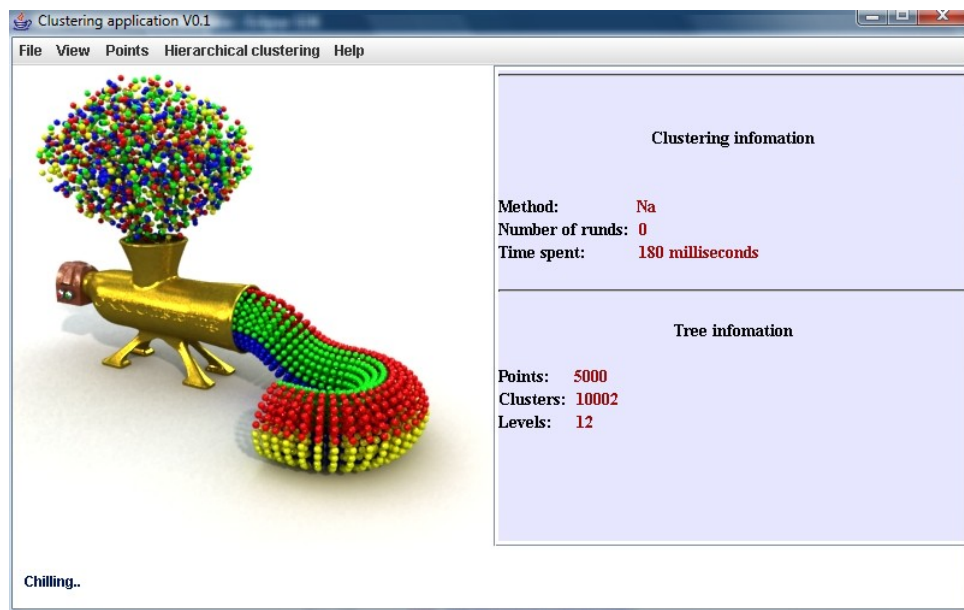


Figure 18 Shows the main interface for our application

4.1.2 Settings panel

The settings panel will provide the main settings for the framework.

With all these settings, we can easily change the parameters for the algorithm and also change the settings for the visual settings.

E.g. it is important to switch off all GUI updates when solving a problem if speed is important. And it can be a good thing to turn on all GUI settings when we want to look at the visual results.

In this panel the user can change the following parameters:

Solving parameters:

- **Number of points**
This is the number of points that are going to be created. This is to easily change the complexity of the problems the algorithms should solve.
- **Number groups**
This is the number of groups/clusters when creating Gaussian problems. The algorithm should be able to recognize the clusters, regardless of how many there might be. Therefore it is useful to test the algorithms on clusters with different sizes.
- **The max integer value which the points can have**
In our application, the points can have an integer value between zero and a maximum value for each dimension. If this value is low, the points will have grid-like positions, with a high possibility of overlapping each other. Increasing this number will give the points more freedom of where they can be placed. Changing this value should however not have any influence of the quality of the clustering, and clustering overlapping points was also a special case we found worth examining. Especially during testing we found it useful to have the ability to change this value in runtime.
- **Point dimension**
This is the number of dimensions to be worked with. The application supports 2 or more dimensions although the main focus has been on solving clusters with two dimensions. If solving a three-dimensional problem, one has the opportunity to view the problem in a 3D window. When rotating the camera around the points, it is pretty easy to assess the quality of the result. When having more than three dimensions, there will be little use in visualizing, as one or more dimensions will not be seen. However, the GUI with hierarchical tree will try to scale the problem down to one dimension and this often helps to get an impression of the clustering quality.
- **Number of runs when clustering the hierarchical tree**
The different algorithms will be iterated through a specified number of times. In our previous version of the application, the algorithm was stopped after clustering a certain amount of iterations without any improvement in the clustering. But since the total quality of the problem can both decrease and increase during clustering, knowing when to stop is more complicated in this variant. Also we have strived to keep the algorithm as local as possible, just having a specified number of iterations means less need to have a global perspective of the problem.

- **Number of runs before taking a new measure**
How often a measure of the current total distance with a timestamp should be stored. These measurements are used when displaying the progress of the clustering. When solving complex problems, this value can be increased to avoid taking too many measurements. Currently, when viewing the progress 30 measurements will be used. Having many thousands measurements is a waste of resources. As it is hard to say in advance how many loops the algorithm will do, it is difficult to decide when to take a new measure. But this may be changed and made more dynamic in our future work.
- **Number of runs between each time the quality is correctly calculated (grayed out)**
The algorithm can run a specified number of iterations using approximate calculations for the quality, and then calculate the correct quality. This was implemented to speed up the quality calculations when using SSR since that method uses a lot of time to include all points in a cluster to find the quality. Since SSR has mainly become a method of adjusting the quality when using the much faster mean-methods, this option is grayed out. The results of using this option with SSR are also highly questionable.
- **Number of runs before checking if current solution has best quality (grayed out)**
It is possible to have the algorithm every n-round checking if the current solution is the best solution so far and storing this. Then the best found solution will be returned at the end of the clustering, rather than just the current solution. This was included because the quality of the clustering may increase during clustering and the solution with the best quality may not be the one at the end. But in order to achieve this, the global perspective of the problem is needed. Using mean as quality method to find the total quality is not preferable, as it does not represent the “correct” quality of the problem. Then the quality has to be calculated for every cluster using SSR, which requires a lot of time. Because of that, this option is grayed out.
- **Max number in the k-means algorithm when creating the tree (grayed out)**
This is the maximum number of iterations allowed by the k-means algorithm when creating the tree. When creating a hierarchical tree, using k-means for each level is one option. This often results in well-defined clusters without using too much time. To limit the time used even further, restrictions to the number of iterations used by the k-means algorithm can be specified. But since the main focus in our project is clustering dendrogram, this option is not that important and is grayed out.
- **Method to use when calculating the quality of the clusters**
Here the quality method of the clustering can be specified. This is used when clustering a single run (clustering one time with selected algorithm on current problem). When doing multiple runs, the quality method is selected for each algorithm and this option is not used.

- **Number of runs when calculating the average distance (doing multiple runs)**
This is the number of times one algorithm is going to solve the same problem. This is used when running the algorithms multiple times on the same problem to get an average result of the current algorithm. The results of the different algorithms can then be compared. The more runs an algorithm has on a problem, the more accurate the average graph will be. It is important to generate this type of average results since the work is done by randomly choosing and moving clusters and points. This means that both time to complete and the quality can, in rare cases, be either much better or much worse than usual, thus giving a faulty result. Calculating average values limits these types of errors.
- **Max number of clustering threads running at once**
When clustering multiple times with several algorithms at once, the max number of threads that is running at the same time can be specified. We used our application on different machines, with different hardware. We wanted to have the ability to run several algorithms in parallel, taking advantage of machines with multiple cores / CPUs.

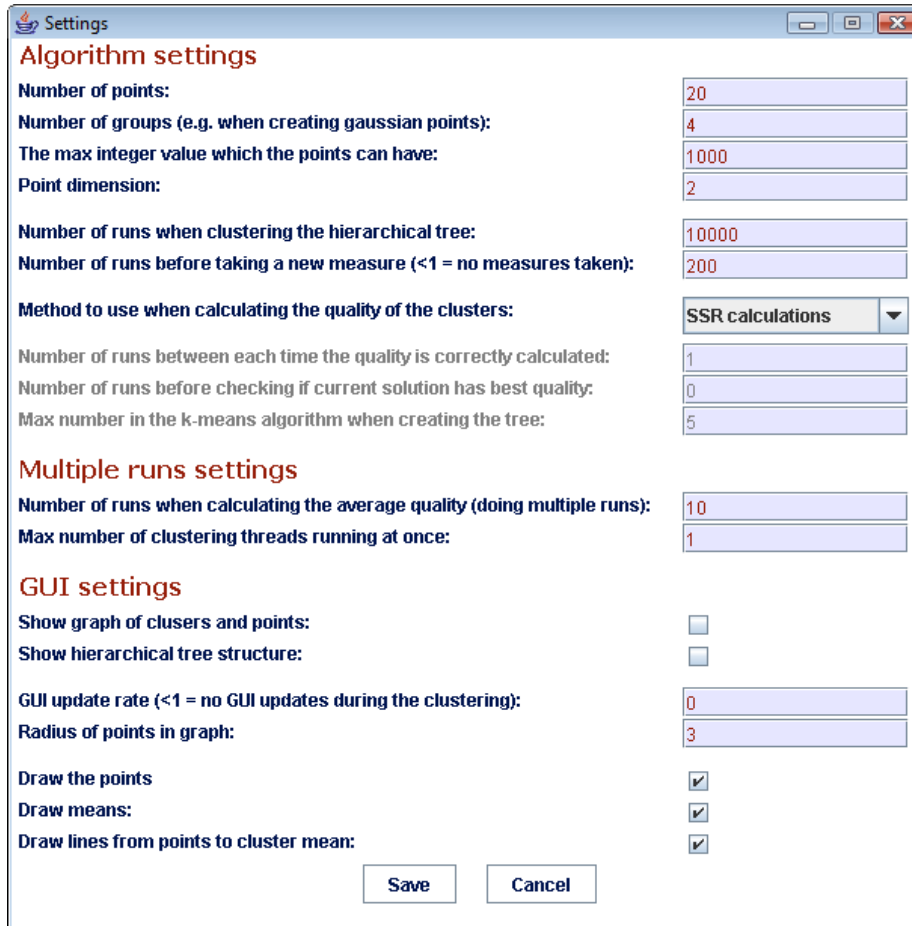
GUI parameters:

- **Show graph of clusters and points**
Turn on or off the visual representation of the points. It is useful to have this enabled when testing algorithms and tuning the settings. But when measuring the efficiency of the algorithms it is not needed to waste time by doing GUI updates.
- **Show hierarchical tree structure**
Turn on or off the visual representation of the hierarchical tree. The hierarchical tree will show the tree structure of the problem. The points on level 0 in this GUI will be the points in the problem, arranged with a variant of multidimensional scaling. Having this enabled can be useful when verifying the quality of the clustering. Generally, less crossing of lines means better clustering. It is also possible to zoom (both on X, Y and XY axis) in order to get a better view on specific parts of the tree.
- **GUI update rate**
The number of runs the algorithm has between each GUI update. A lower value will give a smoother animation of the clustering, on account of the clustering speed. If the value is below one, the GUI components will not be updated during the clustering.
- **Radius of points in graph**
This is the size, given in pixels, of the points in the 2d visualization window.
- **Draw the points**
Turn on or off whether the points themselves should be shown in the point visualization window. Turning this off can give more focus to the clusters and there means.
- **Draw means**
Turn on or off whether a cross should be drawn at the mean of the clusters. By turning this on the application will show where the mean of the clusters are placed.

- **Draw lines from points to cluster mean**

Turn on or off whether a line should be drawn from the points to the mean of the cluster they are related to. Turning this on gives a better impression of which clusters the points are related to, but may make it more difficult to see the points.

The picture below shows a screenshot of the application with some basic settings.



Section	Setting	Value
Algorithm settings	Number of points:	20
	Number of groups (e.g. when creating gaussian points):	4
	The max integer value which the points can have:	1000
	Point dimension:	2
	Number of runs when clustering the hierarchical tree:	10000
	Number of runs before taking a new measure (<1 = no measures taken):	200
	Method to use when calculating the quality of the clusters:	SSR calculations
	Number of runs between each time the quality is correctly calculated:	1
	Number of runs before checking if current solution has best quality:	0
	Max number in the k-means algorithm when creating the tree:	5
Multiple runs settings	Number of runs when calculating the average quality (doing multiple runs):	10
	Max number of clustering threads running at once:	1
GUI settings	Show graph of clusters and points:	<input type="checkbox"/>
	Show hierarchical tree structure:	<input type="checkbox"/>
	GUI update rate (<1 = no GUI updates during the clustering):	0
	Radius of points in graph:	3
	Draw the points	<input checked="" type="checkbox"/>
	Draw means:	<input checked="" type="checkbox"/>
	Draw lines from points to cluster mean:	<input checked="" type="checkbox"/>

Figure 19 Shows the settings panel

4.1.3 Representing of the clusters

Because it is easier for a human being to evaluate the result of the clustering by visually inspection we have created an application for exactly this. The viewing application can be run during the clustering process, which gives the users a visual simulation of the actual moves done during the whole solving process. Since the drawing process is CPU demanding some settings are available for the user to set. The settings that can be changed are update rate; toggle on and off drawing of the lines between points and toggle on and off drawing the middle points.

The figure below shows how 400 points can be clustered into 9 different cluster levels. This figure shows the results from the highest level.

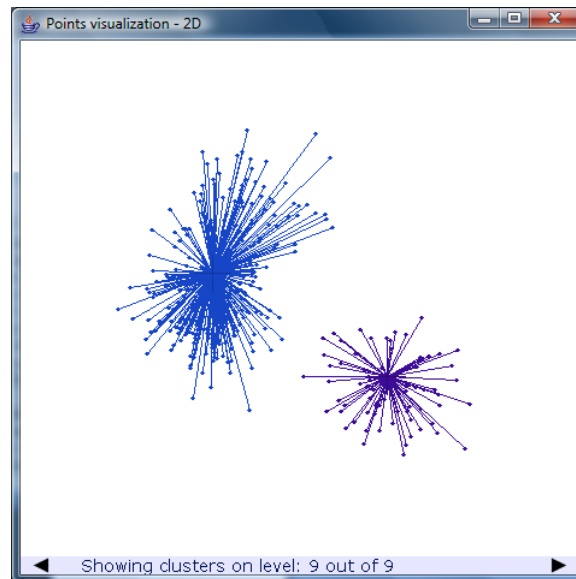


Figure 20 Shows 400 points clustered into 9 different cluster levels

4.1.4 Tree structure for hierarchical clustering

Sometimes it can be hard to get an overview of the entire problem. Therefore we created an application which shows the entire tree structure for the hierarchical clustering. This structure is build with multidimensional scaling technique, or simply MDS. This means that all points in the two dimensional space are mapped to a one dimensional line. This is not a hundred percent perfect mapping but it gives a good understanding of the problem.

This application gives the user the possibility to click on different clusters to get info about the clusters SSR, mean value, parent cluster, child cluster and number of points that belongs to the cluster. It also highlights the entire path to all the children. The tree can also be updated during the entire solving process.

When the number of points increase it can be hard to see all the connections between each cluster. Therefore we created a zoom function. This zoom function gives the user the possibility to zoom in at points of interest. It is possible to zoom into the X, Y and XY axis. The picture below shows an example of this application. It shows the tree structure for 20 points. A small dialog box at the right shows information about the cluster.

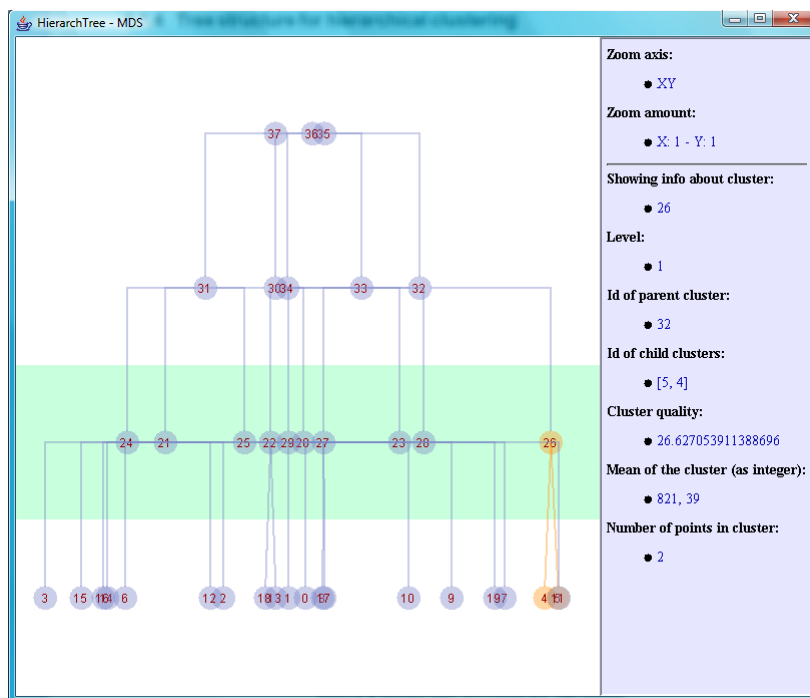


Figure 21 Shows tree structure for 100 points

4.1.5 GUI for creating custom clusters

After experimenting with the algorithms and our application, we decided it would be useful to have the option to manually create our own clusters in a simple and easy way. When we tested the algorithms there were some cases where it was desirable to work on specific clusters rather than the random clusters generated automatically by the application. This was in particular very useful when trying to produce the case of a solution lock caused by a local optimum.

The user interface is pretty basic; the mouse buttons adds one point. Or use the spray function to add many points inside a given shape. It is also possible to clear all points and to save or discard the clusters. If the clusters are saved, they are not saved to file but rather opened as current problem in the cluster application (where they can be saved as xml file). The mean (or center point) of a cluster is represented a cross with the same color as the cluster and is updated each time a new point is added to its cluster. The figure below shows a screenshot of the GUI for creating custom clusters.

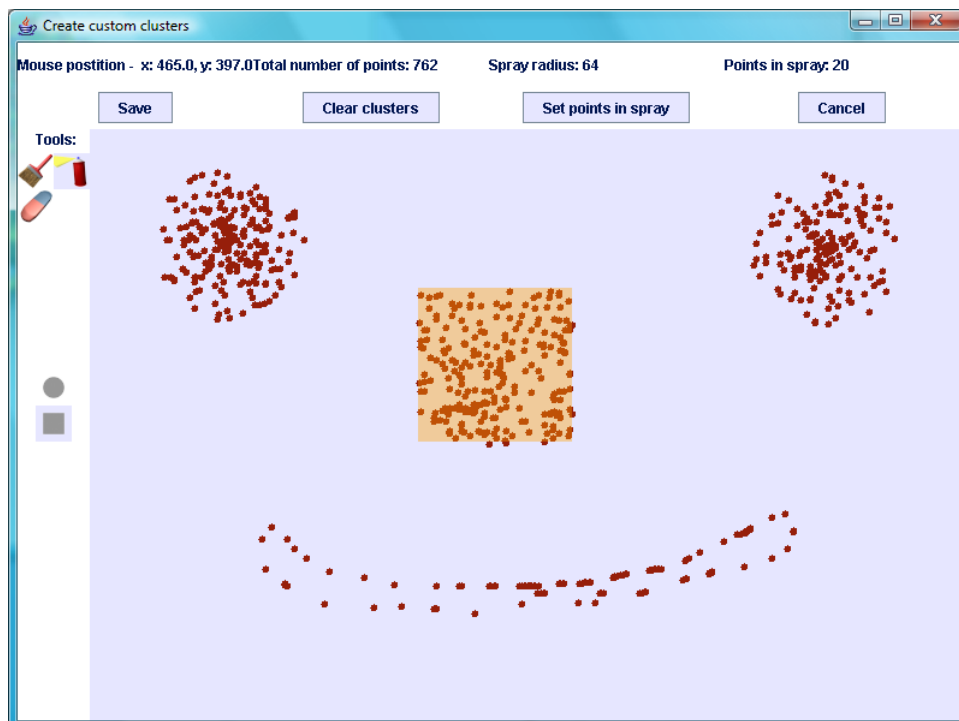


Figure 22 GUI for creating custom clusters

4.1.6 Visual representing of the clustering results

During clustering the applications saves at certain intervals, information on its current status. These intervals can be configured by the user in the settings panel. This information can then be viewed as a graph where the Y-axis represents total SSR value and the X-axis represents the time spent on the problem. The SSR value is shown both as number and the percent this number is compared to the starting SSR value.

When working on complex problems, the solving can be very time consuming. To generate average values several runs on the same cluster can be performed automatically. The user can set the number or times the problem should be solved to get an average value in the settings dialog and the average values for all the runs will be calculated without the need of any further user involvement.

It is also possible to run all algorithms on the same problem to evaluate the results. The picture below shows the progress of how the “moves one cluster to random cluster”-algorithm solved a problem. In this experiment the average SSR is reduced to about 25% (reduction of 75%).



Figure 23 Shows the result from clustering

4.2 XML

Extensible markup language, or XML, is a widely used system for defining data formats. [20]. In order to compare the different algorithms we have developed a structure which saves the different problems into an extensible markup language, or XML, format. We could not find any standards for saving cluster problems so we had to develop our own. XML is fast, easy to use and well structured. It is therefore a good way to store our problems.

The source code we used for reading and writing to an XML file is found on the internet site www.labe.cz [19]. The picture below shows how the XML file looks on in a regular web browser. The structure we created may not be the most efficient structure we could have created, but it was created with simplicity and readability in mind. If this structure has to be used with very large problems it may need some further revising (Problems with 100.000 points in two dimensions would give 300.000 values for the points and the file would be over 500.000 lines long).

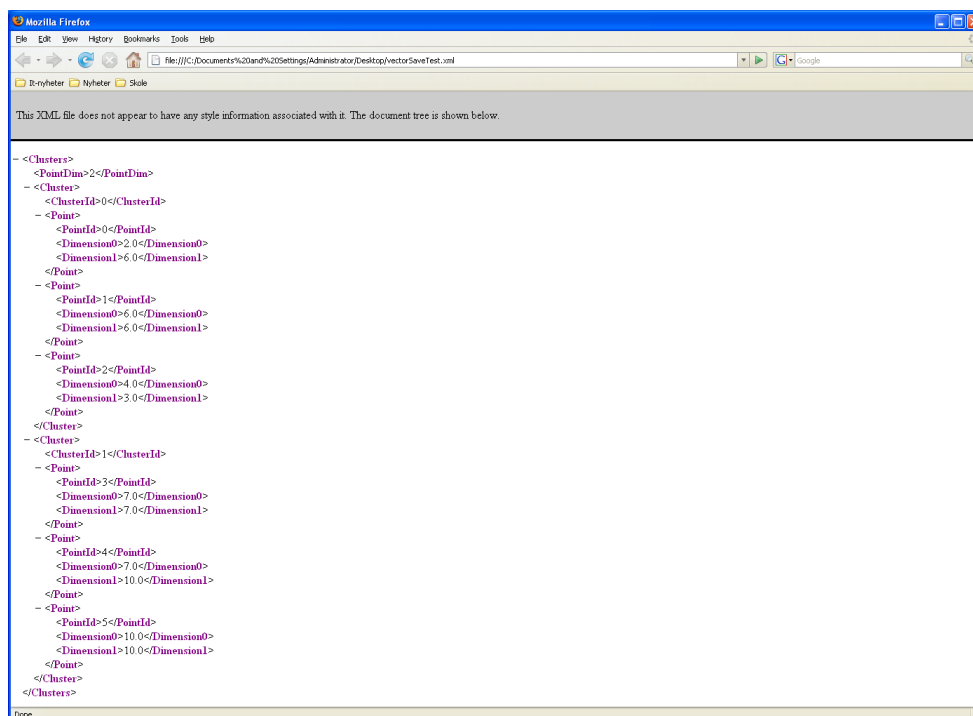


Figure 24 Shows the XML structure of two clusters

To get a better understanding for how the XML is build we have created a sample piece of the structure in the textbox below.

```
<Clusters> // Start of clusters
<PointDim>2</PointDim> // Dimension of the clusters

<Cluster> // Start of one cluster
<ClusterId>0</ClusterId> // Cluster id
<Point> // Point in cluster
<PointId>0</PointId> // Point id
<Dimension0>2.0</Dimension0> // X-value point
<Dimension1>6.0</Dimension1> // Y-value point
</Point> // End point 1
</Cluster> // End cluster
</Clusters> // End Clusters
```

Figure 25 Shows the basic XML structure of two clusters

5 Approach for partitional clustering

The purpose of chapter 5 is to explain the use of partitional clustering. We will explain the state-of-the-art algorithm for this field and explain three partitional clustering algorithms we have worked with. We will also give an overview of our implementation of these algorithms and results from experiments.

5.1 Partitional clustering algorithm

Our approach for partitional clustering is based on the state-of-the-art algorithm found in the book “*Pattern classification*” [b1]. From this algorithm we have created two different approaches that have a local search perspective. A local search approach means that the algorithm will only focus on a small part of the problem and try to find the best solution for this part instead of focus on the entire problem. This is an approach which reduces the amount of work and hopefully reduces the total solving time. Figure 26 shows the algorithm described in the book [b1].

Partitional cluster algorithm:

Give an input data (set of points) with the coordinates (x, y, z... n).
Set the number of clusters you want to create.

```
Do
{
    Choose appropriate clusters
    Choose appropriate point(s)
    Move point(s)

    Compute the gain in SSR
    If (the new-gain < previous-gain)
        accept the move
}
While (No improvement during a successive number of moves)
```

Figure 26 Pseudo algorithm

This algorithm will first choose a random cluster. From this cluster it chooses a random point. The algorithm will then find the cluster that lies closest to the point and move the point to this cluster. The algorithm will then calculate the SSR for the two clusters. If the SSR value does not improve the point will be moved back to the original position. This will be continued until the total SSR has not been changed in a given number of iterations. In our experiments this algorithm will be referred to as “one best point” algorithm.

5.2 Different partitional cluster algorithms approaches

Based on the “one best point” algorithm we have created two different approaches for the cluster algorithm. Both of them will be based on the local search approach. This means that the algorithm will only look at a small part of the problem when performing changes to the clusters. Since the algorithm only needs to work on a local problem we hope to reduce the workload and reduce the time needed to solve the problem.

1. This algorithm chooses two random clusters (C1 and C2) and two random points (P1 and P2). P1 is chosen from C1 and P2 is chosen from C2. These points will then switch place, moving P1 to C2 and P2 to C1. New SSR values for both clusters will be calculated and if this sum is lower than the original sum (or it could be within a specified threshold), the switch will be accepted. If not, P1 will be moved back to C1 and P2 to C2. This is repeated until the SSR does not improve after a specified number of runs. Since the algorithm just switches two points, the number of points in each cluster will not change, which in many cases may be a weakness. Therefore the points should be evenly distributed amongst the clusters before the clustering starts. In this report this algorithm will be known as the “two points” algorithm.
2. The second algorithm also chooses two random clusters to work with each round, but instead of switching two random points between the two selected clusters, it takes one random point from the first cluster and moves it to the second cluster. If this results in a lower total SSR for the clusters the point will be kept in the new cluster. Otherwise it will be moved back. As in the first algorithm, these steps are repeated until there is no improvement after a specified number of runs (or the new value is within a specified threshold). In contrast to the first algorithm, this does not need the points to be evenly distributed between the clusters, e.g. all the points could initially be in one single cluster and the algorithm will itself distribute the points amongst the numbers of clusters desired. In this report this algorithm will be known as the “one point” algorithm.

The reason these variants uses local search while the original algorithm does not is that they will not check distance from the selected point (or points) to find the closest neighboring cluster. With this local search approach we hope to increase the scalability to the algorithms. The only way to make sure you have the closest cluster is to check distance to all available clusters.

5.3 Experiments

In this section we will present the results we got from our experiments. In these experiments we tried the algorithms on several problems of different complexity. The intention of these experiments was to find out how the algorithms worked on the different problems and to see if we could discover any weaknesses. This section also explains the problem of local optimum and gives a short example of how this may be solved with our algorithms. Finally we have discussed and analyzed the results and given some comments regarding further work.

5.3.1 Setup

In our experiments part we tried the three different algorithms on the following problems:

- 4 clusters 400 points
- 4 clusters 2000 points
- 10 clusters 1000 points
- 10 clusters 5000 points
- 60 clusters 6000 points

In addition we tried to create a problem that would cause the algorithm to get a local optimum. Local optimum is a state where the optimal solution is not found, but there is not one single step that can improve the solution. In order to fix this, some bigger restructuring is often needed, which first has to make the solution worse before it can get better and the optimal state is found. An effect of this can be that the algorithms gets stuck, unable to undo the local optimum even if would be for the better. One cause of local optimum may be when there are some “noise” elements. By noise we mean objects which belong to no apparent natural cluster. So if an algorithm is sensitive to noise, it may also get stuck in a local optimum. In that case we would try to see how this problem could be solved.

These problems were chosen to see how the algorithms worked on problems with different complexity. We have then evaluated the quality of the clustering. In this context, the quality is based on a combination of speed and precision; the algorithms should reduce the SSR value as fast as possible. We tried to cover the cases from having few points and clusters to many points and clusters, and some in between.

All points were generated with a Gaussian distribution model. This distribution of points has the potential to get a high reduction off SSR value if the groups are recognized by the algorithms. SSR is a very good measure to use when finding clearly defined clusters. If the distribution of the points within these clusters is distributed Gaussian the potential for reduction in SSR is great. This is because a Gaussian distribution will have a high concentration of objects located around the center of the cluster and thus would be a compact group.

For each algorithm we made 100 test runs on the same problem and calculated the average time and reduce in SSR. The algorithm will check SSR every 100 rounds, and when the algorithm has done 30 runs on a row without any improvements on SSR, it will stop. All GUI updates are turned off to make the calculations as fast as possible.

The experiment was done on a Tundra Pentium Centrino 1.7, 512 Mb ram with Windows XP sp2.

To give the possibility to later test these results, all the initial problems are saved to XML formats which are attached as an appendix to this report (CD-ROM).

5.3.2 Results

Results from 4 clusters 400 points

Clustering 400 points into 4 is a small problem which is relatively easy to solve compared to the other problems we cover by these experiments. From the graph below we can easily see that all of the partitional algorithms that we cover reduce the SSR rapidly.

Observe that when the SSR is reduced approximately to 10% the calculation stops. As seen in the figure, the “one point” and the “one best point” algorithms gives the best results, with the “one best point” being a little bit faster. The “Two points” algorithm is not as efficient, it uses quite a lot more time to get to a certain reduction of SSR, and it ends up with a worse solution than the other algorithms. Given more time, the “Two points” could possibly get to the same reduction as the other algorithms, but it would take a lot more time. And if the number of points in the generated groups were uneven, the “Two points” would be unable to find the best solution.

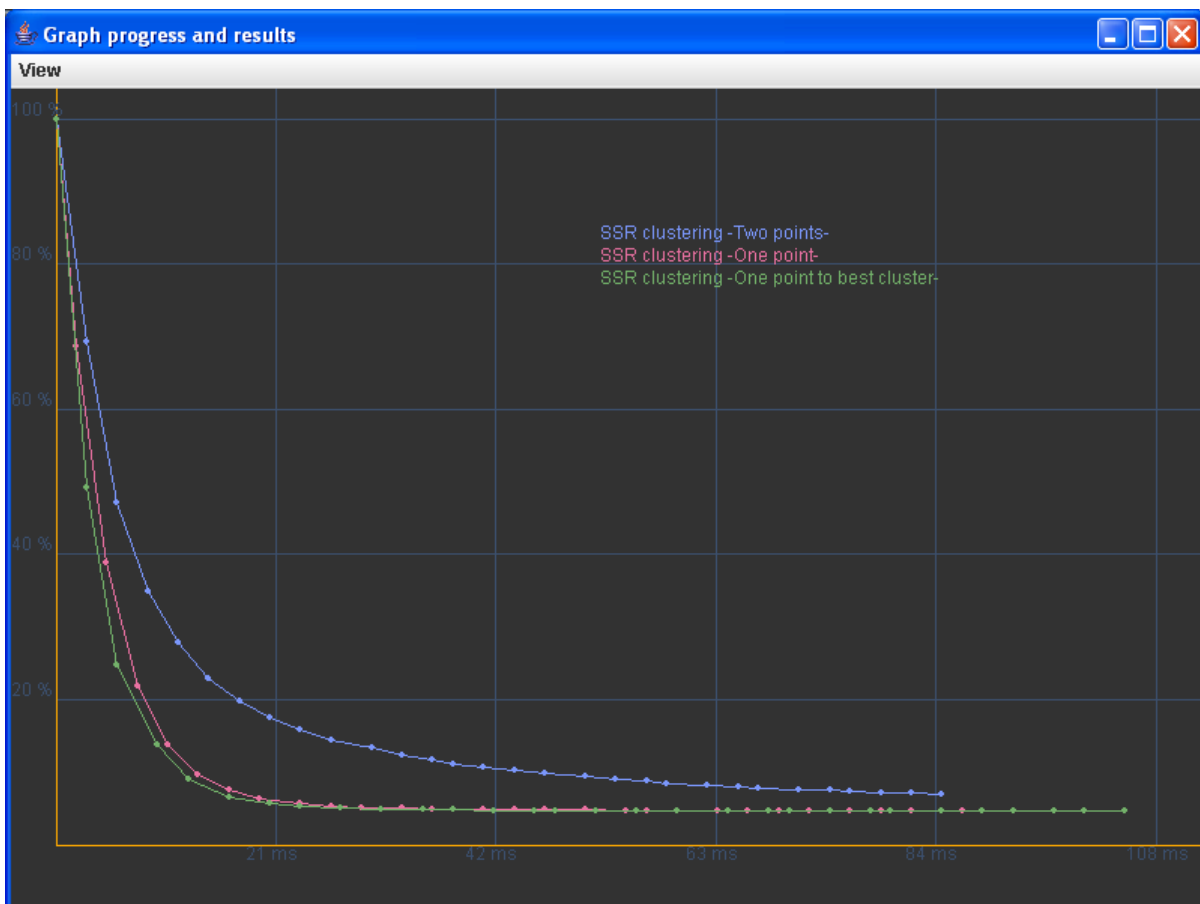


Figure 27 Compares results of 3 algorithms clustering 4 clusters and 400 points

Results from 4 clusters and 2000 points

In the 4 clusters and 2000 points experiment we can see from the graph that the results of the different algorithms are starting to become quite different. The “one point” and “one best point” algorithms still give the best results. Both of them reduced the SSR to around 10 percent of the initial value. It is also interesting to see that the “two points” algorithm gives the worst results at any given time.

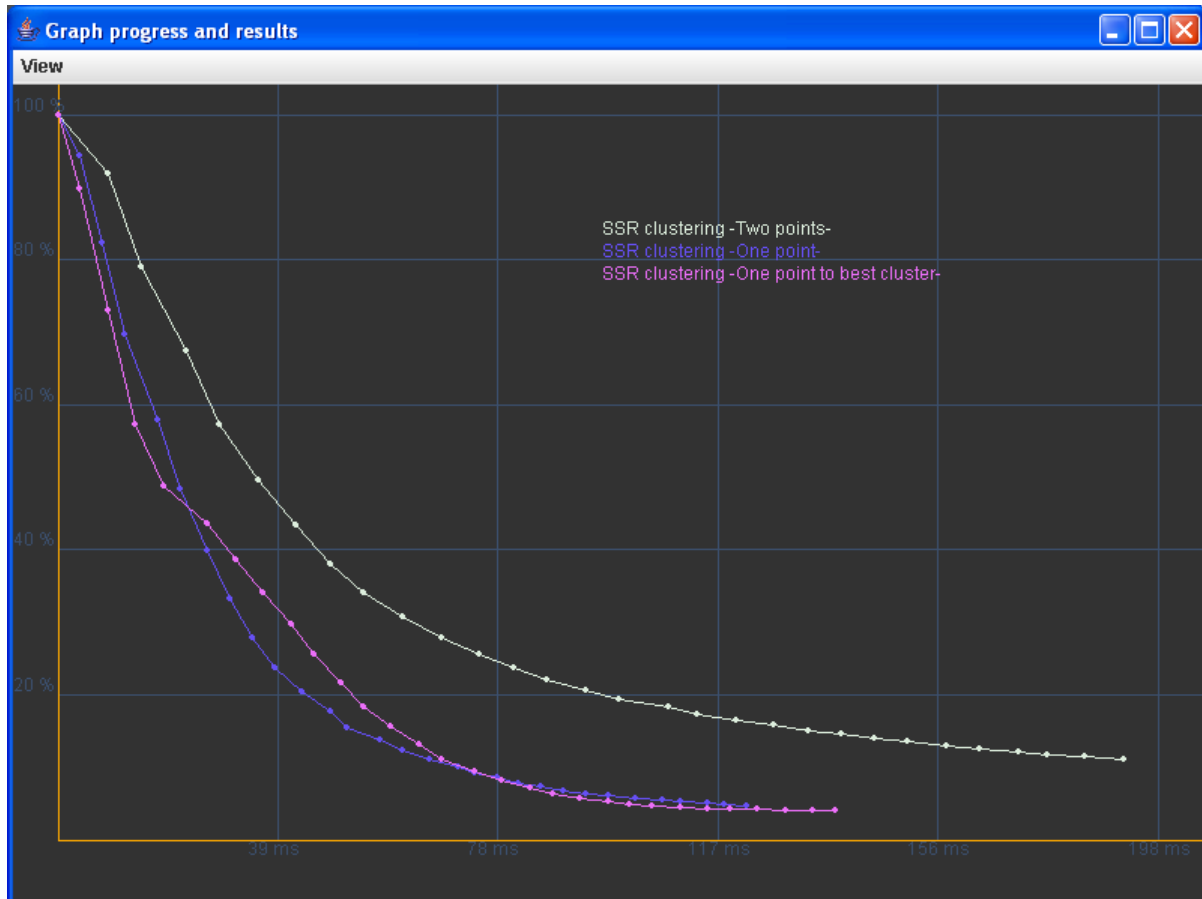


Figure 28 Compares results of 3 algorithms clustering 4 clusters and 2000 points

Results from 10 clusters 1000 points

On this problem, the “one point” and “one best point” algorithms still gives the best result. However, the “one best point” starts to spend more time compared to the “one point” algorithm. In the beginning, before the clustering reaches a certain level, there is not much difference between these algorithms. But as the solution gets better, the “one best point” hits a critical point and starts to flatten out.

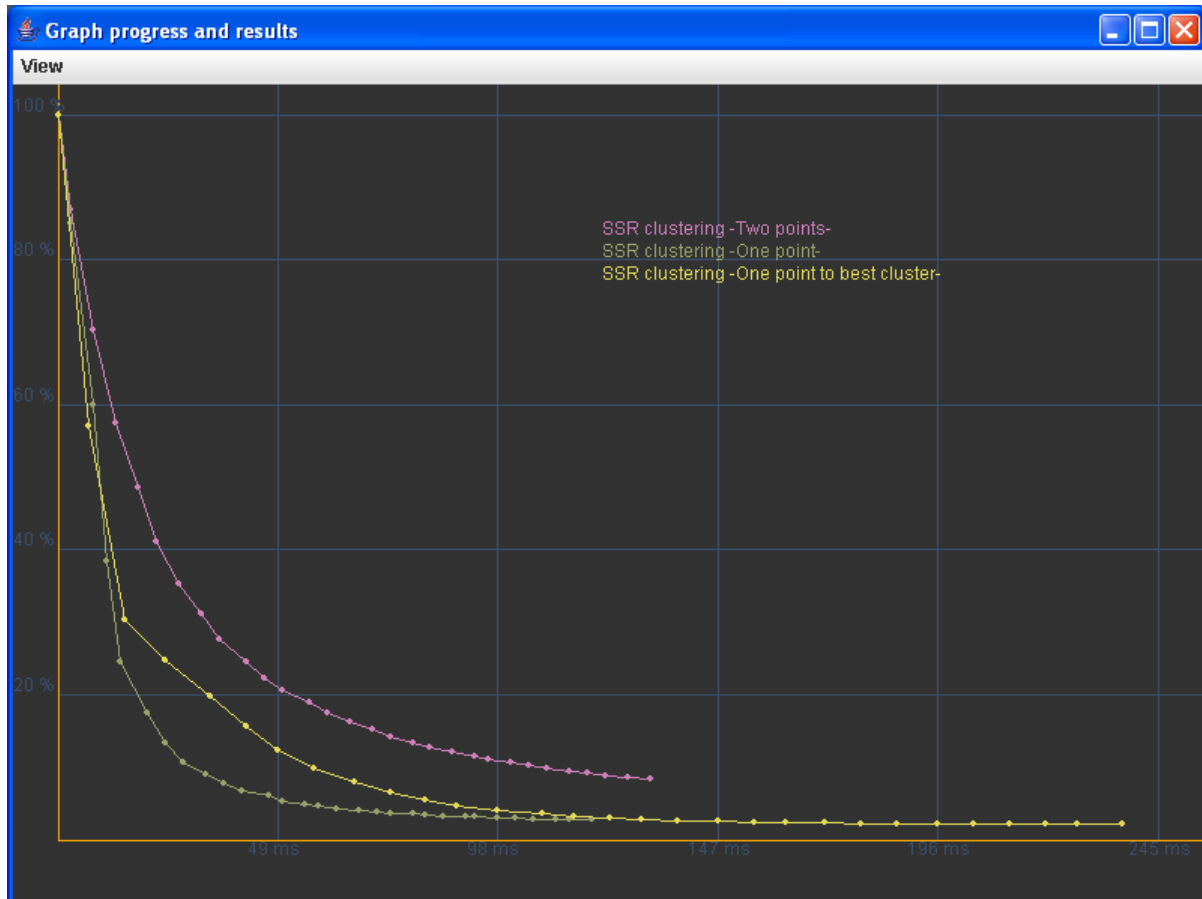


Figure 29 Compares results of 3 algorithms clustering 10 clusters and 1000 points

Results from 10 clusters 5000 points

The result of this experiment is similar to the results of the previous experiment, but for a while the “one best point” algorithm provides, for the first time, the worst solution. As the complexity of the clusters rise, the “one best point” algorithm seems to change from being the fastest variant to be the slowest. Still, given enough time it will in most cases give a better final result than the “two points”, but so will the “one point” only quicker.

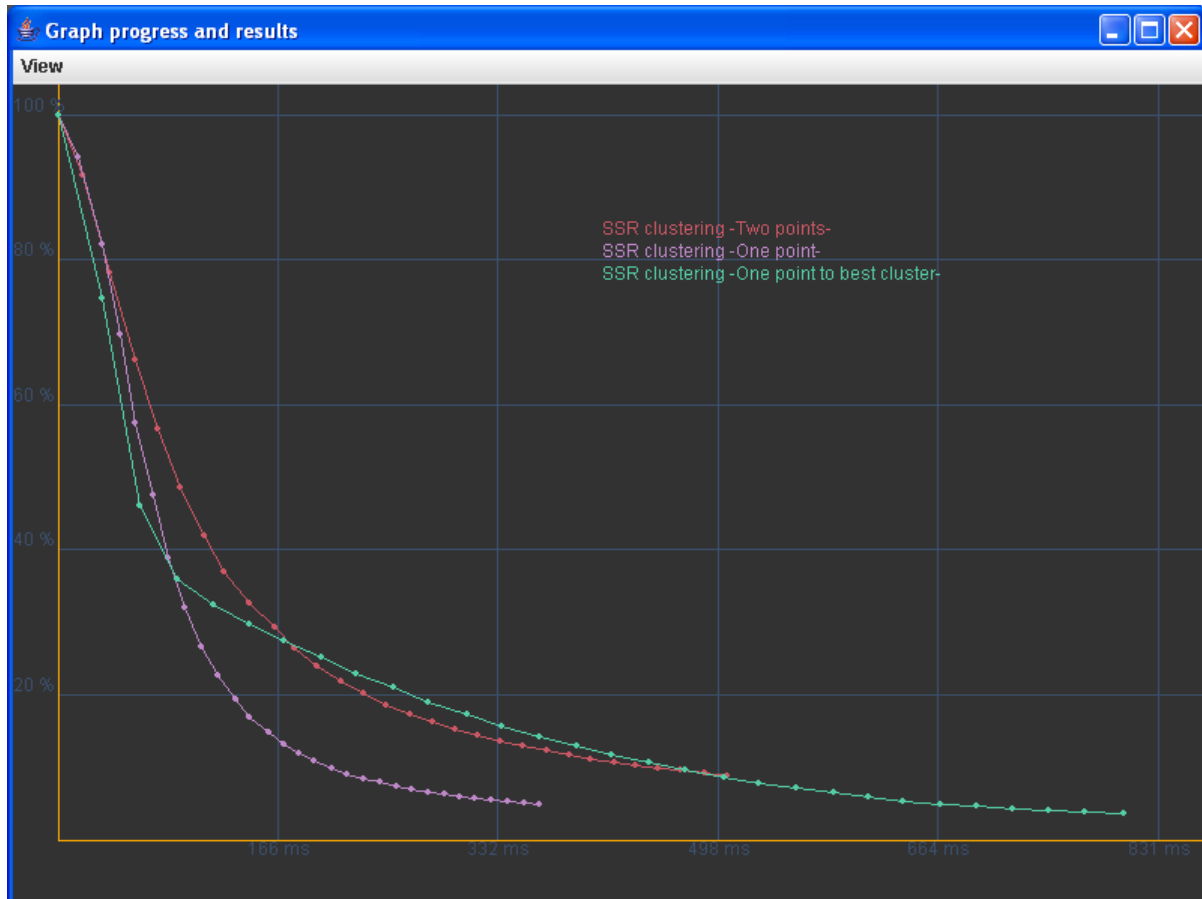


Figure 30 Compares results of 3 algorithms clustering 10 clusters and 5000 points

Results from 60 clusters 6000 points

In this case the “one best point” is clearly slower than the other algorithms. From the graph, we can see that it uses almost four times longer to get to a solution which also is not as good as the solutions of the other algorithms. This is because the calculations needed to decide which cluster to move to gets increasingly time-consuming.

Still the “one point” remains the best algorithm regarding the result at any given time.

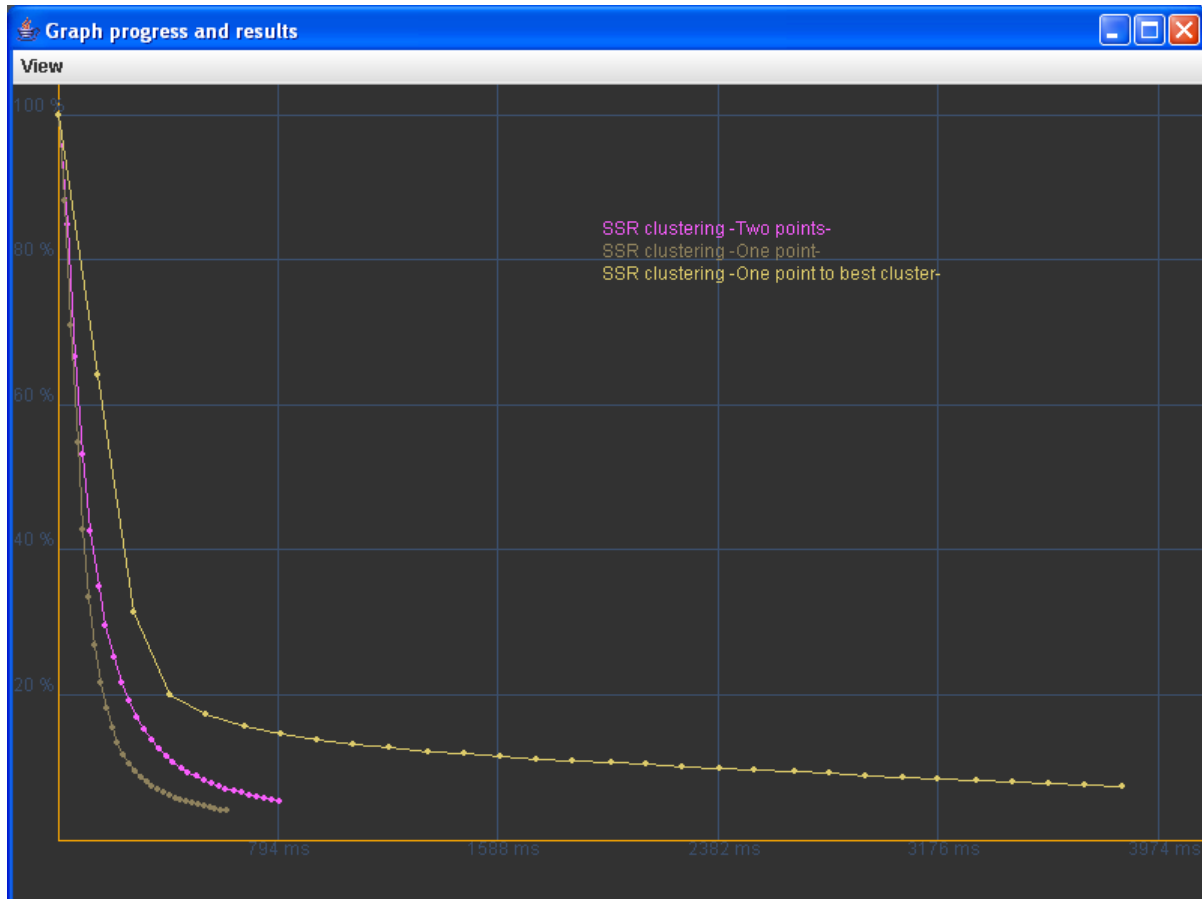


Figure 31 Compares results of 3 algorithms clustering 60 clusters and 6000 points

5.4 Local optimum experiment

As mentioned earlier in this report, there are some cases which may trap the algorithms in something called a local optimum. Local optimum is a state where the optimal solution is not found but further improvements would require more than a single step. In order to fix this the solution would have to get worse before it can get better.

If the algorithms then are configured to not accept any point moves which does not give a better SSR value, the algorithms will be locked. An example of a clustering solution causing a local optimum can be seen in the figure below. Here there are three groups with 20 points in each group. In addition there are some “noisy” points outside the main groups (10 blue points). These points should be clustered in three clusters. In the figure the noise elements have been mistaken for a separate cluster when it really should be a part of the green cluster. This mistake is forcing two of the actual clusters into a single cluster. The clustering algorithm we used was both the “one point” and “one best point” algorithms. The reason why we did not use the “two points” algorithm is that the number of points in each cluster should be dynamic, rather than fixed.

There is not one move of a point or a switch of two points which will improve the solution. Still, it is quite easy to see that this is not the optimal solution.

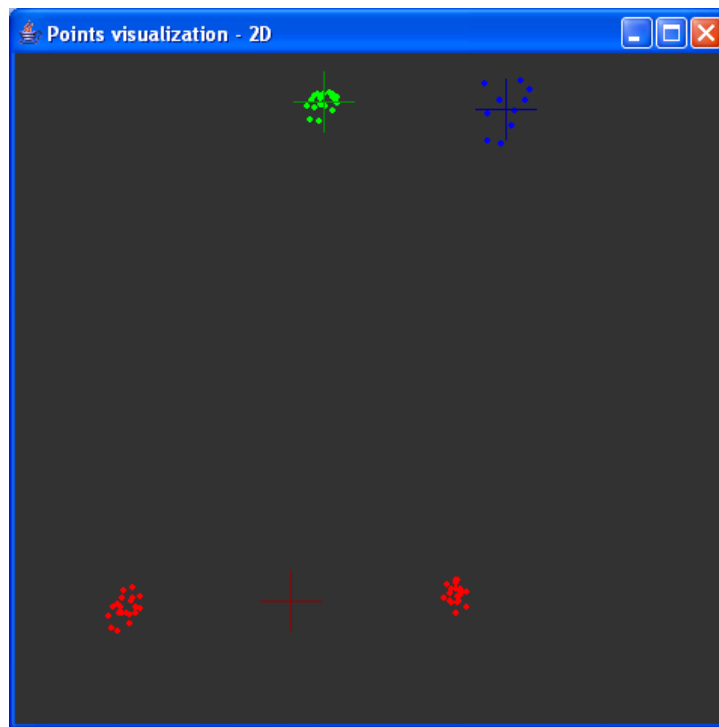


Figure 32 The local optimum problem

The solution to this problem is to sometimes accept a move of points even though it may not give a better solution. We then changed the settings of the algorithm as shown in the picture below.

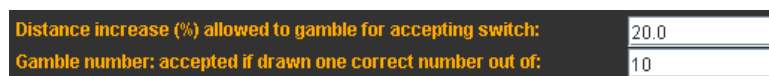


Figure 33 Settings to solve local optimum

This means that if a SSR value after a move of points is worse, but within 20% of the previous value, the move of points still has 10% chance of being accepted. When we then tried to solve the problem, we got a new result, as seen in the picture below. Note that the colors themselves are only defining the clusters. In this figure the blue points are not noise but a proper cluster (now containing the noise objects).



Figure 34 Local optimum problem solved

5.5 Experiments summary

Two point's algorithm

From the results of our experiments this algorithm seems to be quite scalable throughout all the test cases. However, this is the only algorithm we are testing that does not change the number of points in each cluster, so the points have to be distributed before the algorithm can start. In certain scenarios where it would be optimal to have an uneven number of points in the clusters, this algorithm would have its limits. If there is a point in one cluster that doesn't really belong there, it has to find a suitable point in another cluster to switch with. This may be a limiting constraint and part of why the solutions of this algorithm never get as good as with the other algorithms in our tests.

When dealing with noise and local optimum, this can be avoided by sometimes allowing the points to be switched, even if this does not immediately improve the solution. But once again the fact that this algorithm does not change the number of points in each cluster may prevent the algorithm from finding the optimal solution.

One point algorithm

In three out of five tests this algorithm gives the best result. In the rest of the tests it is a bit slower than the "one best point" algorithm, but it still gives comparably good results. When the numbers of clusters are high, in the cases of 10 clusters and 1000 points and above, it gives the best results. Also, it will distribute the points dynamically and it should put more points in some clusters if needed, not having to rely on getting another point to switch with. This would make this algorithm suitable for detecting clusters of uneven size, as opposed to the "Two point's algorithm".

The problem with a local optimum is solved in the same way as with the "Two point's algorithm", by sometimes allowing a point to be moved even though it does not improve the solution. This algorithm also seems to be scalable and generally one of the fastest algorithms amongst the ones we have tested. As the number of clusters increase, it will prove to be the best algorithm and should generally give a better result than the first algorithm both regarding time and quality. On average the SSR is reduced by at least 90% in these cases.

One best point algorithm

This algorithm is really fast as long as the numbers of clusters are low. In almost all of the small experiments it is clearly the fastest one, with about the same SSR value, or better, than the one point algorithm. It also distributes the points amongst the clusters dynamically, just as the "One point algorithm". But when the number of clusters increases, the performance degrades quickly. This is because it does not utilize the local search principle and therefore has to iterate between all the clusters to find the closest one to its middle point. In other words, the algorithm scales poorly when it comes to the number of clusters, but will still scale better when it simply comes to number of points without affecting number of clusters. This is something that really shows in the last experiment where it is up to 10 times slower to get to a solution. From our experiments, this algorithm seemed to handle noise and local optimum in the same way as the "One point algorithm".

The local optimum problems we worked with, were solved equally well by both of these algorithms. But the fact that this algorithm moves a point to the closest cluster, rather than a random one, may affect how easy a local optimum is solved. In the cases where some points have to be moved further away rather than to the closest cluster, this algorithm may have a hard time finding the optimal solution. This will be more probable when having more clusters and the solution to the local optimum requires a big restructuring of the clustering.

As seen on the graphs of the tests (especially with more clusters) the SSR value will decrease quickly to a certain point, but after this it will flatten out a lot and use a long time to do minor improvements. At the beginning of the clustering process when the points are badly clustered, most of the points are placed in the wrong cluster, and there is quite a lot to gain when finding the right cluster to place a point. But as the clustering is getting better, the algorithm is starting to more often than not choose points that already are in the correct clusters. This will generate a lot of unnecessary work just to check whether these objects can be moved. That is probably the reason why the algorithm is working as best until it reaches a critical point after which it almost stops to a halt.

Clearly, this algorithm does not seem to be as scalable as the other algorithms when the number of clusters increases. But with a low number of clusters it is clearly the fastest of our algorithms, with a result at least as good as with the one point algorithm, if not even better.

6 Approach hierarchical clustering

We have tested our dendrogram algorithms on a set of different experiments, designed to reveal their essential properties. As a result of these experiments, we have obtained valuable information about the algorithms. It was important to find out how scalable they are, as well as learning more about the quality of the clustering.

The cluster switcher algorithm generally needs less computational time per iteration, while cluster mover uses a more flexible technique. Our experiments indicate that cluster mover is the best algorithm both regarding clustering speed and scaling, not unexpected according to our early research.

6.1 Online hierarchical clustering algorithm

Our approach for hierarchical clustering is based on a state-of-the-art algorithm found in the book "*Pattern classification*" [b1]. This was the starting point for our research and we wanted to see what improvements could be made and how effective we could get the clustering. The next section provides more information on two main types of clustering and how our algorithm differs from these.

When it comes to clustering, there exist a number of different algorithms with various properties. Clustering can be divided into partitional and hierarchical clustering. Until now, partitional clustering has been the fastest method, but it only contains information on a single solution with a fixed number of clusters. Hierarchical clustering provides a structure, where multiple solutions with any desired number of clusters could be represented. The main disadvantage of hierarchical clustering has been that it is very time consuming. A new algorithm that combines the flexibility and strength of hierarchical clustering with the speed of local partitional clustering would be of great value in many fields. This was the main idea and the aim of our work. However, in order to achieve this there were several obstacles that we needed to overcome.

In this project we had some different approaches to solving the cluster problems which we describe in the next sub chapter of this chapter. We have actually been studying five different methods. Two for clustering into a dendrogram structure and three for clustering into a tree structure where the nodes can have more than two children each. The three methods for the non dendrogram structure were abandoned. The main reason for this decision is that sub clusters are easier to locate using a dendrogram. If a node has more than two children you actually skip some layers which might be important. If a node has three children (for example) you lose some information which might be important. Two of those children could form a natural sub cluster. Instead of the three children solution there should be an additional layer with a node having two of those nodes as its children. This new node could then be the child of the original parent node. This way the original parent node will still contain the exact same objects but you will also have the natural sub cluster as a separate child. If a cluster only got one child, that child would be a copy of the cluster itself which will be useless duplication of data. The three methods are explained, but our main focus will be on the two dendrogram clustering algorithms.

The idea behind these algorithms has been carried out after several discussions with our supervisor and internal discussion in the group using the book “*Statistical concepts and methods*” [b2] as a source of inspiration. The algorithm we used as a starting point in our project is explained below.

Hierarchical cluster algorithm:

Give an input data (set of points) with the coordinates (x, y, z... n).

Build tree structure:

```

Do
{
    Create a tree structure by merging one cluster (xi) with another cluster (xj)
    from the same level to make a new cluster at level + 1
}
While (NOT Cluster level size equals max -1)

```

Clustering:

```

Do
{
    Choose random level in tree
    Choose appropriate clusters
    Choose appropriate point(s)
    Move point(s)

    Compute the gain in SSR

    If (new gain < previous gain)
    {
        accept move
    }
}
While (No improvement during a successive number of moves)

```

Figure 35 Online hierarchical clustering algorithm

As stated earlier, the closest agglomerative algorithm is the hierarchical algorithm we have decided to use as a comparison to our algorithm. This algorithm will create the tree structure by always merging the two clusters that are closest to each other. The result of this process is a clustered tree structure. Even though this is a simple procedure, it scales very poorly (One additional object in a set with $n-1$ objects would generate $\sum_{i=1}^n n - i$ additional comparisons of clusters during work) and a lot of processing is required as the number of objects is high.

Our method is a bit different. It is divided in two phases; one where the tree is created and one for improving the tree. The reason for this is that the improvement algorithm, or clustering algorithm, needs to have a tree structure to work with. Doing this provides several advantages. Firstly, algorithms doing this in one phase may not have a complete tree structure at all until the clustering is completed, meaning it will not have any results at all before it is finished.

Some algorithms (for example K-means) do the clustering in steps, where each step can be stopped and a partially clustered result can be obtained. As the time complexity of these steps increase with the size of the problem, each step can take a significant amount of time. If the clustering should be stopped during a step, it either has to wait for this potential time consuming step to finish or do a rollback to the previous completed step to get a result.

With our algorithm, the tree can be created very fast and the tree structure can be improved for as long as desired. The clustering can be aborted at any time and there will still exist a tree structure that may be usable. Since each clustering step requires a small amount of computational time, almost independent of how complex the problem is, the clustering can be stopped almost instantaneously. If needed, a tree can later be improved further by the algorithm, using the previous tree as a starting point.

It also has a big advantage when it comes to dealing with new data. If some new elements should be inserted into an existing tree, these could then be placed randomly in the tree. The clustering can then be started (or it might never have been stopped), and the new elements should be correctly clustered into the existing tree. The other methods would have to redo the whole clustering, discarding any previous work.

6.2 Different hierarchical cluster algorithms

We have divided the problem in two parts; first a hierarchical tree has to be created, then the clustering can start. Based on the pseudo hierarchical cluster algorithm, we have implemented six different approaches for creating the hierarchical tree and five clustering approaches that we have tested in this project. Amongst these approaches, there are two main types; hierarchical tree and dendrogram. Dendrogram is a specific variant of hierarchical tree, where each node, except the leaf nodes, has to have exactly two child nodes. On each level two clusters are combined into a new cluster.

The hierarchical tree was the first variant we implemented. An example of this kind of tree is shown in Figure 36. It generally has a structure of two nodes on the top level, four on the next, then eight and so on. It is also possible to have nodes with one, two, three or more child nodes. This is something that changes during the clustering when moving the nodes. Moving or switching nodes can only be done on the same level. For example it is not allowed to move a node from a higher level to the bottom level. In that case there would be a node on the bottom level containing several child nodes, when nodes on that level are supposed to be leaf nodes. With this structure there will probably not be one level representing the natural clusters, which may be desirable. To get levels with natural clusters they will have to be forced to have the correct amount of clusters (equal to that of the number of natural clusters) when creating the tree in the first place.

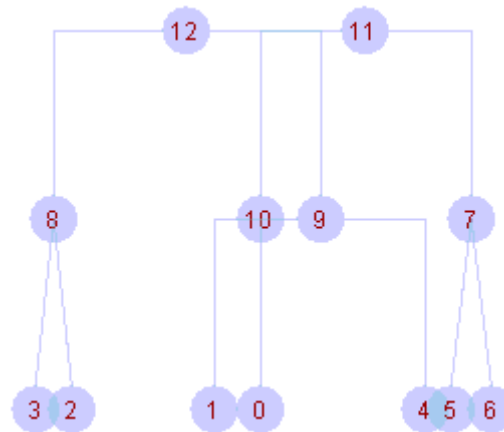


Figure 36 Example of a Hierarchical tree

As a result of our discussions about the hierarchical tree, we implemented the dendrogram. In Figure 37 an example of a dendrogram is shown. As stated above, this is a specific variant of the hierarchical tree where each node should have exactly two child nodes, except for the leaf nodes. The idea of having nodes stored in levels also gets somewhat less importance here. In the non dendrogram structure all levels would always contain all objects, and each level would thus contain one partitional solution (many of the levels would contain a semi-clustered solution since not all levels contains the “correct” number of clusters). We have kept the levels solely for the visual representation of the tree. In the tree-viewer there will be only one new node for each level, which means that there is the same number of levels as there are points. There will be a level for all possible number of nodes, and the number of natural clusters will be represented at one level in the tree. Since the actual clustering does not have any knowledge of levels, and that each level is not a representation of all objects it is allowed move/switch nodes regardless of position in the tree, as long as some important rules are obeyed. These rules are explained in part 6.3.1. This structure is a lot more flexible than our non dendrogram structure, with a bigger variety of possible moves.

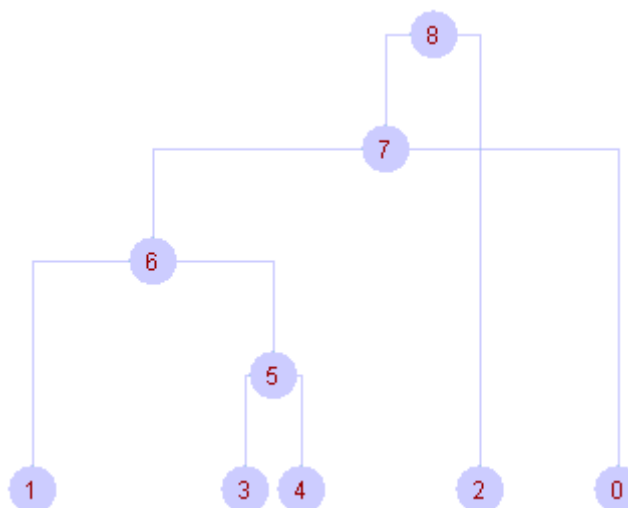


Figure 37 Dendrogram example

In the next section we will present the algorithms we have implemented for both initializing and clustering the tree.

6.2.1 Algorithms for initializing the tree

This part contains a summary of the algorithms we have implemented to create the hierarchical tree. 1 to 4 creates trees where nodes can have 1 to n numbers of child nodes and exactly one parent, while 5 and 6 creates dendrograms.

We wanted to try different methods for creating the tree. The first algorithm we created went through the list of points and created the tree in that order, but then the structure of the tree was determined by the order the points had in the list. This was a fast algorithm, but we also wanted a method that creates a random tree, unaffected by the order of the points and which would be different each time. We also wanted to experiment with doing some clustering during the initializing of the tree, so we created a few algorithms for this as well. The main focus of this project has been to improve the clustering algorithms itself, but it was interesting to see how adding clustering to the initial phase influenced the result.

Next we will present the different tree initialization algorithms we implemented.

Non dendrogram:

The non-dendrogram was for a long time the only structure we worked with. It worked well with clearly defined clusters, but appeared to get into trouble very quickly if the structure was not so well defined. Also this structure was locked on only having to make changes to single levels. This will in many cases make it need more iterations to solve a problem since it often will have to sort the lower layers before the higher layers can be fixed. We did discuss the possibility of weighting the levels so that the algorithm would more often than not chose the lower levels, but we had at that point gotten the feeling that this structure is a bit limited and discontinued working with it. We still discuss the methods for it because it was a very important step in the evolution of our algorithms.

1. Fast

This is the fastest and simplest algorithm. The tree created with this algorithm will be affected by the order of the points in the list. It works well in the cases where the points have random positions in the list and when the tree should be created fast, without any clustering done.

The algorithm iterates through the list of points and puts two and two points in one cluster. If there is just one point left in the end, that point will be the only one in the last cluster. This is then repeated for the next level; the two first clusters on the previous level will form a new cluster and so on. The algorithm stops when it reached a given number of clusters on the topmost level. . If order of the points are random before building the tree this method will be random, but the result will be the same every time for equal input sets.

This method was used mainly for internal testing when we needed a quick way to generate the tree structure and not for any of our tests in the experiment phase.

2. K-mean

One idea we had was to create the tree with K-mean clustering on each level. This is a fast method for creating a clustered tree and can be used as a comparison to our algorithm.

Initially all points are put in one cluster, and then this is clustered into two clusters using the K-mean algorithm. These two clusters will be the top level of the hierarchical tree. Each of these clusters will again be clustered into two new clusters and the new clusters will then be the next level. This process is repeated until there exists a level with one point in each cluster; this will be the bottom level. This is a divisive approach to creating a hierarchical tree, and can actually be thought of as a way of both creating and clustering the tree since each level will be clustered using the k-means algorithm. The number of clusters each level should be divided into could be set manually if it was to be used.

This method was an idea we had early in our work. It seems to work fast and generates a decent quality clustered tree. But because we wanted to test our own algorithms on totally random trees this method was discontinued and not used in any "official" testing.

3. Random

As the main focus of our project has been to test and improve the clustering algorithm, it was very important that we had a way of creating an absolutely random tree. By testing it on random trees, we ensure there are no pre-determined qualities needed for the clustering to work.

This algorithm is very similar to the first algorithm, but instead of just picking points and clusters sequentially from the beginning of the list, they are picked randomly. The random functions and the need to keep track of already used objects makes this algorithm a little bit slower than the “fast” algorithm, but the result will be completely random where clusters are not affected by their position in the input set. Also this algorithm will create different trees almost each time it is run even if the input set is equal (the only times it will generate equal trees is when it happens by chance).

4. Closest mean

This was another idea we had for quickly creating a clustered tree. It was in order to have something more to compare our algorithm to. And also to see if it could be worth spending a bit more time during the initialization process in order to have a better starting point for the clustering.

One cluster will be selected from the list; this will then be merged with the closest cluster to form a cluster on the next level. This is repeated for all the clusters on the first level, and then for all clusters on the next levels, until there is only two or three clusters in the top level. This method will not merge the closest clusters available. The second cluster selected could very well have another cluster which it is closer to it, even if the first cluster (selected randomly) does not. With this algorithm the result is often well defined clusters which are closer to a clustered result than the random ones, but with some misplaced points. This is because when the last points/clusters is chosen, the closest points/clusters may be already had been taken and they has to merged with some further away.

This method was a very quick way of generating a partly clustered tree. It could very well be more research on this would prove that there could be benefits of using a quick method like this before actually performing the clustering. As with the k-means tree-creator algorithm we it was discontinued because we didn't want our algorithms to have to rely on non-random structures to work.

Dendrogram:

Realizing that the structure we had been using had some limitations to the way changes could be made, we decided to use a dendrogram structure instead. Along with the change to dendrogram we also removed the concept of levels since we wanted to be able to make relation-changes to nodes regardless of where they were positioned in the tree. The dendrogram structure is the one used in all our hierarchical experiments.

5. Random dendrogram

When our algorithms evolved into working with dendrograms, we needed a way to create a totally random dendrogram.

Random points/clusters with no parent will be merged together, until one top-level cluster containing all points is created. This is a fast way of creating the dendrogram, and as it is completely random, it is a good starting point for our clustering algorithm.

6. Merge closest (closest agglomerative)

In order to validate our algorithm, we wanted to compare it to the state-of-the-art algorithm. This method creates very well arranged trees, at the cost of high computational time. We wanted to compare both the clustering speed and quality between our algorithm and this method.

Each point starts out alone in a cluster. Each step of the algorithm will merge the closest clusters available. When two clusters are merged they are taken out of the list of possible candidates and the newly formed cluster will be added instead. This is repeated until no further steps are possible, meaning there is only one cluster left. This is a systematic algorithm to find a solution close to a “perfect dendrogram” (where all objects are siblings to their closest neighbor), but as the number of points increase, this algorithm soon becomes extremely slow. The solution will often have minimized total SSR as much as possible and therefore it is interesting to see how close to this our algorithm will get and compare the time used. As explained in chapter 2.11.3 there are some cases where the closest agglomerative solution will not generate the absolutely minimal SSR solution, but the value will still be close to the theoretically lowest possible.

6.2.2 Clustering algorithms

Between the non dendrogram trees and the dendrogram clustering approaches there are two very important differences: while the non dendrogram tree algorithms switch/move clusters on the same level, the dendrogram approaches operates across the tree. One cluster can be switched with or moved to another cluster no matter where it belongs in the tree, as long as some preconditions are fulfilled. One example of a precondition is that there is no point in switching two clusters if they are siblings. Also there are some moves which are not possible, i.e. to attach a node as a child to its grandchildren makes no sense and will not be allowed.

Non dendrogram:

1. Switch two random clusters

This was our first clustering algorithm on the hierarchical tree. This is an expansion of the partitional algorithm we worked with in our previous work, described in chapter 5.2. We now wanted to see how this way of thinking worked on a hierarchical structure.

The algorithm:

One random cluster is picked (c_0), and then another random cluster on the same level is chosen (c_1). These two clusters will then be switched, meaning that the parent of c_0 will lose the points in c_0 and get the points in c_1 and likewise for the parent of c_1 . The new quality of these parents is then calculated and compared to the old qualities; if the sum of these new qualities is lower, the switch will be accepted, if not they are switched back.

2. Move one cluster to random cluster

As we discovered with our previous work, switching two clusters has its constraints. When there number of clusters that should be moved is low compared to the total number of clusters, the algorithm may need a lot of time before selecting two clusters appropriate for switching. In some cases there may not even be any clusters causing a good switch, even though some clusters should be moved to another node in the tree. An algorithm moving a cluster rather than having to find an appropriate switch should then be more flexible and effective.

The algorithm:

One random cluster with some siblings is picked, and then a suitable parent on the level above is found. The cluster is moved from the old parent to the new one and if the sum of qualities for the parent clusters is better than the old one, the move will be accepted. Else the cluster is moved back to the old parent. The reason why the cluster to move has to have at least one sibling is because if not, the old parent would not have any clusters left when the cluster is moved.

3. Move one cluster to closest cluster

In addition to trying to move a cluster to random target cluster, we also wanted to see how putting some work into finding the best target cluster would affect the algorithm. This would mean that all iterations in the algorithm would take more time, but should do more sensible moves.

The algorithm:

Almost the same as “move one cluster”, but instead of choosing a new random parent, the closest cluster on the level above will be chosen as the new parent. The cluster is then moved from the old to the new parent and if the sum of their new qualities is better than the old, the move is accepted.

Dendrogram:

4. Cluster switcher

As we changed our structure to dendrogram, we still wanted to see how our initial clustering method would work. New with this structure was that the algorithms could switch/move clusters across the different levels, which enables a lot of new possible moves.

The algorithm:

Two random clusters (c_0 and c_1), which satisfies different conditions is selected. These clusters will then switch positions, and if there is a quality gain in the switch, it will be accepted.

The conditions the clusters have to fulfill include:

- The two clusters cannot be siblings (switching two siblings makes no difference)
- Neither can be the top-level cluster
- Neither of the clusters should be an ancestor of the other cluster.

5. Cluster mover

As discovered in our previous work, the method of moving a cluster to another random cluster has been the overall most effective way of working. It was important to have this variant on the dendrogram structure as well and evaluate the performance.

The algorithm:

In this algorithm one random cluster will be moved to a new parent. If there is an acceptable quality gain in the parents (p_0 and p_1) after the move, it is accepted. In this algorithm there are no restrictions to the parents; they can be at any level in the dendrogram. But as moving a cluster would result in p_0 ending up with less than two children and p_1 ending up with more than two children, some restructuring of the tree has to be done in order to keep the dendrogram structure. Also this algorithm has some restrictions to the selected clusters. Note that the first selected cluster is the cluster which will be moved while the second cluster defined where the cluster will be moved to. After a move the two selected clusters will be siblings.

- The first clusters selected cannot be the top node (Moving all the points in the problem to another parent make no sense).
- The clusters should not be siblings.
- Neither of the clusters should be the other clusters parent.
- The first cluster to be selected should not be an ancestor of the second cluster (the other way around is fine).

6.3 Description of algorithms

6.3.1 Clusters switcher

The cluster switcher algorithm will select two clusters and try to switch parents of those clusters. Quality for the parent clusters will be recalculated and finally the move will be either accepted or rejected based on the quality after switch versus the quality before the switch. In this explanation the clusters selected will be called c_0 and c_1 and the parent of the clusters will be called p_0 and p_1 .

Pseudo Algorithm:

```
While (not finished)
  Select  $c_0$  and  $c_1$  from clusters
  Store old values.
  Switch parents.
  Calculate new quality.
  If (old quality > new quality) accept switch.
  Else reject switch.
End while
```

Figure 38 Cluster switcher

Select stage:

The select stage selects two clusters from the clusters collection. These clusters cannot be siblings and they cannot be ancestors of each other. This is because the clusters are to switch parents. So if they are siblings no change will be made and if they are ancestors one of them will be set to be a child of one of its own descendants.

Store old values:

During this stage old mean and quality values will be stored to temporary variables. This has to be done since the values will have to be used when testing if a step should be accepted or not.

Switch clusters:

c_0 and c_1 will exchange parents. So that p_1 is the parent of c_0 and p_0 is the parent of c_1 . Both child and parent relations have to be updated.

Calculate new value:

During this step new quality values for the two parent clusters are calculated. By only updating the parent's quality and not all clusters which change higher up in the tree we focus the algorithm on the local search method. All the affected clusters should only be updated when accepting a move. The quality can either be SSR or mean-quality. SSR means that all the clusters involved needs to have fully updated list of points which they will iterate through to calculate the new SSR for the cluster.

The mean quality is an alternative method of defining quality used to simplify the process of recalculating quality. Here all which needs to be updated are the size of the cluster and the mean of the cluster. The new mean will be calculated based on previous mean and cluster size. The quality itself is the distance between a clusters two children's means. This way we save a lot of time updating and reading points lists.

Accept switch:

When accepting a switch, nodes from c_0 and c_1 up to the first common ancestor have to be updated. The nodes higher up in the tree will still have the same clusters as descendants and will therefore remain unchanged. Keeping the changes only to the nodes up to the first common ancestor is important because it makes the algorithm focused on local search.

Reject switch:

When rejecting a switch all that needs to be done is to restore the size, mean and quality value stored in the store old value step.

The Figure below shows an example of a dendrogram switch.

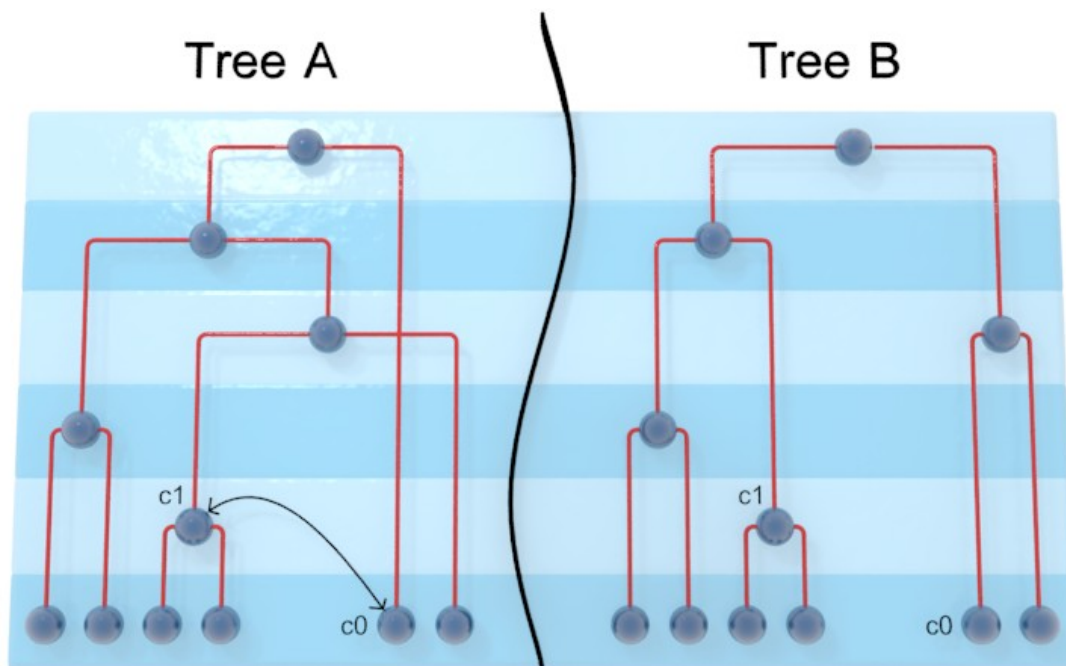


Figure 39 Example of a dendrogram switch

6.3.2 Cluster mover

In this algorithm one random cluster will be moved to a new parent. If there is an acceptable quality gain for the parents (p_0 and p_1) after the move, it is accepted. The way this is done is to select two clusters, then attach one of them to the parent node of the other. This requires restructuring of the tree around both nodes. We will call the two selected clusters c_0 and c_1 . In this explanation we will also refer to the parent of c_0 and c_1 as p_0 and p_1 , the grandparent of c_0 as pop_0 (parent of $parent_0$) and finally the sibling of c_0 as s_0 .

Cluster mover pseudo code:

```
while (not finished)
  select two random clusters
  store old values
  move cluster
  calculate new quality for  $p_0$  and  $p_1$ 
  if(quality has improved) accept move
  else reject move
end while
```

Figure 40 Cluster mover

Select clusters step

There are certain rules that have to be obeyed when selecting the two clusters.

- c_1 should not be an ancestor of c_0 ; $parent_0$ would become a child of one of its own descendants
- c_0 should not be the top node; a special case of the rule above.
- c_1 should not be parent of c_0 ; will end up with the same structure as before
- c_0 and c_1 should not be siblings; will end up with the same structure as before

There is one more step not involved in the pseudo code. This step examines the lineage of the selected clusters. This is important because what and how to perform updates depends on the relations of c_0 and c_1 .

Store values step

Stores old mean quality and relations for p_0 and p_1 . If c_1 is the top node there will be no p_1 , instead pop_0 will be used.

Move cluster

Here the actual restructuring of the tree gets done. What we want to do is actually to create a new node, attach c_0 and c_1 as children of that node and attach that new node in the old position of c_1 . But to simplify the process what we really do is to use p_0 as this new node and attach it as a child to p_1 . This eliminates the need for a method of creating and destroying nodes and clusters. The steps performed to fix all the relations are:

- Remove c_1 as a child of p_1 and add it as a child of p_0
- Remove p_0 as a child of pop_0 and add it as a child of p_1
- Remove s_0 as a child of p_0 and add as a child of pop_0

In cases where p_0 is the top node s_0 will be set to be the new top node. If c_1 is the top node p_0 will be set as the new top node. c_0 can never be the top node since it cannot be allowed that c_0 are higher than c_1 in the tree IF they are ancestors because this will create a situation in the tree where a parent is descendant of one of its own descendants which of course should not be allowed.

The figures bellow shows two example moves. Figure 41 shows an example when the two clusters are not ancestors of each other, while Figure 42 shows an example when c_1 is an ancestor of c_0 . The left part of the figures shows the tree before the move and at the right the tree after the move is displayed.

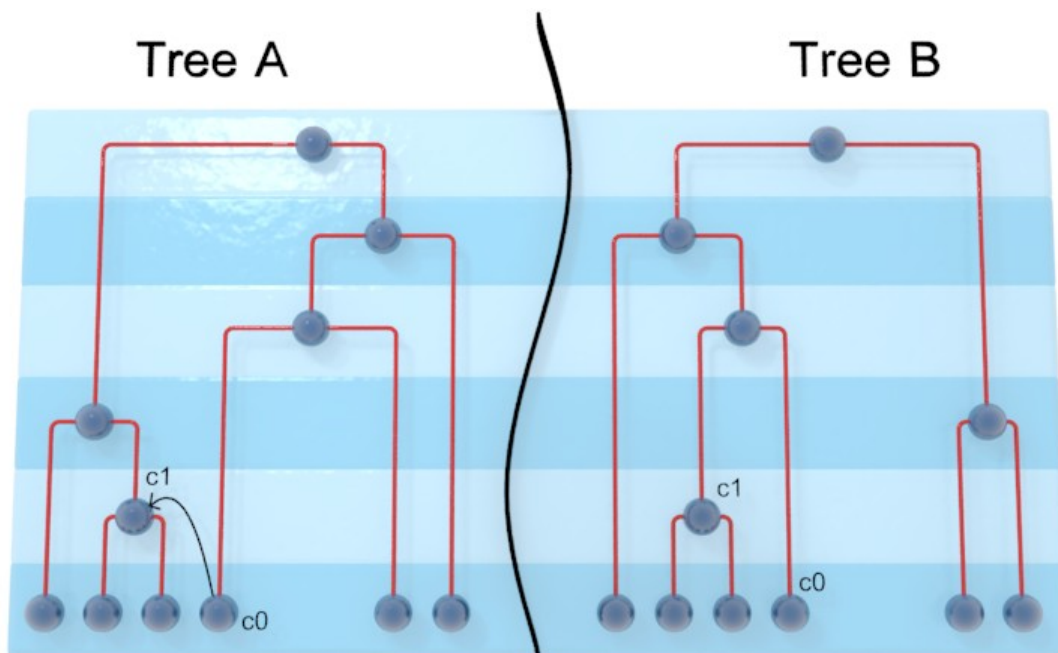


Figure 41 Move to different sub tree.

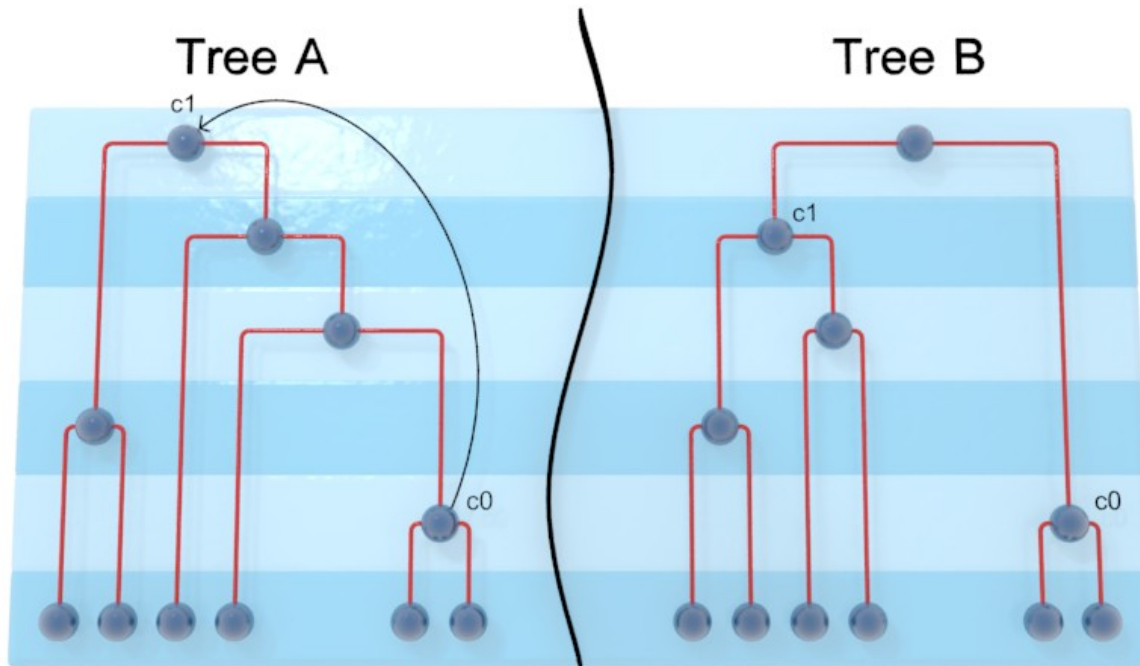


Figure 42 Move to same sub tree.

Check Quality Gain

After performing a move we want to check new quality values for p_0 and p_1 , but to do this also the size and mean of the clusters have to be updated. This is because the quality is directly connected to the mean values of the child nodes and because the mean is directly calculated based on old mean and size of a cluster.

To ensure local search usage we only want to update an as small as possible amount of clusters. Which clusters needs update are dependent on if the nodes are ancestors or not. If c_1 is an ancestor of c_0 all the clusters between p_0 and p_1 has to be updated. This is because the quality of any cluster is dependent on all of its descendants.

If the clusters are not related in this way however, all that needs to be updated are the values of the parent clusters themselves. The only thing to think of here is that p_0 needs to be updated before p_1 since p_0 , after the move step, is a child of p_1 .

In cases where c_1 is the old top node of the tree p_1 will not be defined, and the quality of pop_0 is used instead of the quality of p_1 .

After updating quality values for the necessary clusters the new quality value for p_0 and p_1 are compared to the new qualities.

What to accept as an improvement is one of the most important parts of the algorithm. With more time on our hands we would have liked to study this step even further to see if there are any clever ways of measuring improvements. The quality of a cluster can be set in our application. The slowest, but most accurate, way is using SSR. As discussed earlier SSR would require large lists of points for a lot of clusters to be updated. This would in cases of large trees take a lot of time to do. In an SSR-quality based environment we sum the old qualities and the new qualities. If total new quality is lower than total old quality the move is accepted.

The mean quality calculations are a measurement of the distance between the means of the children of a cluster. This method will not give us any direct information on how the clusters actually are, but the thought is that clusters where the children are located close to one another are more compact than a cluster where the children are further apart. We also had some discussions about weighting the qualities based on how many points are in a cluster. If

a cluster got one child with a lot of points, and one child with few points it seems natural that the one with many clusters should somehow count for more than the one with few points, but this may not always necessary be true for different reasons, so we decided to keep the quality without taking into account how many points each cluster has.

Even when deciding how to measure quality we had to decide on how to measure improvement when using the mean quality method. We tested some different ways of measuring this. One was to simply say that an acceptable step had improvements both in p_0 's and p_1 's quality. This seemed to work ok enough, especially in cases with no added noise, but we still wanted to test some other options. We experimented some with accepting cases where the quality declined for one of the parents as long as the other was improved, but in the end we ended up with accepting a move if the quality of p_0 is improved AND that the improvement in percentage is higher than the loss of quality in p_1 .

Accept move

If a move is accepted there are some more values which need to be updated. If the nodes are ancestors the necessary nodes have already been updated, but in the case where they are not ancestors however it is necessary to update all the nodes from both parent nodes to the first common ancestor node. This is because the first common ancestor is the last node affected by the move. So the first common ancestor node is identified and two updates, one from p_0 to first common ancestor and one from pop_0 to first common ancestor are performed.

Reject Move

The first step of the reject move is to old associations. Here the changes made during the move stage of the algorithm are reversed. The reason for rebuilding the tree structure before resetting mean, size and quality values are that the method used to update size takes size from both children, it is therefore important that the relations are correct when this is done. If the clusters are ancestors all that needs to be updated are the nodes from p_0 to p_1 or, if c_1 is the top cluster, from p_0 to c_1 .

If the clusters are not ancestors values for p_0 and p_1 , or alternatively pop_0 if c_1 is the top node, will be set back to the values stored in the store values step.

6.4 Experiments hierarchical clustering

6.4.1 Setup

The experiments

In our experiments part we tried the two different dendrogram clustering algorithms on several problems. We created four experiments of varying size without noise and four similar problems of the same sizes with noise. In each of these experiments there were four groups of points. The algorithms should then at one level in the tree find the four clusters. One example of such an experiment can be seen in Figure 43. The sizes of these experiments were as follows:

- 20 points
- 100 points
- 1000 points
- 4000 points

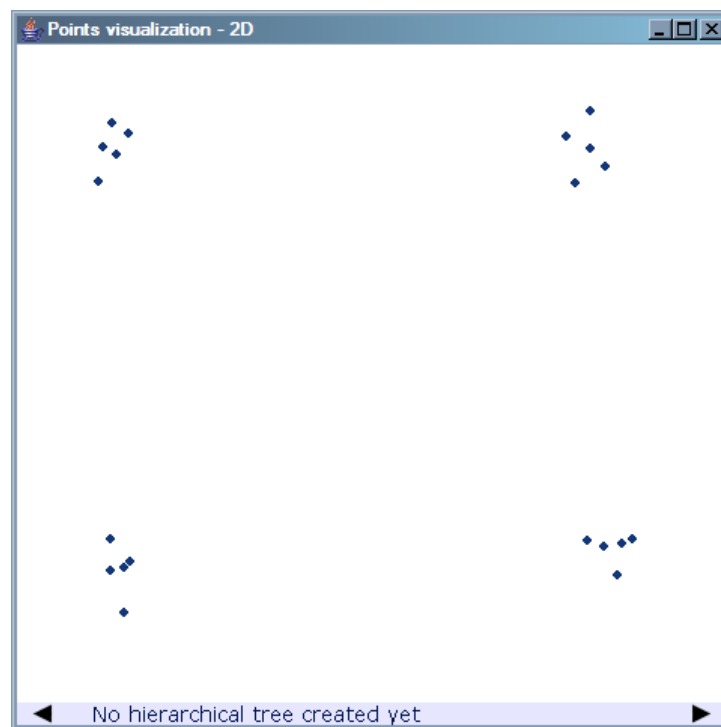


Figure 43 Example of experiment with 4 clusters

In addition we created a more complex experiment; at one level there should be found four clusters, but these clusters were divided into three smaller clusters. The hierarchical tree should then include both solutions, where the four and the twelve clusters are recognized. Finding all the natural clusters at some level is an important quality of the algorithm. The size of this experiment was 400 points, and the points in this experiment are presented in Figure 44.

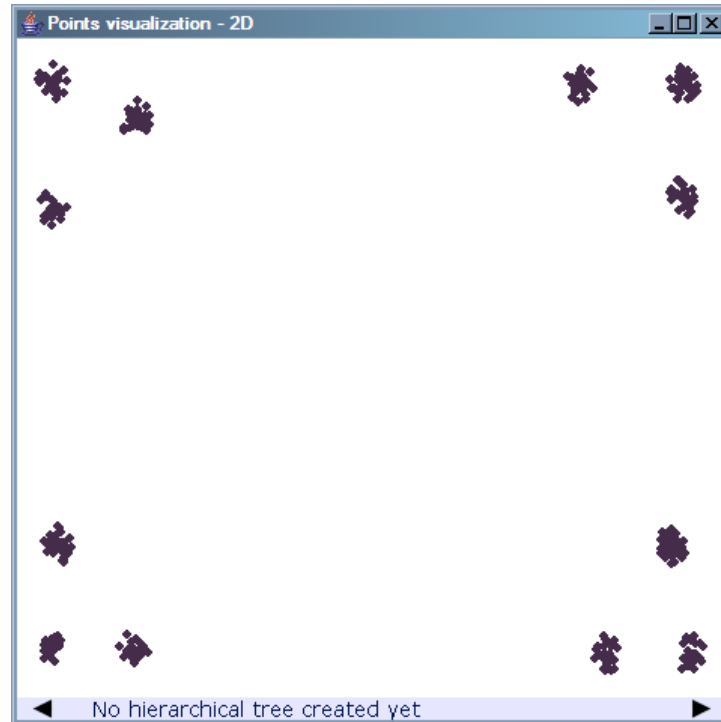


Figure 44 Example of experiment with 12 clusters

Each point was placed in two-dimensional space and was generated by placing the points in several more or less obvious clusters. This distribution of points has the potential to get a high reduction off SSR value, if the groups are recognized by the algorithms. When the optimal solution is found, the SSR value will usually have been greatly reduced, as the points are distributed in more or less compact groups.

The goal of the experiments

These experiments were chosen to see how the algorithms worked in cases with different complexity. We then evaluated the quality of the clustering. The method used for calculating the quality in these experiments was SSR. This gives a good measure of how dense the clusters are. In this context, the quality of the clustering is based on a combination of speed and precision; the algorithms should reduce the SSR value as fast as possible. We tried to cover the cases from having few points and clusters to many points and clusters, and some in between.

In each graph we compared the results to the closest agglomerative algorithm. This algorithm represents the state-of-the-art solution, so we wanted to see how our algorithms did compared to this, both regarding the quality of the clustering and the time spent.

Settings

For each algorithm we made 100 test runs on the same problem and calculated the average time and reduce in SSR. The algorithm was set up to take 50 measurements recording the time spent and the current quality of the solution. The number of runs was adjusted to fit each of the experiments. All GUI updates were turned off to make the calculations as fast as possible.

The experiments was done on an Intel Pentium 4 2.26 GHz with 512 MB ram, running on Windows XP Home. We also did the experiments on an Intel Xeon dual core 3.0 GHz with 8 GB RAM, running on Linux Debian. This machine allowed us to run two algorithms in separate threads at once. But as we logged on to this computer over the network and didn't have any physical control over it, we could not confirm that it was not working with some

other tasks. The times we got with this machine on the same experiments varied, so we decided to do the experiments on one of our local machines. There might also have been some issues with running two threads at the same time, we are not sure if this would be the same as running two threads on two different processors and any problems here might explain some odd results we got from the tests on this computer.

To give the possibility to later test these results, all the initial problems are saved to XML formats which are attached as an appendix to this report (CD-ROM).

6.5 Empirical results

In this section we have presented the results of our experiments.

To show the hierarchical tree we have used a variant of the technique called Multidimensional Scaling (MDS). This technique allows us to try and show the distances between the points in the bottom of the tree by projecting the distance of the points in two dimensions to a distance in one dimension. It is then easier to see if the clustering of the hierarchical tree makes sense. This is a good way to represent the points, but remember that it will rarely (if at all) be possible to get this to be totally accurate. Sometimes close clusters will be shown as being quite far apart from each other. In these cases one can check the distance between the points by looking at the 2 dimensional point windows.

6.5.1 Results from 4 clusters 20 points without noise

Settings

Algorithm iterations: 4000

Number of times the experiment is solved: 100

Comments

In this small experiment it is quite easy to see from the hierarchical tree in Figure 46 that a good structure is found. The left part shows the initial, randomized tree and at the right, an example of a clustered tree is shown. From the top, the points are first divided in two clusters, and then each of these is divided in two new clusters. These four clusters represent the natural clusters, which are shown in Figure 46.

From the graph (Figure 47), we see that “cluster mover” algorithm is getting really close to the quality of the closest agglomerative clustering, at just above 10% of the initial quality. The “cluster switcher” settles for a quality of about 19% of the initial value, a bit behind the “cluster mover”, and requires some more time. At this small example the closest agglomerative algorithm finds the solution before either of the other clustering algorithms gets any satisfying results.

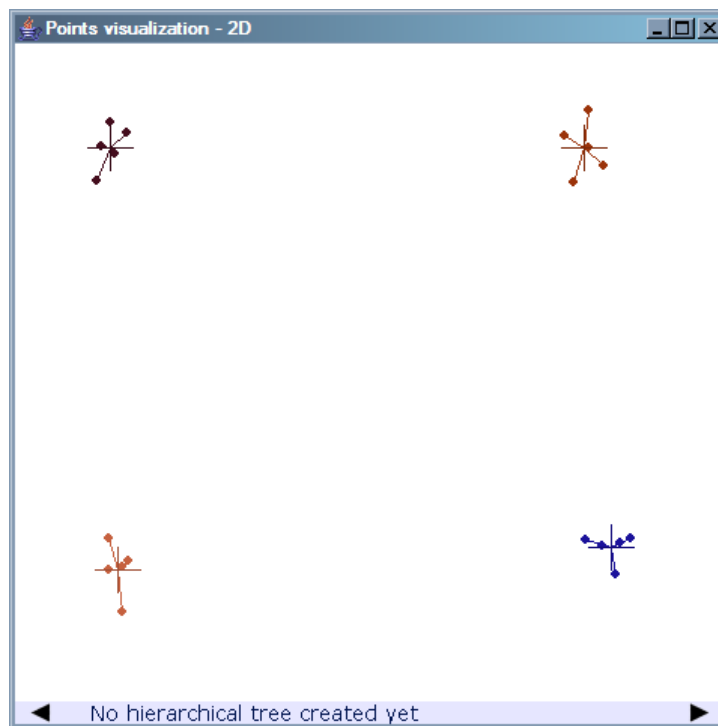


Figure 45 20 points, without noise - the natural clusters

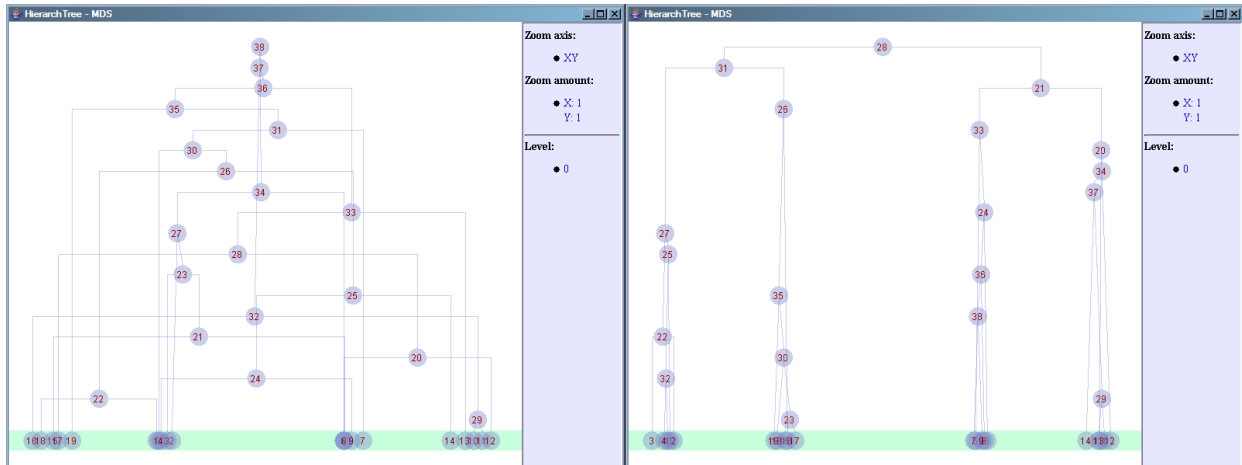


Figure 46 20 points, without noise - hierarchical tree before and after clustering

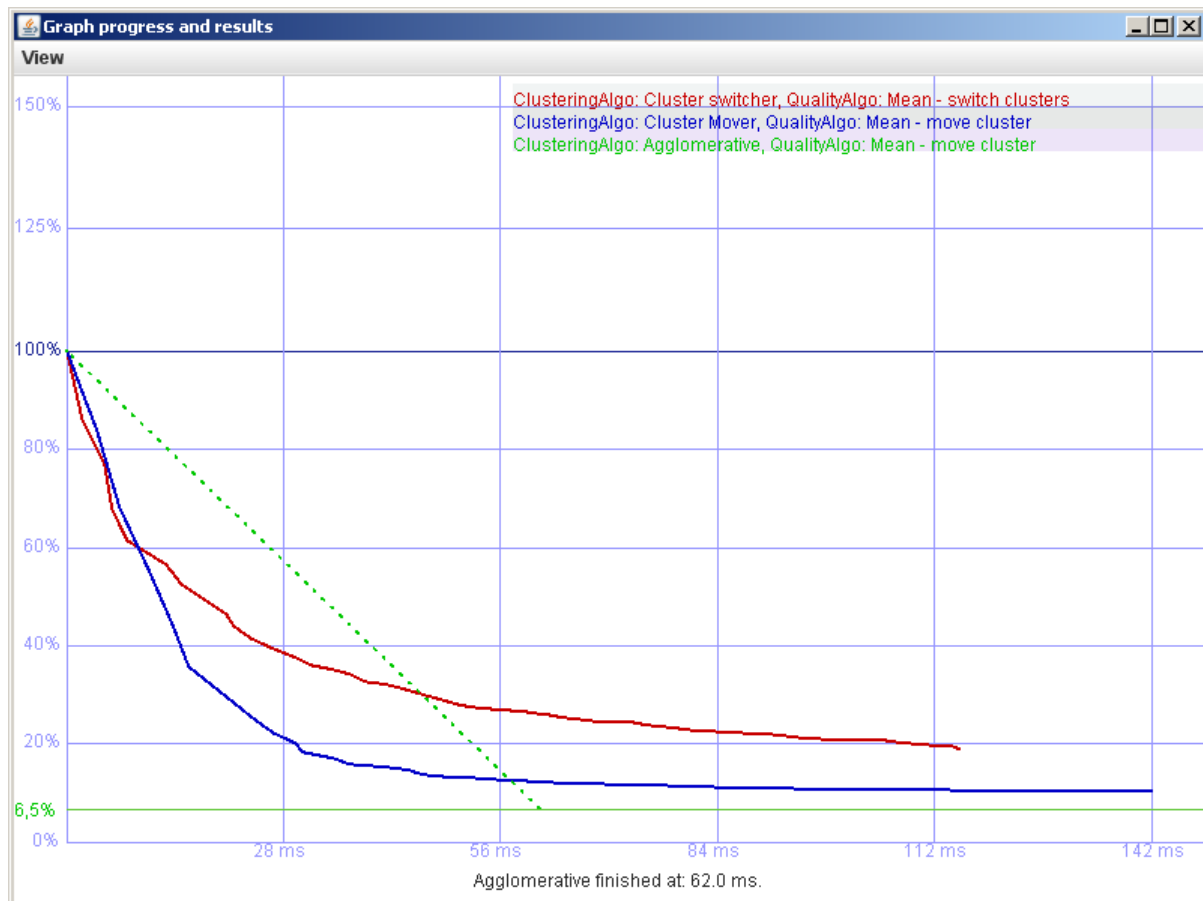


Figure 47 20 points, without noise - graph showing the clustering progress

6.5.2 Results from 4 clusters 20 points with noise

Settings

Algorithm iterations: 8000

Number of times the experiment is solved: 100

Comments

Another small experiment, this time with noise (some points are more spread around the natural clusters). From the hierarchical tree in Figure 49, the result is a bit messier than it was without the noise. The four natural clusters including the noise is still found (as seen in Figure 48), but the internal structure of these four sub-trees are not entirely perfect.

From the graph (Figure 50), we see that “cluster mover” still comes close to the closest agglomerative clustering. Again, the “cluster switcher” is somewhat behind “cluster mover”, both regarding clustering speed and the final quality. Not unexpectedly, the closest agglomerative algorithm again finds the solution before either of the other clustering algorithms gets any satisfying results.

The reason the closest agglomerative algorithm seems to be taking 0ms to complete is that the computer seems to have some problems handling extremely short time intervals when using the timer function in Java. It will, of course use some milliseconds to complete but this time is so minute that it is virtually zero (at least from a human perspective).

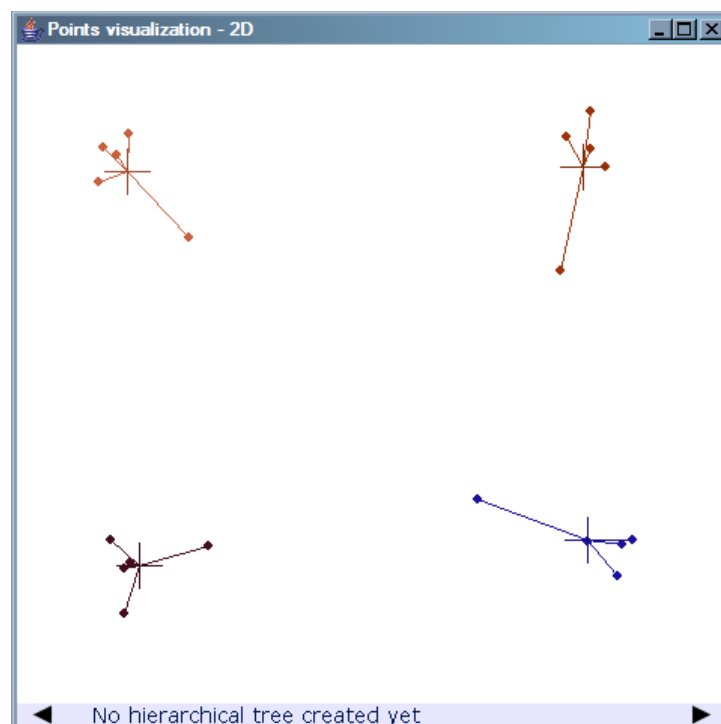


Figure 48 20 points, with noise - the natural clusters

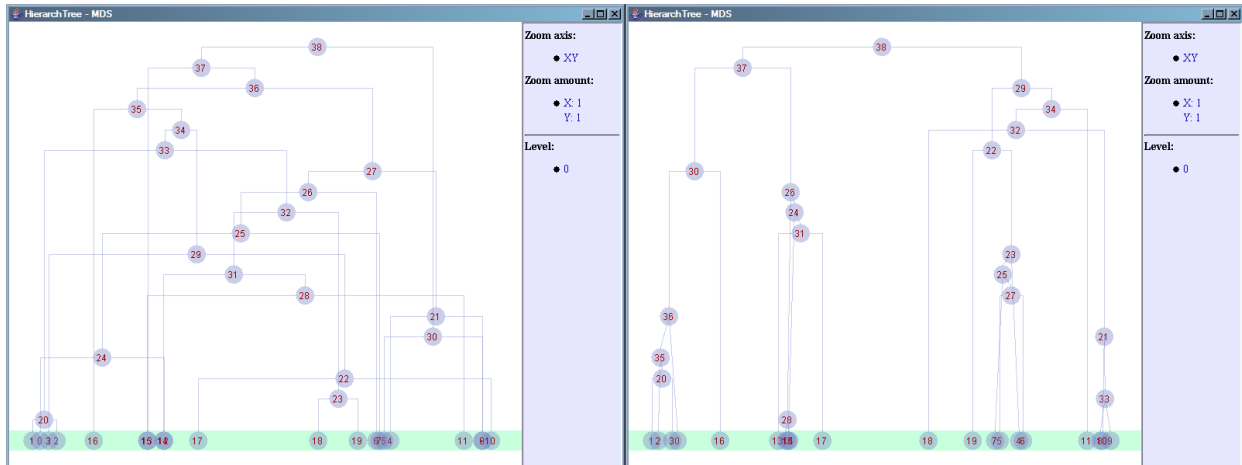


Figure 49 20 points, with noise - hierarchical tree before and after clustering

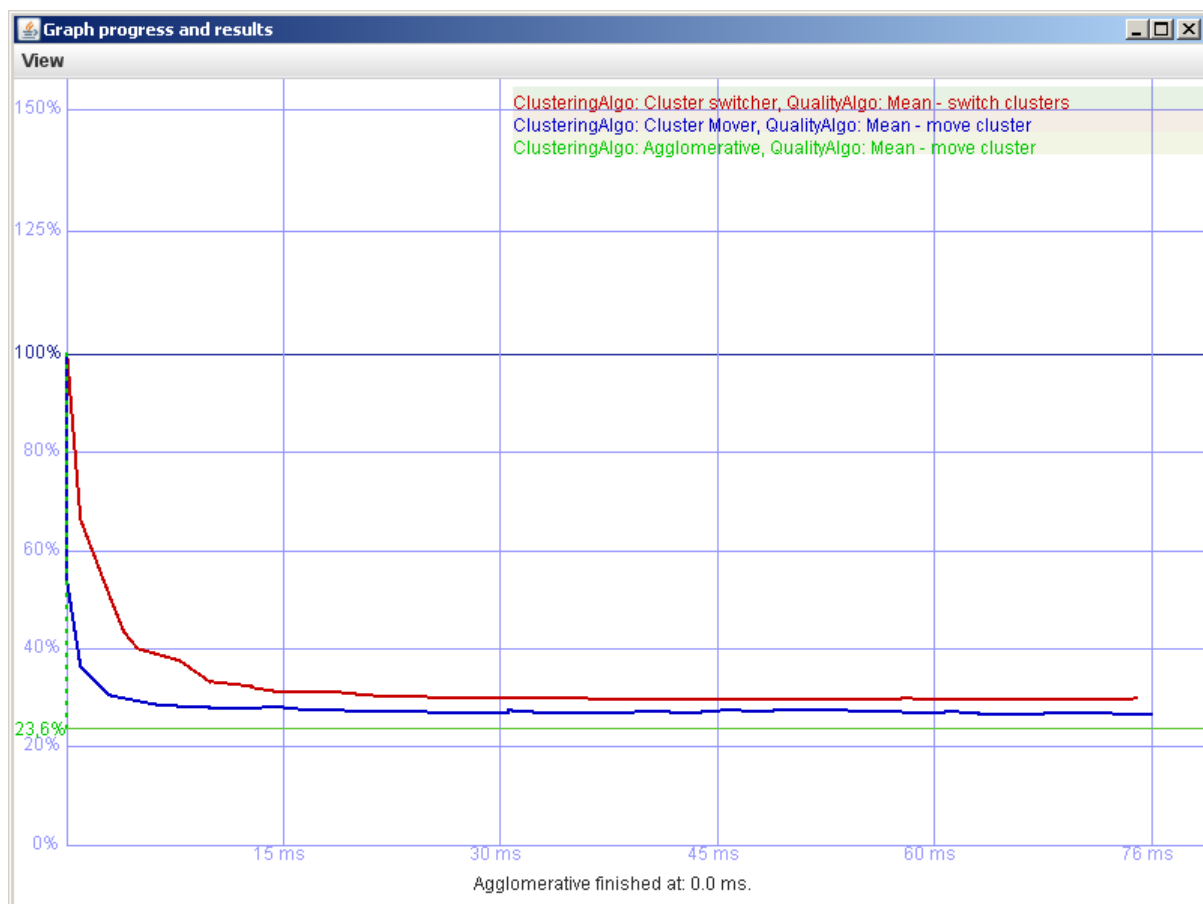


Figure 50 20 points, with noise - graph showing the clustering progress

6.5.3 Results from 4 clusters 100 points without noise

Settings

Algorithm iterations: 10000

Number of times the experiment is solved: 100

Comments

From the hierarchical tree in Figure 52, we see that the natural clusters are recognized. As the tree is getting more complicated, it gets harder to manually evaluate the hierarchical tree. But it is a good sign when crossing lines from one group of points to another doesn't occur until the top of the tree.

From the graph (Figure 53), "cluster mover" is now faster than the closest agglomerative clustering down to about 7% of the initial quality. After this point "cluster mover" generally requires more time in order to do minor improvements to the solution. The "cluster switcher" is showing to be more ineffective compared to "cluster mover".

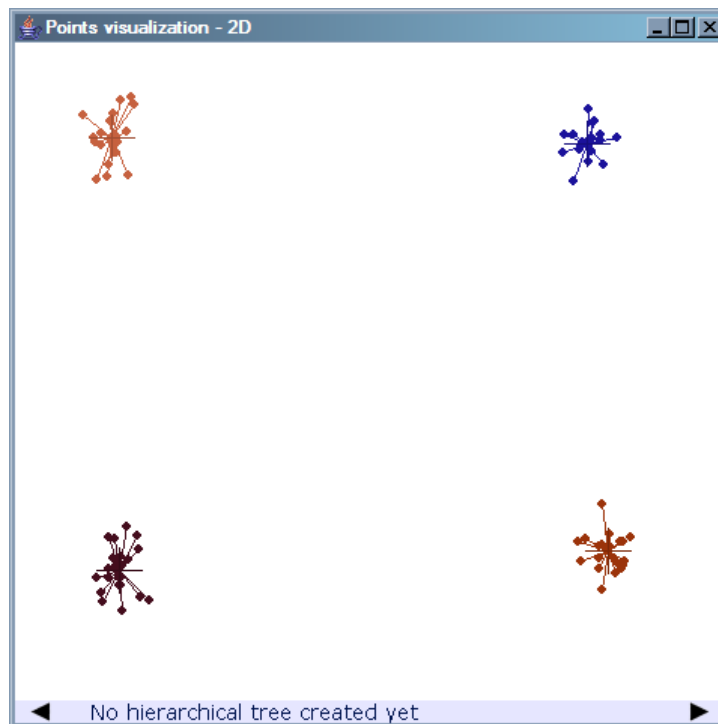


Figure 51 100 points, without noise - the natural clusters

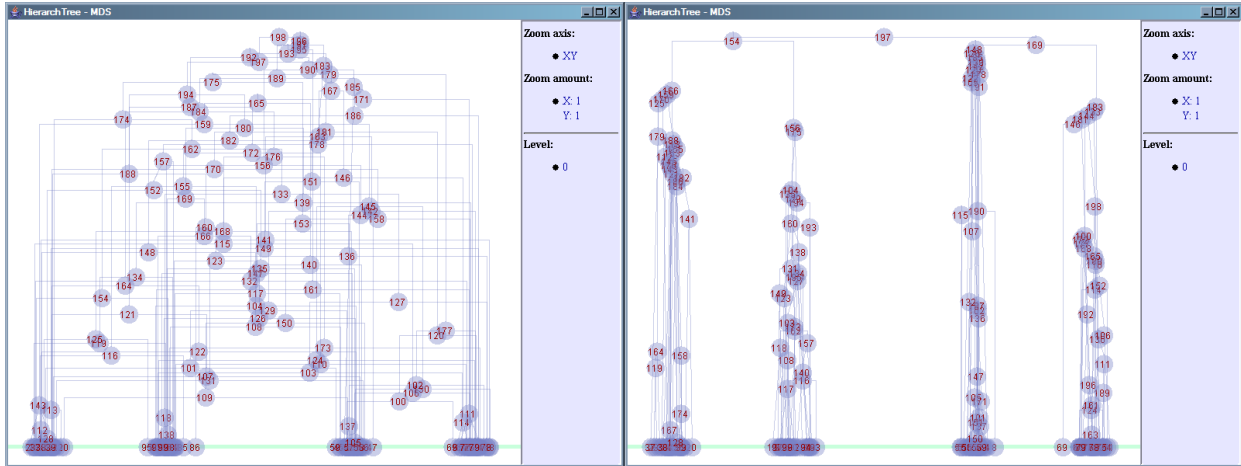


Figure 52 100 points, without noise - hierarchical tree before and after clustering

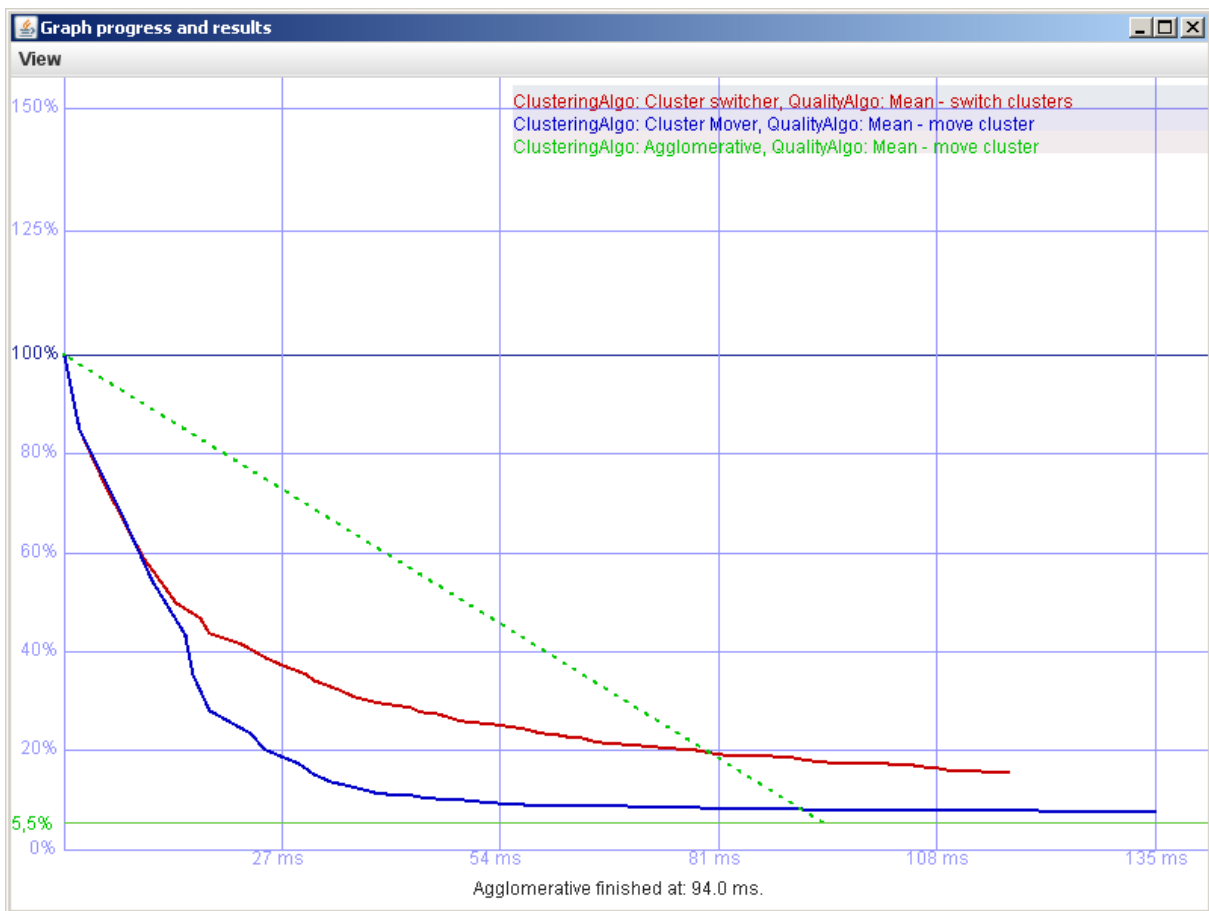


Figure 53 100 points, without noise - Graph showing the clustering progress

6.5.4 Results from 4 clusters 100 points with noise

Settings

Algorithm iterations: 16000

Number of times the experiment is solved: 100

Comments

From the hierarchical tree structure in Figure 55 we clearly see the structure of the natural cluster. Because it is noise in this experiment the tree structure is a bit messier than the structure without noise (Figure 53Figure 55). But again the algorithms are able to find the natural clusters as seen in Figure 54.

From the graph (Figure 56) “cluster mover” is again faster than the closest agglomerative clustering down to about 12% of the initial quality. After this point “cluster mover” generally requires more time in order to do minor improvements to the solution. The “cluster switcher” is showing to be more ineffective compared to “cluster mover”.

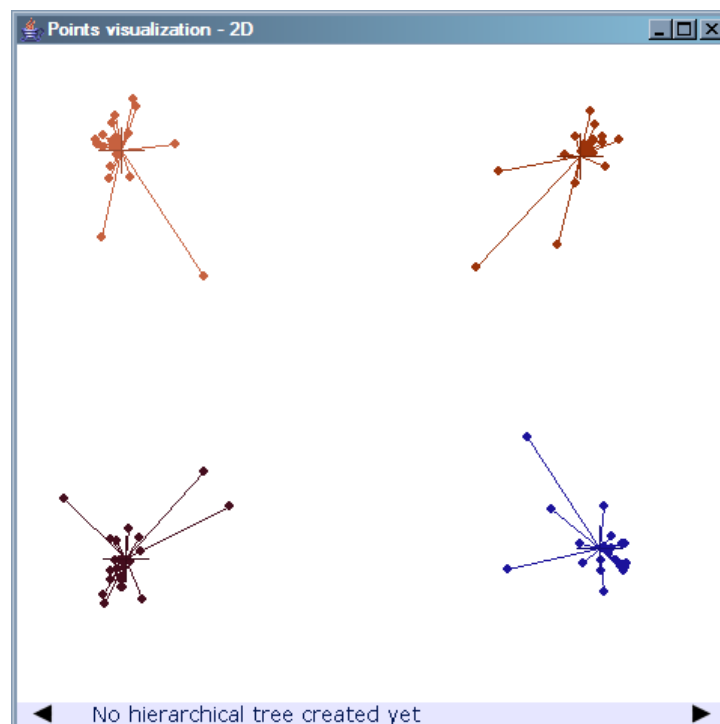


Figure 54 100 points, with noise - the natural clusters

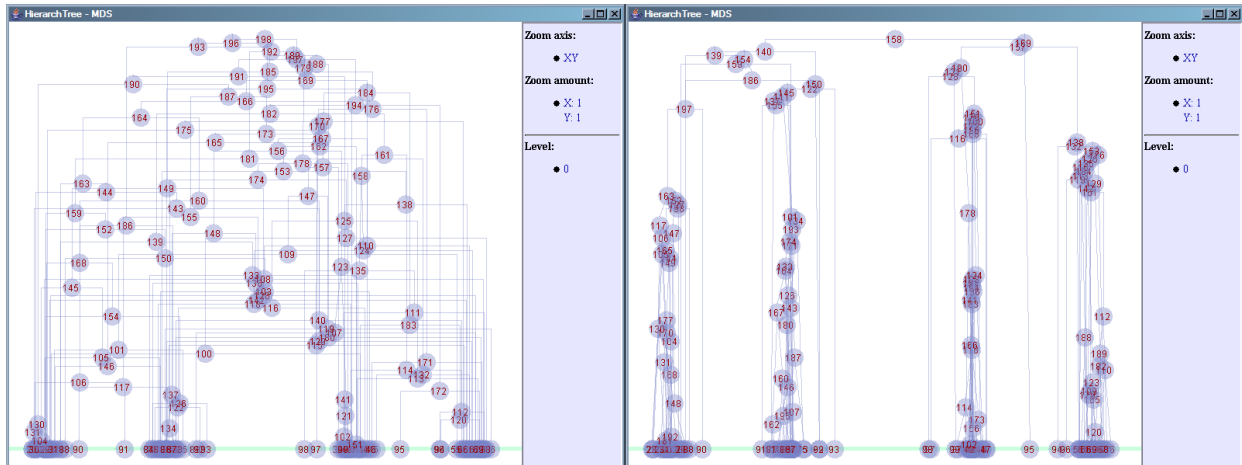


Figure 55 100 points, with noise - hierarchical tree before and after clustering

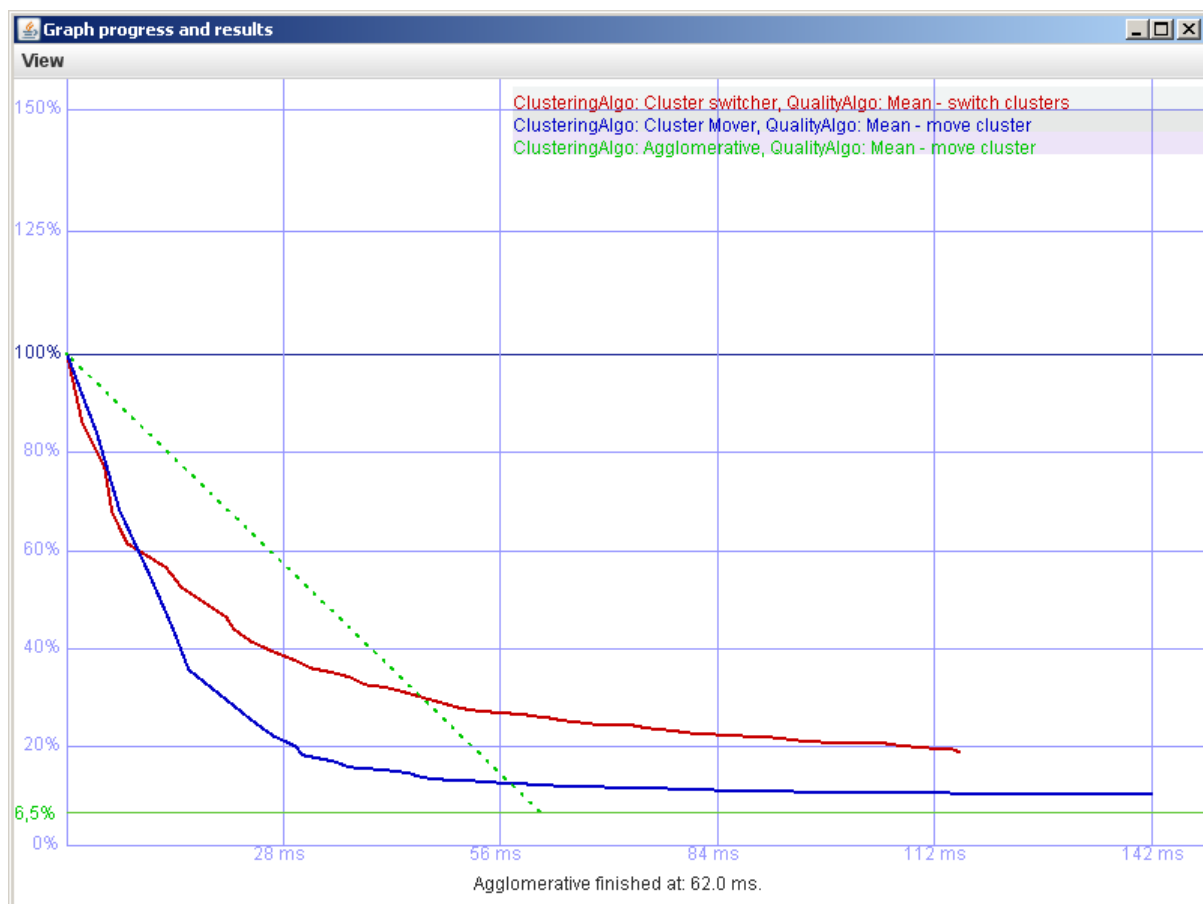


Figure 56 100 points, with noise - Graph showing the clustering progress

6.5.5 Results from 4 clusters 1000 points without noise

Settings

Algorithm iterations: 500000

Number of times the experiment is solved: 50

Comments

In this experiment we have increased the number of clusters noticeably. From the hierarchical tree structure in Figure 58 we see that the tree at the left seems to be rather chaotic. The result on the right clearly shows the structure of the four natural clusters. Another view that shows the natural clusters can be seen in Figure 57.

From the graph (Figure 59), we see that the closest agglomerative algorithm requires a lot more time than our algorithms. Our “cluster mover” algorithm spent around 3 seconds on the same problem while the closest agglomerative spends 64 seconds.

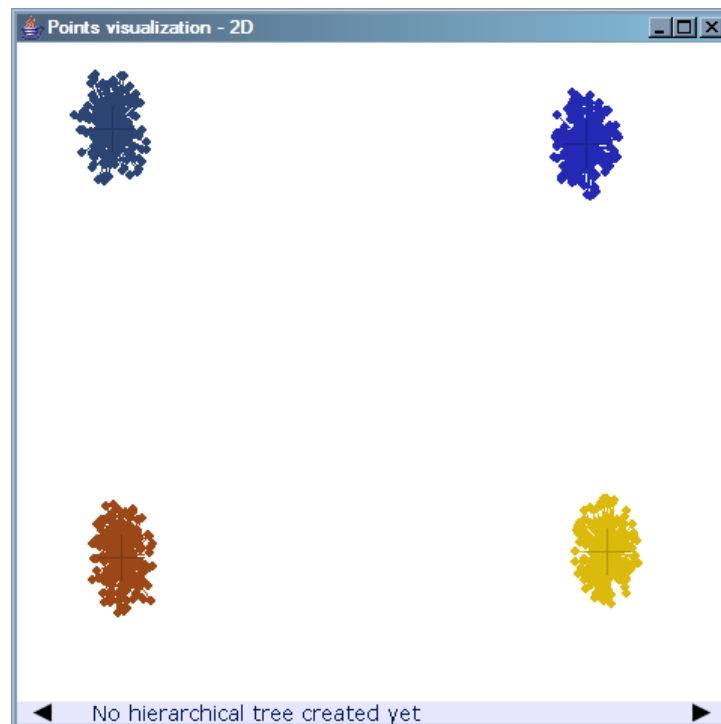


Figure 57 1000 points, without noise - the natural clusters

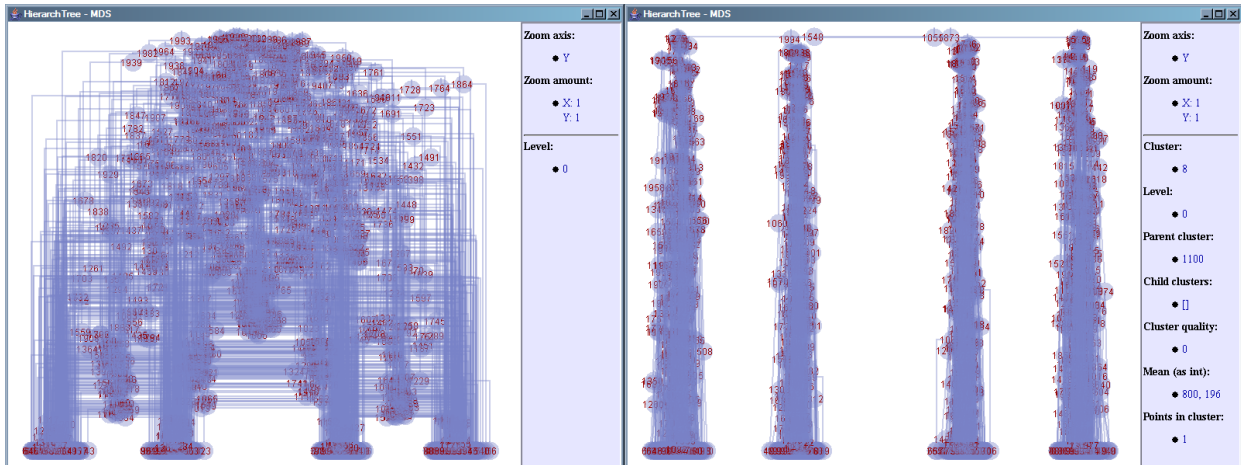


Figure 58 1000 points, without noise - hierarchical tree before and after clustering

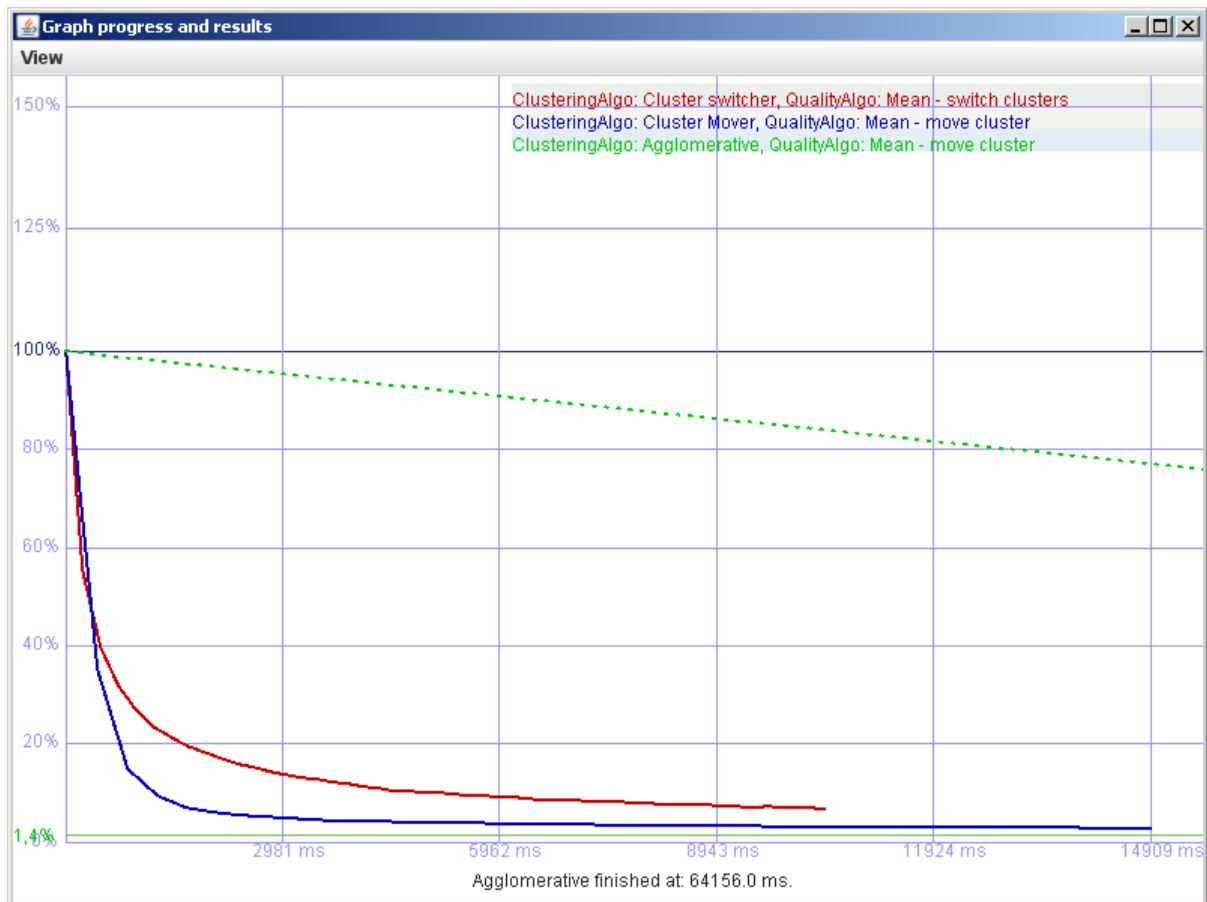


Figure 59 1000 points, without noise - Graph showing the clustering progress

6.5.6 Results from 4 clusters 1000 points with noise

Settings

Algorithm iterations: 500000

Number of times the experiment is solved: 50

Comments

When making some of the points in the previous experiment noisier, by scattering them around the clusters, the natural clusters are still recognized. As expected, the hierarchical tree structure before the clustering (at the left in Figure 61), is random and untidy, but the clustering sorts that out and finds the natural clusters. The result can be seen in Figure 60.

The graph (Figure 62) shows once more that “cluster mover” is considerably better than the “cluster switcher”, regarding speed and quality. “Cluster mover” seems to converge at about 3% of the initial value, while closest agglomerative clustering makes it down to 1,7%. This tells us that our algorithms still has problems with the internal structures in the clusters, even though it still finds the natural clusters. In this case our “cluster mover” algorithm spends around 5 seconds on the same problem while the closest agglomerative still spends 64 seconds. The closest agglomerative has to do the same number of iterations no matter how the points are placed.

The difference between the “cluster switcher” and “cluster mover” is about the same in this experiment as in the previous.

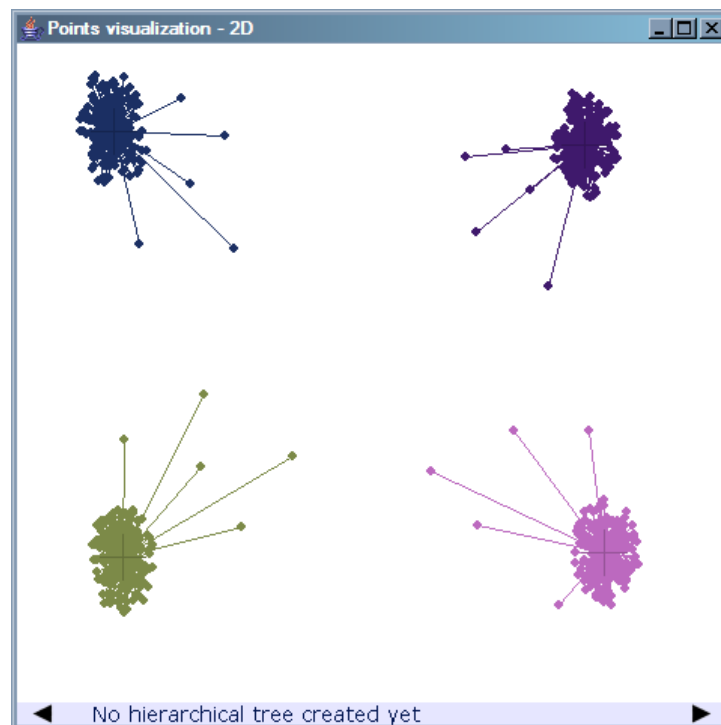


Figure 60 1000 points, with noise - the natural clusters

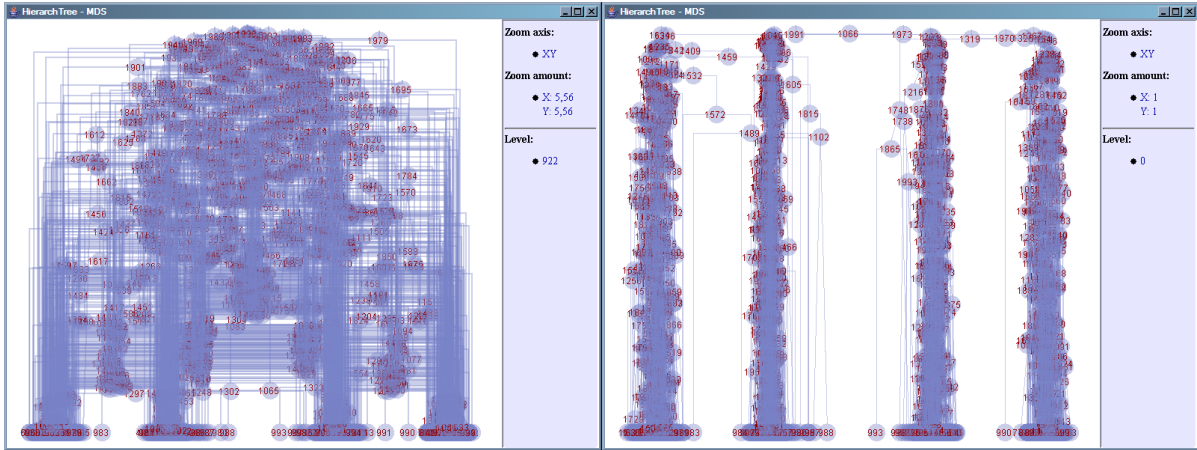


Figure 61 1000 points, with noise - hierarchical tree before and after clustering

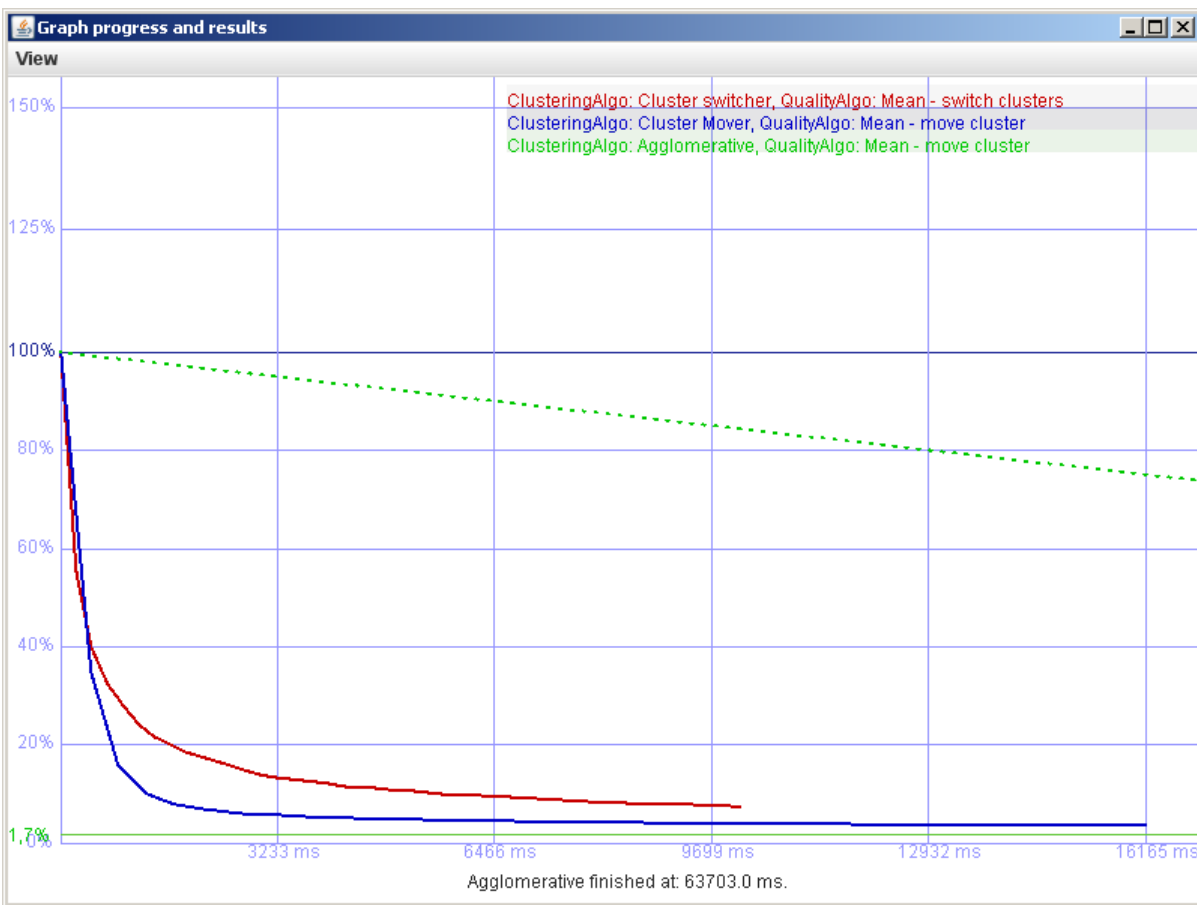


Figure 62 1000 points, with noise - Graph showing the clustering progress

6.5.7 Results from 4 clusters 4000 points without noise

Settings

Algorithm iterations: 2 500 000

Number of times the experiment is solved: 50

Comments

Increasing the number of points further, the hierarchical tree structure before the clustering (at the left in Figure 64), is becoming one big pile of points, but it gives an indication of the natural clusters being recognized.

The graph (Figure 65) now shows that “cluster switcher” in this case works almost equally well as the “cluster mover”. “Cluster mover” reaches 40% in about 238 seconds while the closest agglomerative algorithm needs 1.5 hours to reach 1,5%.

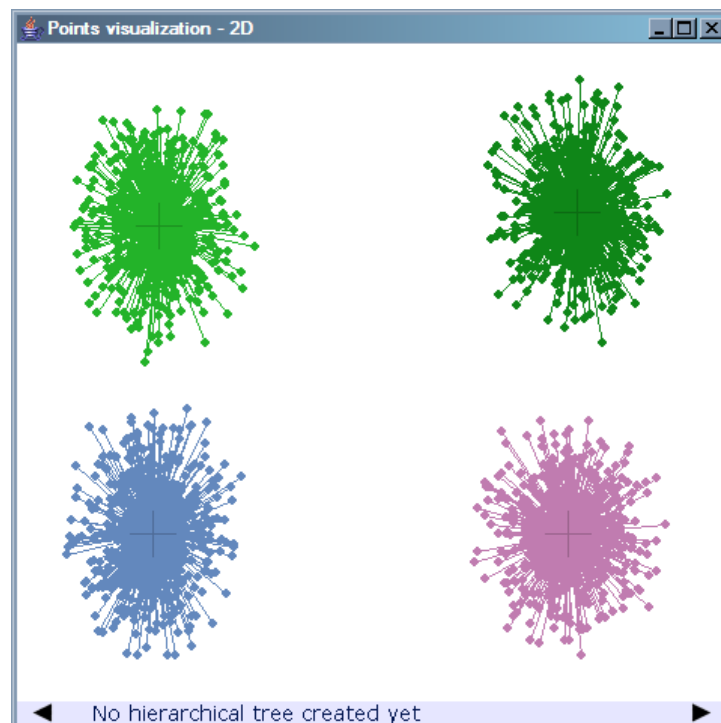


Figure 63 4000 points, without noise - the natural clusters

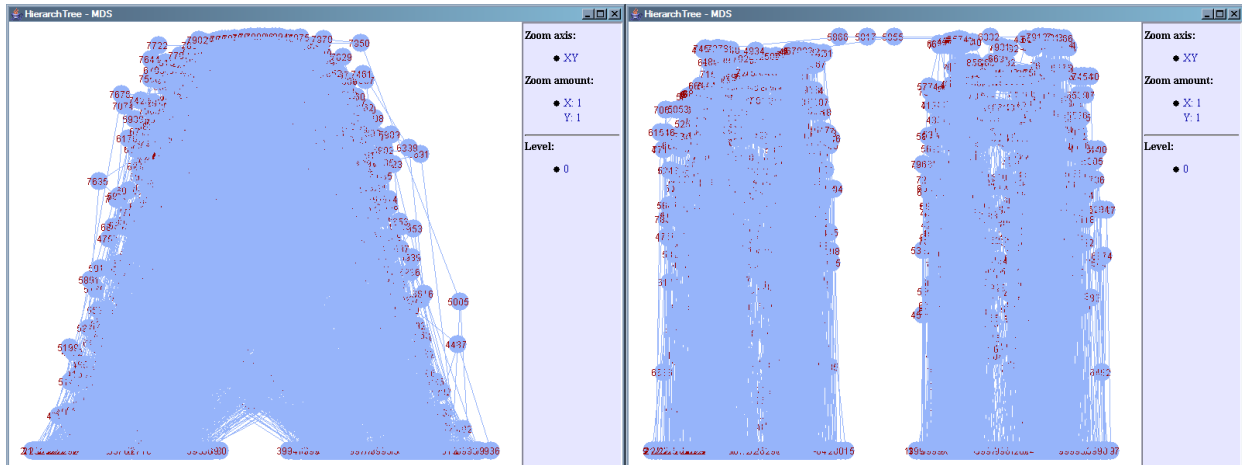


Figure 64 4000 points, without noise - hierarchical tree before and after clustering

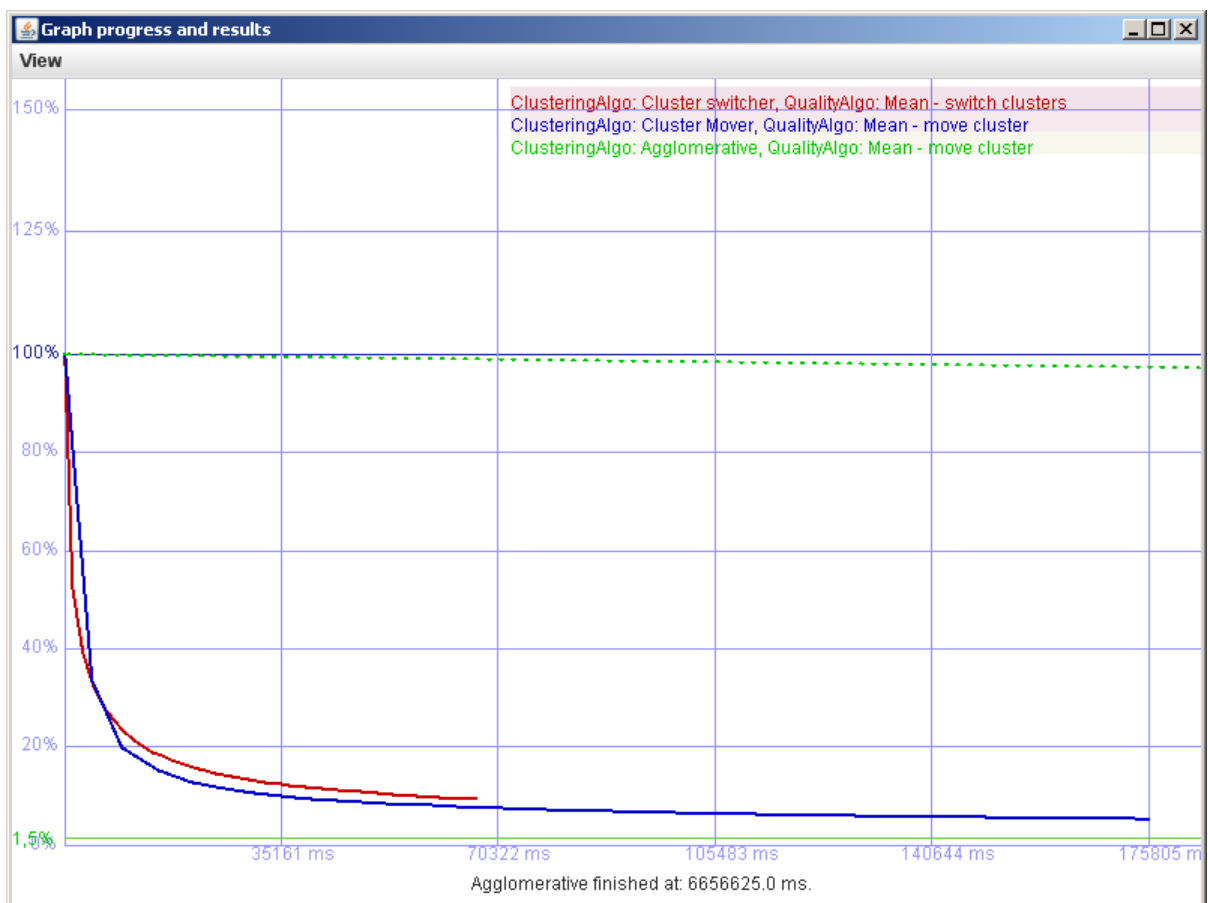


Figure 65 4000 points, without noise - Graph showing the clustering progress

6.5.8 Results from 4 clusters 4000 points with noise

Settings

Algorithm iterations: 2 500 000

Number of times the experiment is solved: 100

Comments

Once again it is hard to figure out all the details from the hierarchical tree structure (Figure 67), but the tree at the right clearly suggests that the natural clusters were recognized. Figure 66 shows that the natural clusters are correctly clustered, but there are some problems with the noisy points. Fixing this would need more iteration in the algorithm, but it could take a lot of time just to do minor improvements.

The graph (Figure 68) shows the same tendencies as the previous experiment, with “cluster mover” being a bit better than “cluster switcher”. The average of our algorithm is still higher than the closest agglomerative.

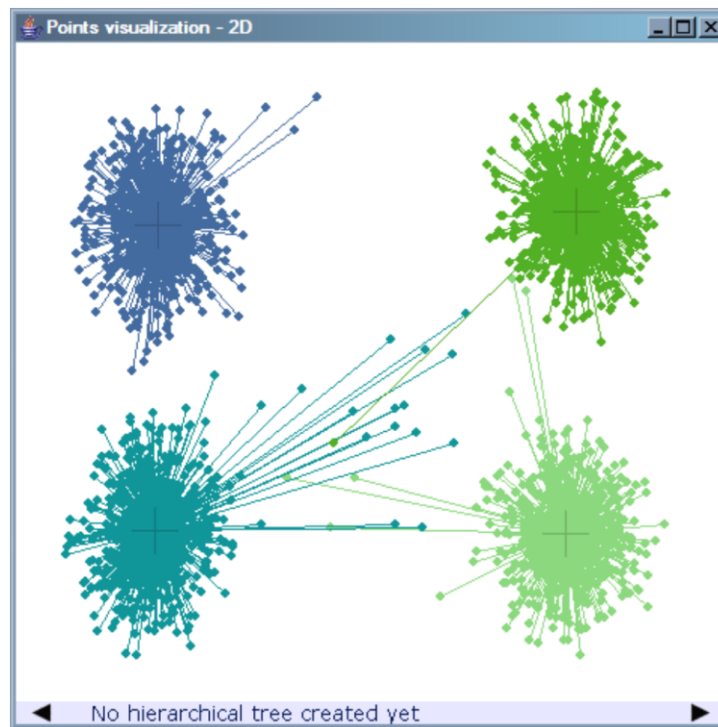


Figure 66 4000 points, with noise - the natural clusters

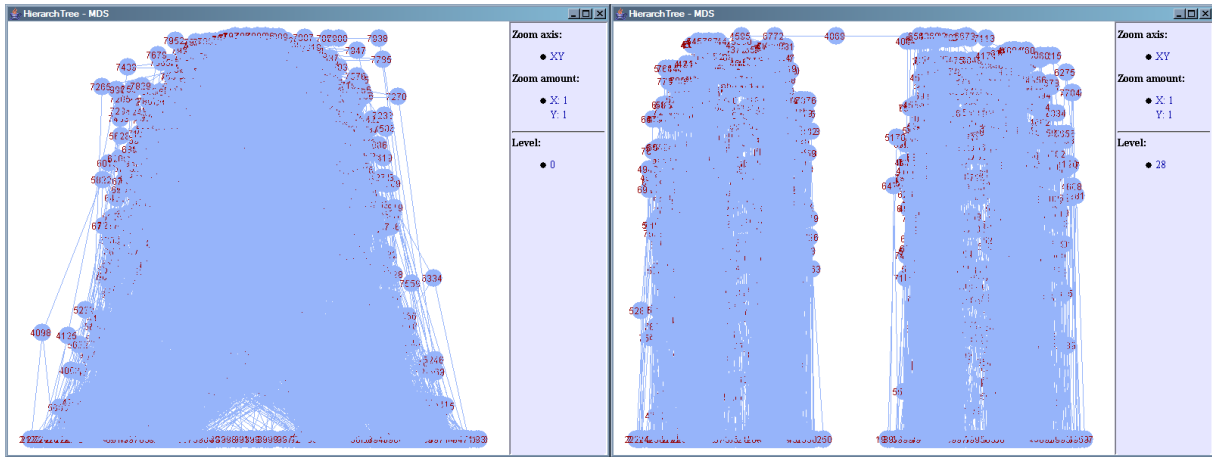


Figure 67 4000 points, with noise - hierarchical tree before and after clustering

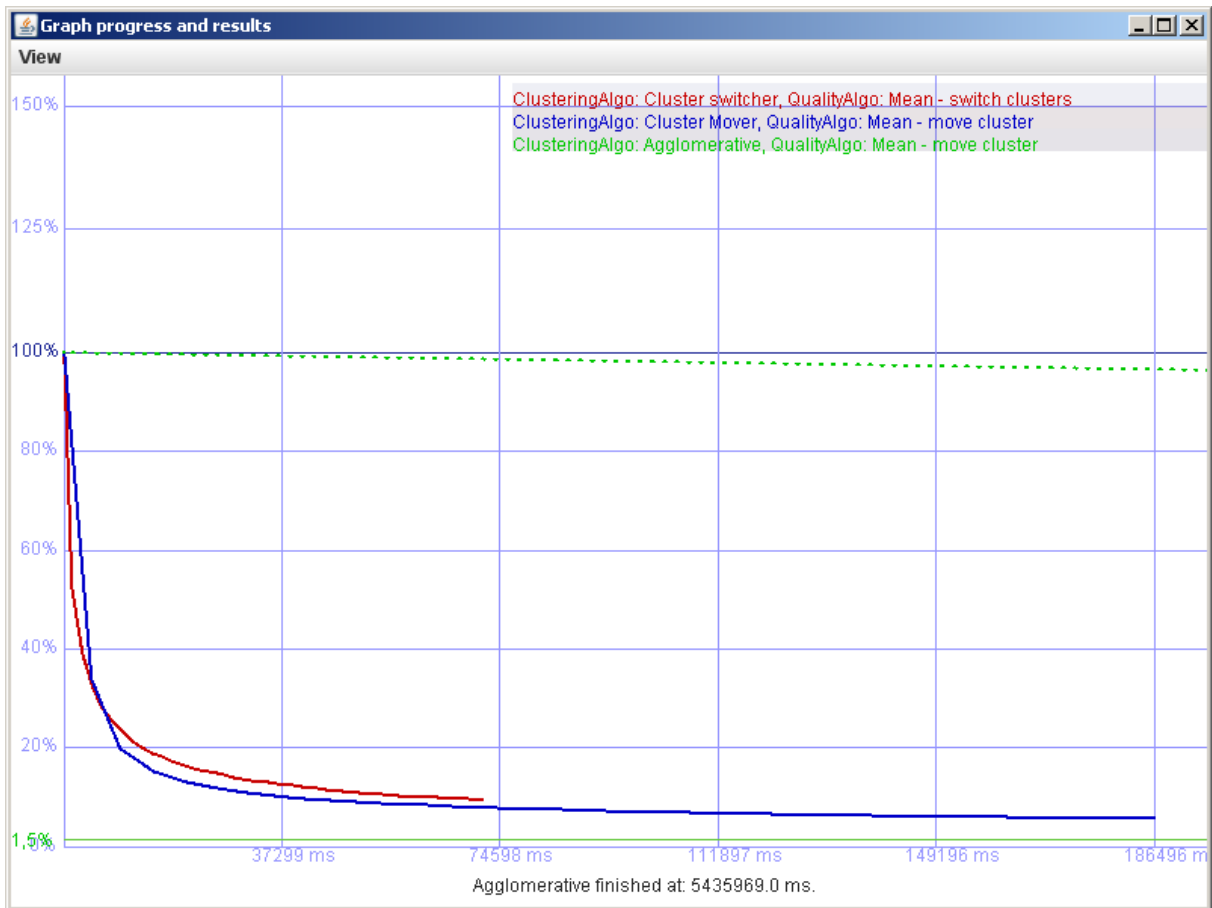


Figure 68 4000 points, with noise - Graph showing the clustering progress

6.5.9 Results 400 points, different levels

Settings

Algorithm iterations: 500 000

Number of times the experiment is solved: 100

Comments

This was a different experiment we made in order to see how the algorithm would deal with sub clusters. We created four clusters, where each of them could be divided in three new clusters. The goal was then to have both these solutions represented in the hierarchical tree. Figure 70 shows that the hierarchical tree was much tidier, although it is hard to get too much information from it. In Figure 69 we have found the two solutions we were after; at the left the four main clusters was recognized and at the right all twelve was found.

Figure 71 compares cluster mover, cluster switcher and the closest agglomerative algorithms. Cluster mover gets really close to the quality of the agglomerative clustering, but at a shorter amount of time. And the more points the clusters would have, the bigger this time difference would be.



Figure 69 400 points, different levels - the natural clusters

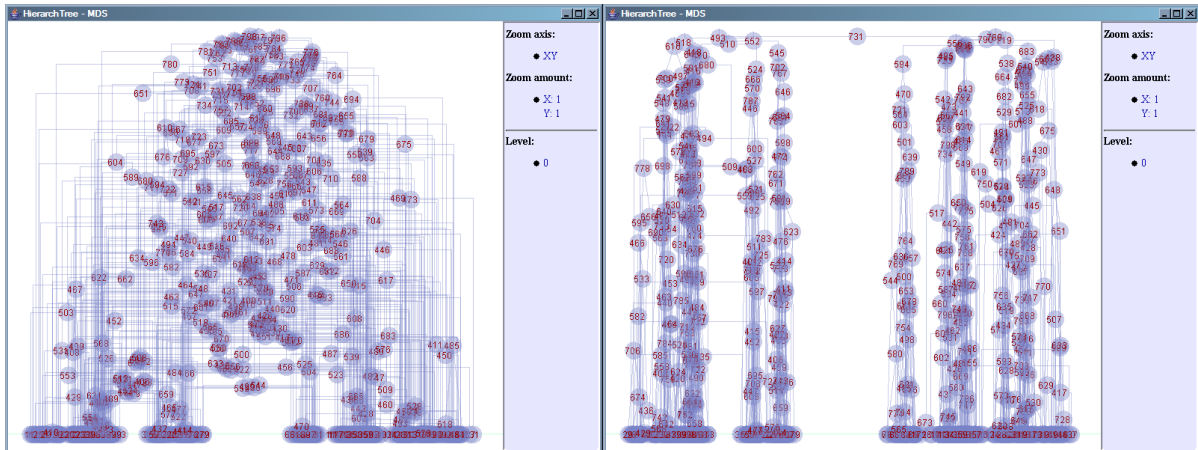


Figure 70 400 points, different levels - hierarchical tree before and after clustering

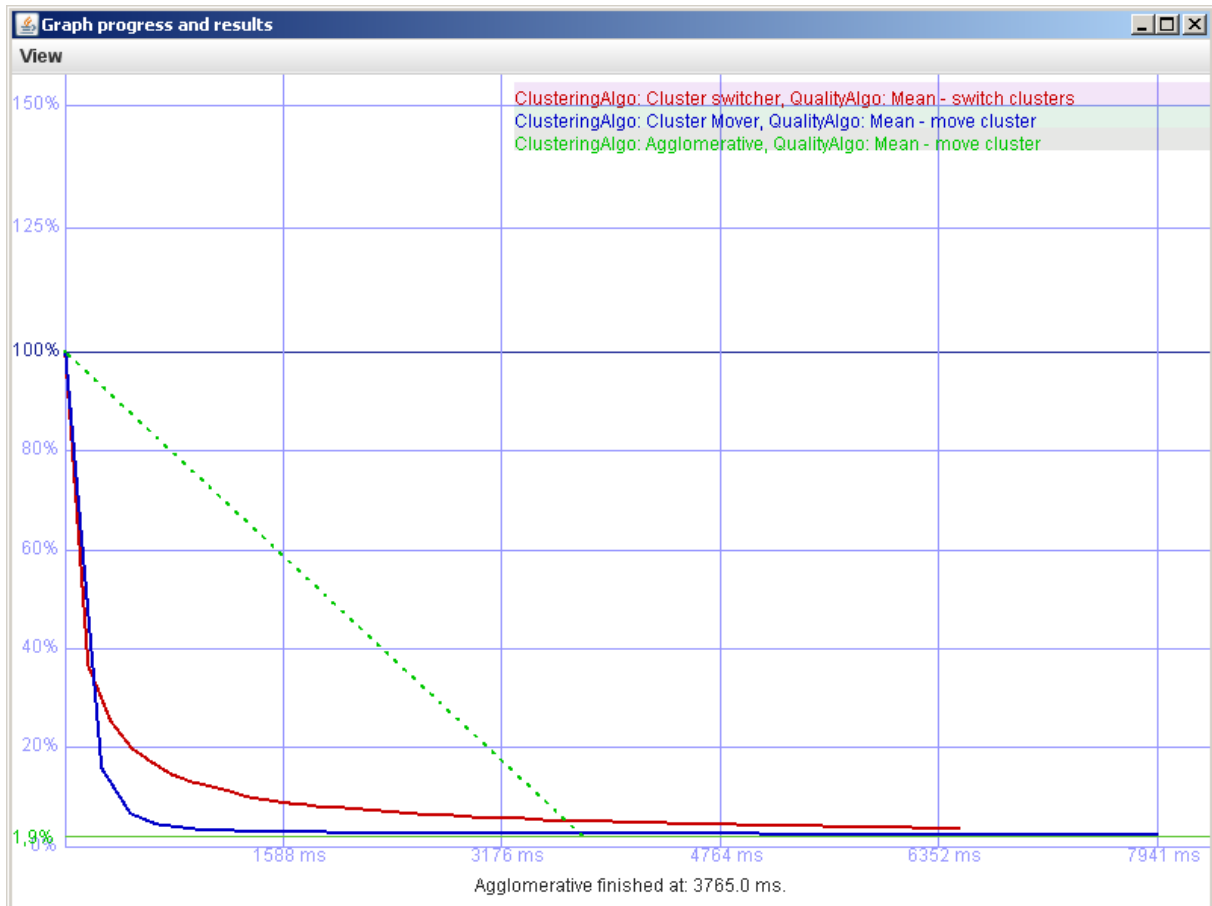


Figure 71 400 points, different levels - Graph showing the clustering progress

6.6 Experiments summary

These experiments have given us several interesting answers regarding the algorithms. Cluster mover proves to be a faster algorithm compared to cluster switcher and scales better. This was expected as the same approach worked out to be the best variant in the previous experiments we did on the subject of partitional clustering. Although the scenario is a bit different this time, cluster mover still seems more dynamic, because it will just try to move a cluster from one parent to another, instead of having to find another cluster to switch with. Having to find this other cluster may be seen as a restriction and can in some cases prevent the algorithm from finding the best solution.

When the clusters are not clearly defined and some points are getting too close other clusters, our algorithms needs more time to identify the clusters. There may even be some points misplaced after a significant amount of time. Much of the reason for this is probably because of the way the algorithm checks if a move should be accepted. As we use the mean method for calculating the quality, only the distances from the two child clusters matters and the distances to each of the points are not directly taken into consideration. This is much faster compared to the SSR-method, but it is a sort of approximation and the clustering may suffer because it doesn't have the full perspective of the situation.

Even though there is a various difference between the quality with our algorithms and the closest agglomerative, the natural clusters could still have been found. An example of this is shown in Figure 72. This could be an example of a sub-cluster of a bigger tree structure. Tree 1 will have a lower and better SSR value than tree 2. Generally it is important to have the most optimized structure of the whole tree, but if the most important task would be for the node on the top to include the four leaf nodes (all the objects in that sub-tree), both trees could be used.

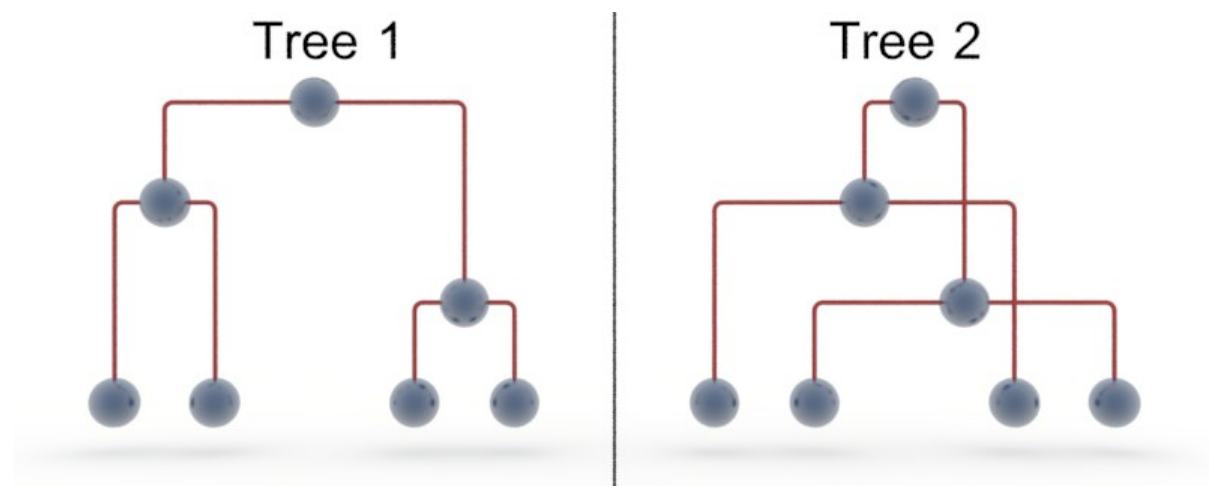


Figure 72 Two different dendrograms

6.7 Scaling of the algorithms

The scalability of the algorithms is one of the most important properties of a clustering algorithm, especially when it comes to the online ones. The closest agglomerative method will always deliver a better result (in theory they could deliver equally good results, but this is not feasible for even small problems within reasonable time). In our experiments systematic algorithms (like the closest agglomerative and the “best point” algorithms) scale poorly in general.

The online algorithm on the other hand, should preferably have the property that one single round of clustering should take the same amount of time no matter how large the input set is. Unfortunately this is not the case.

First off, a larger input set means that the lists containing points and clusters will be larger thus the java functions needed to manipulate and access those lists will work slower. Secondly, and more importantly, the number of clusters in need of updating each step plays a big role in the scalability. If the tree structure is large most of the steps will need to update larger parts of the tree to be consistent. When testing a move where the clusters are ancestors of each other (the move cluster method) all clusters in-between the two clusters have to be updated. Similarly when a move is accepted all the clusters from both clusters to the first common ancestor have to be updated. We have thought of some ways which might improve this, but due to time limits we will keep this to a discussion in the report rather than actually implementing and testing whether it works.

One of the thoughts on this is to not allow the clusters to be ancestors of each other at all or at least make those cases occur rarer than they do now. In general a run of the clustering algorithm will have a much larger amount of rejects than accepts. When rejecting a move where the clusters are not ancestors all that need to be updated are the two clusters which will be tested for improvements, this makes both the check for improvement and the reject step quicker than for the cases where they are ancestors.

We do not really know if all the possible tree-structures can be generated this way. The algorithm would almost certainly need more rounds to reach the same result, and to know if this increase in rounds will outweigh the time gained we need to perform more experiments. When it comes to the move cluster algorithm we also have to take into account the fact that larger clusters needs more rounds to complete. This will impact the graph negatively since we are measuring total time clustering and not average per object.

When it comes to the cluster switcher algorithm each round of clustering would probably outperform the cluster mover. This is because the algorithm already will not allow moves where the selected clusters are ancestors of each other. But as we see in the graph, and as we know for a fact from our experiments the cluster switcher method scales worse when it comes to number of rounds needed to get the same result. The cluster switcher implementation we have used could probably also be improved somewhat so that each step performs faster, but since the gain in time taken to cluster here mainly is because of the additional number of rounds it needs, it would probably be less improvable through this method than the move cluster.

When we did the experiments on the algorithms the time needed for each experiment was measured. These numbers are the basis for the scaling graph (Figure 73). The y-axis of the graph is time taken to complete and the x-axis is the number of clusters in the experiment. Keep in mind that the blue and red lines, which represents the move cluster and switch cluster algorithms, are not the time needed to reach a “perfect result”, but the time to where the clustering graph has flattened out so much that additional work time would result in small

improvements. As the graph shows the closest agglomerative algorithm scales very poorly. It shoots almost straight up, but this is exactly as we had expected it to be.

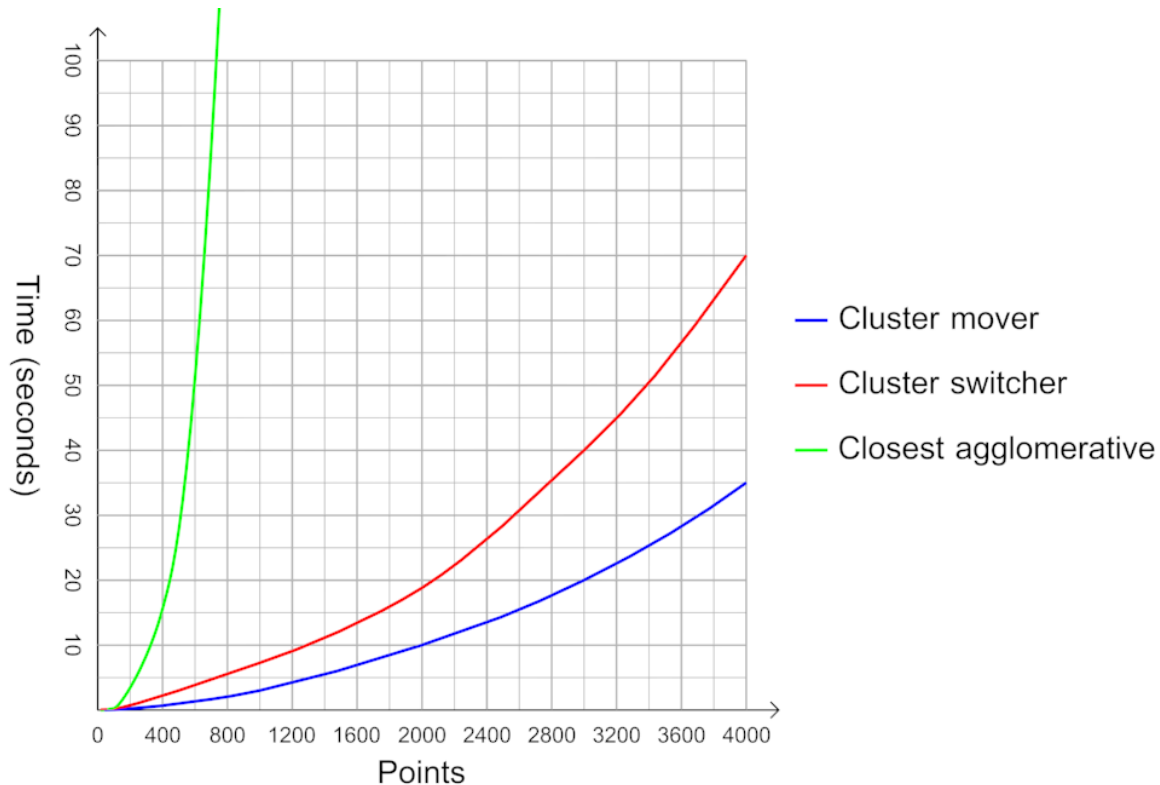


Figure 73 Graph showing the scaling of the algorithms

As a conclusion to the scalability question; we had hoped that the move cluster algorithm would scale better, but at larger problems it still outperforms the closest agglomerative algorithm by far. So the question is how badly a complete result is needed. If the “natural” clusters are what you are interested in the move cluster method could probably suit your needs, but if a more complete solution where the internal structure of clusters also is needed the agglomerative algorithm may so far, be the only way to go, even if it might take vast amount time to finish on really large problems.

7 Conclusion

In this thesis we have researched, developed and tested different hierarchical clustering algorithms. Also a test framework for testing and creating experiments has been built from scratch.

The impressions we have from this research are a bit split. In the cases where the clusters are clearly separated these natural clusters are quite easily recognized. The process of recognizing these distinct clusters does also scale very well by both dendrogram algorithms, especially cluster mover. However, if the elements are placed too close to each other, the algorithm seems to have more problems with finding the optimal solution. This is the case when the clusters have some outliers that are close to other clusters, or when it comes to the structure inside a natural cluster.

As mentioned earlier, this is probably because of how we decide if a move should be kept or not. Improving this method could probably noticeably increase the efficiency of clustering not so distinct clusters, as well as improving the clustering in general. During our experiments of improving this, we got a lot of different results doing minor adjustments, ranging from more or less chaotic clustering to the results we ended up with.

When it comes to the speed of the clustering, as the number of objects increase our algorithm clearly outperforms the closest agglomerative down to a certain quality level. We have also noticed that in both our partitional and hierarchical experiments with the online algorithms the ones which tries to more or less systematically improve clusters tends to scale worse than the ones which randomly performs changes. This is probably a result of the increasingly difficult job of systematically finding appropriate changes as the solution gets better and better.

When it comes to online algorithms they provide another great advantage in environments where new objects are continuously added. Then a clustering process could be running nonstop, where new objects could just be randomly placed in the tree and these will be clustered correctly into the existing structure.

Compared to the partitional clustering methods the hierarchical approach still spends more time of the same problem, but the benefits from the hierarchical approach is its ability to represent all natural clusters without having any idea of how many they are.

Good cluster algorithms are still in the scope of many researchers and there is still a lot of work that needs to be done in order to archive the “perfect clustering algorithm” and we hope our contribution to this subject will have an impact in the world of clustering.

8 Appendix

8.1 References

Internet sites

- 1: Java programming language (visited 10.01.2007)
http://en.wikipedia.org/wiki/Java_programming_language
- 2: Eclipse (visited 10.01.2007)
<http://www.eclipse.org/org/#about>
- 3: Unsupervised learning (visited 08.01.2007)
<http://www.gatsby.ucl.ac.uk/~zoubin/papers/ul.pdf>
- 4: Unsupervised learning (visited 15.01.2007)
http://en.wikipedia.org/wiki/Unsupervised_learning
- 5: Reinforcement learning (visited 15.01.2007)
http://en.wikipedia.org/wiki/Reinforcement_learning
- 6: Clustering (visited 26.01.2007)
http://en.wikipedia.org/wiki/Taxonomic_classification
- 7: Data clustering (visited 26.01.2007)
http://en.wikipedia.org/wiki/Data_clustering
- 8: Clustering algorithms (visited 26.01.2007)
http://www.elet.polimi.it/upload/matteucc/Clustering/tutorial_html/kmeans.html
- 9: Java code convension (visited 22.01.2007)
<http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html>
- 10: Clustering (visited 26.01.2007)
http://www.elet.polimi.it/upload/matteucc/Clustering/tutorial_html/
- 11: Design (visited 23.01.2007)
<http://simple.wikipedia.org/wiki/Design>
- 12: Hierarchical clustering (visited 26.01.2007)
http://www.elet.polimi.it/upload/matteucc/Clustering/tutorial_html/hierarchical.html
- 13: K-means (visited 26.01.2007)
<http://en.wikipedia.org/wiki/K-means>
- 14: Quality Threshold (visited 26.01.2007)
http://en.wikipedia.org/wiki/Cluster_analysis

- 15: Clustering lecture 6, 2/6/02. By Georg Gerber (visited 16.02.2007)
<http://www.mit.edu/~georg/papers/lecture6.ppt>
- 16: Agglomerative Clustering from Vias.org (visited 16.02.2007).
http://www.vias.org/tmdatanaleng/cc_cluster_agglom.html
- 17: Master thesis homepage (Visited 16.02.2007)
<http://fag.grm.hia.no/ikt590/hovedoppgave/lister/lstValgteO1.aspx>
- 18: Agile software development (Visited 11.04.2007)
http://en.wikipedia.org/wiki/Agile_software_development
- 19: Java XML (Visited 12.04.2007)
<http://labe.felk.cvut.cz/~xfaigl/mep/xml/java-xml.htm>
- 20: XML(Visited 12.04.2007)
<http://www.unitedyellowpages.com/internet/terminology.html#X>
- 21: Clusty search engine(Visited 16.05.07)
<http://clusty.com/>

Books

- B1: Pattern classification 2. Edition
By: Richard O. Dura, Peter E. Harb and David G. Stork
ISBN: 0-471-05669-3
- B2: Statistical concepts and methods
By Gouri K. Bhattacharyya and Richard A. Johnson
ISBN: 0-471-03532-7

8.2 Attachments

The following items will be attached to this report (as CD-ROM):

- Source code for our application as Eclipse project (Java)
- All XML files which are used in the experiments
- XML file which shows the local optimum problem