Clustering biological data using a hybrid approach:

Composition of clusterings from different features

Masters Thesis

of

Jens Keller
born on the 5th of May 1982 in Worms

March 31, 2008

Supervisor:
A. Lindlöf

University of Skövde
School of Humanities and Informatics
Bioinformatics Group

TU Darmstadt
Department of Computer Science
Knowledge Engineering Group

Dedicated to

Leonie Keller, Otto Keller, Sabine Keller, Philipp Keller

Liebe Familie, diese Stelle in meiner Arbeit sei Euch gewidmet. Vielen Dank für die bedingungslose Unterstützung in allen Aspekten während meiner langen Studienzeit, aber auch davor. Vieles von dem, was ihr mir mit auf den Weg gegeben habt, war mir während dieser hilfreich, nichts dagegen war hinderlich. Danke für die Geduld, für das Verständnis, die Aufmunterungen und die Mühe. Ebenso für das tagtäglich vorgelebte Selbstverständnis, es als Lebensaufgabe zu sehen, voranzukommen, dazu zu lernen und auch in schwierigen Situationen nicht von einem Ziel abzulassen, solange dieses als erstrebenswert erachtet wird.

Clustering biological data using a hybrid approach:

Composition of clusterings from different features

Jens Keller
keller_jens@gmx.de

Abstract

Clustering of data is a well-researched topic among computer sciences. There exist various approaches designed for different tasks. In biology many of these approaches are hierarchical and the result is usually represented in dendrograms, e.g. phylogenetic trees. However, many non-hierarchical clustering algorithms are also well-established in biology. The approach in this thesis is based on such common algorithms. The algorithm which was developed as part of this thesis uses a graph clustering algorithm to compute a hierarchical clustering in a top-down fashion. It performs the graph clustering iteratively, with a previously computed cluster as input set. The innovation is that it focuses on another feature of the data in each step and clusters the data according to this feature. Common hierarchical approaches cluster e.g. in biology, a set of genes according to the similarity of their sequences. The clustering then reflects a partitioning of the genes according to their sequence similarity. The approach introduced in this thesis uses many features of the same objects. These features can be various, in biology for instance similarities of the sequences, of gene expression or of motif occurences in the promoter region. As part of this thesis not only the algorithm itself was implemented and evaluated, but a whole software also providing a graphical user interface. The software was implemented as a framework providing the basic functionality with the algorithm as a plug-in extending the framework. The software is meant to be extended in the future, integrating a set of algorithms and analysis tools related to the process of clustering and analysing data.

The thesis deals with topics in biology, data mining and software engineering and is divided into six chapters. The first chapter gives an introduction to the task and the biological background. It gives an overview of common clustering approaches and explains the differences between them. Chapter two shows the idea behind the new clustering approach and points out differences and similarities between it and common clustering approaches. The third chapter discusses the aspects concerning the software, including the algorithm. It illustrates the architecture and analyses the clustering algorithm. After the implementation the software was evaluated, which is described in the fourth chapter, pointing

out observations made due to the use of the new algorithm. Furthermore this chapter discusses differences and similarities to related clustering algorithms and software. The thesis ends with the last two chapters, namely conclusions and suggestions for future work. People who are interested in repeating the experiments which were made as part of this thesis can contact the author via e-mail, to get the relevant data for the evaluation, scripts or source code.

Declaration

Clustering biological data using a hybrid approach:

Composition of clusterings from different features

Jens Keller

Submitted by Jens Keller to the University of Skövde as dissertation towards the degree of Master by examination and dissertation in the School of Humanities and Informatics.

31-03-2008

The work in this thesis is based on research carried out at the *Bioinformatics Group*, *School of Humanities and Informatics*, *University of Skövde*, Sweden as well as (since I wrote it during an Erasmus exchange program at the University of Skövde) at the *Knowledge Engineering Group*, *Department of Computer Science*, *Darmstadt University of Technology (TU Darmstadt)*, Germany. I certify that all material in this thesis which is not my own work has been identified and that no material is included for which a degree has previously been conferred on me.

Jens Keller

Contents

Abstract iii

Declaration v

1. Introduction 1
- 1.1. Clustering . . . 1
  - 1.1.1. Non-hierarchical approaches . . . 3
  - 1.1.2. Hierarchical approaches . . . 5
- 1.2. Biological data . . . 6
  - 1.2.1. Sequence . . . 7
  - 1.2.2. Gene expression . . . 7
  - 1.2.3. Sequence motif . . . 8
- 1.3. Clustering software . . . 8
  - 1.3.1. Common clustering software . . . 8
  - 1.3.2. Current situation in biology . . . 10
  - 1.3.3. Software requirements . . . 11
- 1.4. Evaluation . . . 13

2. Hybrid clustering 16
- 2.1. The hybrid clustering approach . . . 16
- 2.2. Underlying clustering algorithm . . . 20
- 2.3. Space and run time analysis of the algorithm . . . 22
  - 2.3.1. Run time analysis . . . 22
  - 2.3.2. Space analysis . . . 24
- 2.4. Features for the hybrid clustering algorithm . . . 24
  - 2.4.1. Sequences . . . 25
  - 2.4.2. Gene expression . . . 25
  - 2.4.3. Motifs . . . 26

3. Software 27
- 3.1. Solutions to the requirements . . . 27
- 3.2. Software architecture . . . 29
  - 3.2.1. System architecture . . . 30
  - 3.2.2. Design patterns . . . 30
  - 3.2.3. Framework concept . . . 31

4. Evaluation 32
- 4.1. Data sets . . . 32
  - 4.1.1. WRKY transcription factors . . . 32
  - 4.1.2. AP2/EREBP transcription factors . . . 32

4.1.3. Indica . . . 33
4.1.4. AtGenExpress . . . 35
4.2. Formal methods . . . 35
4.2.1. Comparing two clusterings . . . 36
4.2.2. Measuring a cluster . . . 36
4.3. Informal methods . . . 37
4.3.1. Inspecting the clustering tree . . . 38
4.3.2. Comparing different levels of the clustering tree . . . 38
4.4. Comparison with common clustering approaches . . . 41
4.4.1. HCE and Clustal . . . 41
4.4.2. Information fusion approach . . . 47
4.5. Results . . . 48
4.5.1. Results from clusterings of the AP2/EREBP group . . . 51
4.5.2. Results from clusterings of the Indica set . . . 55
4.5.3. Results from clusterings of the AtGenExpress group . . . 58
4.5.4. Hybrid clustering as a classifier . . . 61

5. Future work **64**
5.1. AT3G11020 and AT5G05410 from the AP2/EREBP family . . . 64
5.2. Genome-mean expression profile . . . 64
5.3. Information fusion approach . . . 65
5.4. Another graph clustering algorithm . . . 66
5.5. Several software tasks . . . 66
5.6. The vision of the software . . . 67

6. Conclusions **70**

Bibliography **73**

Appendix **77**

A. Glossary **77**

B. AP2/EREBP members used for the evaluation **80**

C. Technical documentation **81**
C.1. Technical solutions to the requirements . . . 81
C.2. MVC architecture of the framework . . . 85
C.3. Design patterns . . . 88
C.3.1. Major design pattern: Observer . . . 88
C.3.2. Minor design pattern: Singleton . . . 89
C.4. Coding conventions . . . 90
C.5. Concrete implementation of the hybrid clustering . . . 91
C.5.1. Main clustering method . . . 91
C.5.2. Hybrid clustering method . . . 93
C.5.3. Underlying graph clustering algorithm . . . 93

List of Figures

1.1. Non-hierarchical clustering 2
1.2. Hierarchical clustering (phylogenetic tree). The image was taken from http://www-personal.umich.edu with the author's permission. 2
1.3. A representation of weather data. The set contains three samples. Each sample can easily be transformed into a vector, namely by interpreting the value of each property p_i as a coordinate value on the i-th axis. It is therefore a vector in three-dimensional space. 3
1.4. Three sequences and the similarity matrix after computing their pairwise similarity. It is obvious that each sequence has the similarity 0 to itself, indicating identity. The higher the score, the less similar two sequences are in this example. It has to be noticed that this example is imaginary. Comparison of these sequences with a common alignment algorithm will lead to different values. 4
1.5. The graph representation of the matrix of figure 1.4. 4
1.6. Illustration of a graph clustering. The image was taken from http://www.physik.uni-wuerzburg.de with the author's permission. 5
1.7. A tree of a hierarchical clustering, computed with *HCE* 6
1.8. The clustering tree from 1.7 was cut, delivering three disjoint main clusters. 6
1.9. HCE (Hierarchical clustering explorer). In the main window, the resulting clustering tree of the computation is shown, which was already cut to gain six clusters. Below is a matrix with different colors indicating the gene expression levels. 9
1.10. A clustering computed with Clustal. The lengths of the branches indicate the similarity of the sequences. The longer the branch, the less similar the sequences. 10
1.11. The current situation of clustering in biology with the user handling several software applications. To communicate between applications, the user has to store, load and transform the data. This makes it challenging to keep the overview on all the files, since they may be distributed over the whole file system on the hard disk. 11

1.12. The size of the data sets becomes important when analysing the set mathematical terms. On the left, even if some elements do not lie on the straight line, the linear character of the set is obvious. In the right diagram, it cannot be determined if the set follows a linear order because the sample is too small. Two or three elements behaving unexpectedly may lead to misinterpretation. Using a larger data set is here the better choice. 14

2.1. Illustration of the hybrid clustering. After the first step, the focus is changed to another feature. This leads to different similarities than before, illustrated by a different positioning of the objects. . 17
2.2. Resulting tree from the hybrid clustering approach. 18
2.3. The hybrid clustering algorithm. 19
2.4. The result of the hierarchical clustering has to be transformed by cutting the tree, leading to a non-hierarchical clustering at the end. This result can then be used for the hybrid approach. . 20
2.5. The underlying graph clustering algorithm. 21

3.1. Model-View-Controller 30
3.2. Framework concept 31

4.1. A screenshot of the application. It shows the similarity set imported from a *Blast* result with a cut-off value of e-10. 35
4.2. The tree view of a clustering result, computed by the hybrid clustering approach. 38
4.3. A cut of the clustering tree leading to the non-hierarchical clustering shown on the left and the matrix view shown on the right. The two marked entries in the matrix indicate the entries $(9, 21)$ and $(21, 9)$ The arrows show that for the entry $(9, 21)$ 9 refers to *AT2G44940* and 21 refers to *AT4G32800*. The entry in the matrix is black because both belong to the same cluster. 40
4.4. Two matrices which resulted from cutting the clustering tree at different levels. 41
4.5. The hierarchical clustering tree computed with *HCE* was cut at two positions leading to two clusterings consisting of two (top) respectively three (bottom) clusters. 42
4.6. The two clusterings computed by the graph clustering with respect to gene expression similarity using the thresholds 0.7 and 0.8. 43
4.7. The comparisons with *Rand Index* and *Adjusted Rand Index* of the clusterings computed by the graph clustering algorithm and *HCE*. The comparison of the two clusterings consisting of two clusters each is shown on the left, while the comparison of the clusterings containing three clusters each is shown on the right. . 44
4.8. The results from the measure computation for the two clusterings computed by the graph clustering algorithm (left) and *HCE* (right). 44

4.9. *HCE* provides a lot of functionality. A cut bar (top) helps to efficiently cut hierarchical clustering trees. The software can also compute *k-means* clusterings (bottom). 45
4.10. Two clusterings computed by *HCE* (gene expression, top) and *Clustal* (sequences, bottom). 46
4.11. The hybrid clustering shows that *ATG25490*, *AT5G51990*, *ATG25480* and *ATG25470* are rather similar according to both gene expression and sequences. They remain in the same cluster at the second level of the clustering tree. 47
4.12. Configuration of a hybrid clustering. For the first round a similarity set computed by *Blast* containing *E values* (sequence similarity) was used. The lower the value, the better the similarity. In contrast, when computing a *Pearson correlation*, a high value (with the upper bound one) indicates a high similarity (cf. the second round). The hybrid clustering is performed with three iterations, with a different similarity set in each iteration. 50
4.13. A tree resulting from a hybrid clustering for three rounds. The root (depth 0) represents the set of all elements before the clustering and a leaf node (depth four) represents a single element. The nodes at levels one, two and three represent the clusters of the first, second and third iteration. 51
4.14. Clustering tree of the first computation. The elements *AT3G11020* and *AT5G05410* are in the same cluster at level one, but not at level two and three. 53
4.15. The two matrices resulting from the cut at level one (left) and level two (right). It can be observed that *AT3G11020* and *AT5G05410* belong to the same cluster at level one indicating sequence similarities, but not at level two indicating motif occurence similarities. 53
4.16. Plot created by the clustering software. It resulted from the cut of the tree of the second computation at level two indicating gene expression similarity with threshold 0.9. *AT3G11020* and *AT5G05410* are in the same cluster. 54
4.17. Clustering tree and matrix from the cut of the second clustering described in section 4.5.1 at level three. *AT4G25470*, *AT4G25480*, *AT4G25490* and *AT5G51990* are in the same cluster meaning that they are similar with respect to sequences (*E value <e-10*), gene expression (*Pearson correlation* >0.9) and motif occurrence correlation (*Pearson correlation* >0.8). 55
4.18. The result from the measure computation for the cuts at the first (left) and the second level (right). It can be observed that the measures indicate a quality improvement for the cut at the second level although there are more singletons. 57
4.19. The result from the measure computation for second clustering, focusing only on gene expression 57

4.20. The matrices from the clustering of the *AtGenExpress* group, resulting from cuts at levels 1-3. It can be observed that there is not much of a change between levels one and two, but at level three many clusters were decomposed into smaller sub clusters (less elements are in the same cluster and thus the matrix has less black entries). 59
4.21. Three parts from the clustering tree, which resulted from a cut at level one. As the leftmost and the middle parts show, a lot of elements land in the second cluster. This can be seen by looking at the position of the scrollbar in the middle picture. But there are also a lot of elements which are the only ones in their cluster, depicted by the right part. Therefore, the total number of clusters is comparatively high, although the threshold of *e-2* is quite relaxed. 60
4.22. The tree resulting from the cut at level three. The large cluster, shown in figure 4.21, was divided into two large clusters after clustering according to gene expression with a threshold of 0.95. 60
4.23. Charts resulting from counting the number of occurences of *AP2/EREBP* members for each cluster. The upper chart shows the occurences in the clustering resulting from cutting the tree at level one, while the lower chart shows the occurences in the clustering resulting from cutting the tree at level two. The configuration of the clustering is the same as for the clustering described in section 4.5.3. 63

5.1. The vision. The user can still use the old applications but does not recognise it. The framework either communicates with external applications (through wrappers), or uses internal functionality, and provides a consist displaying of the results. Several tools for analysing, as for instance measures, are also used via the framework. 69

B.1. AP2/EREBP members used for the evaluation, including the gene expression. 80

C.1. The extension points of the framework, including the relevant interfaces and classes. 87
C.2. Interfaces for the *Observer* pattern. 88
C.3. The implementation of the main routine of the hybrid clustering. 92
C.4. The implementation of the hybrid clustering part. 93
C.5. The implementation of graph clustering algorithm. 95

1. Introduction

This chapter gives an overview of the topics involved in this thesis. Basically, there are four sections. In this thesis, a novel clustering approach was developed and evaluated with an emphasis on biological data. The first section gives an introduction into clustering. It will depict that there are two different general approaches. One common approach, namely a graph clustering approach was used as the underlying clustering algorithm on which the novel clustering approach is based. This approach is called "hybrid" because it uses a common clustering approach together with different features of the data to construct a hierarchical clustering. Section two introduces different features of biological data used by the hybrid clustering approach. As part of this thesis a software for the new clustering approach was implemented including a graphical user interface, making the clustering process more efficient. Section three introduces clustering software, especially related to biology. It will introduce some common clustering programs, show that many different programs are involved in the clustering process and depict the requirements of a clustering software. Several data sets containing biological data where used to evaluate the software. These data sets are introduced in section four.

1.1. Clustering

Clustering is an extensive field among data mining as well as among bioinformatics. The task is to decompose a set of data in a way that similar data is grouped together in a so-called cluster. There are many applications of clustering, e.g. finding a common topic for different data, maybe cluster HTML pages resulting from a web search query "jaguar", so that the user can directly see which pages belong to the topic "cars" and which belong to the topic "animals". The result will be a number of sets, each set representing one topic and containing a number of pages delivered by the search for "jaguar". In bioinformatics a common task is for instance to gain information about species relations. This approach differs slightly as it is hierarchical in nature. The result is usually represented in a dendrogram, e.g. a phylogenetic tree. Figure 1.1 shows a non-hierarchical clustering. In this case, a k-means clustering. An efficient implementation of such a clustering algorithm is for instance given by Kanungo et al. [13]. The large dots visualize the cluster centroids (the mean). The small dots represent the data objects to cluster, e.g. HTML pages. Each object is allocated to the cluster with the closest cluster centroid. After each allocation, the cluster centre is re-computed as the mean of its components. A hierarchical clustering is illustrated by figure 1.2. It is a phylogenetic tree depicting relations between genes in different species. It can be observed that larger clusters are composed out of smaller sub clusters.

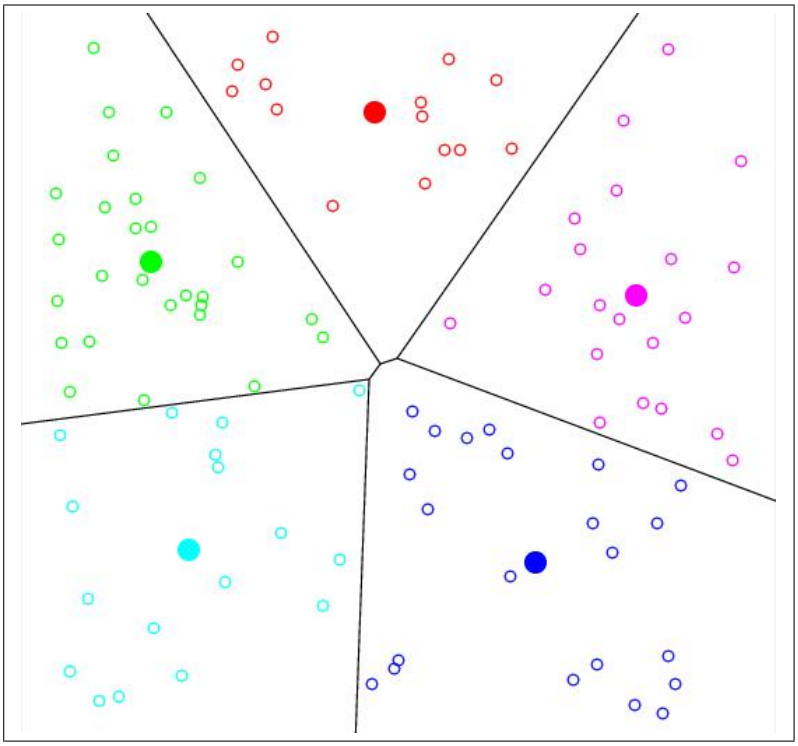

Figure 1.1.: Non-hierarchical clustering

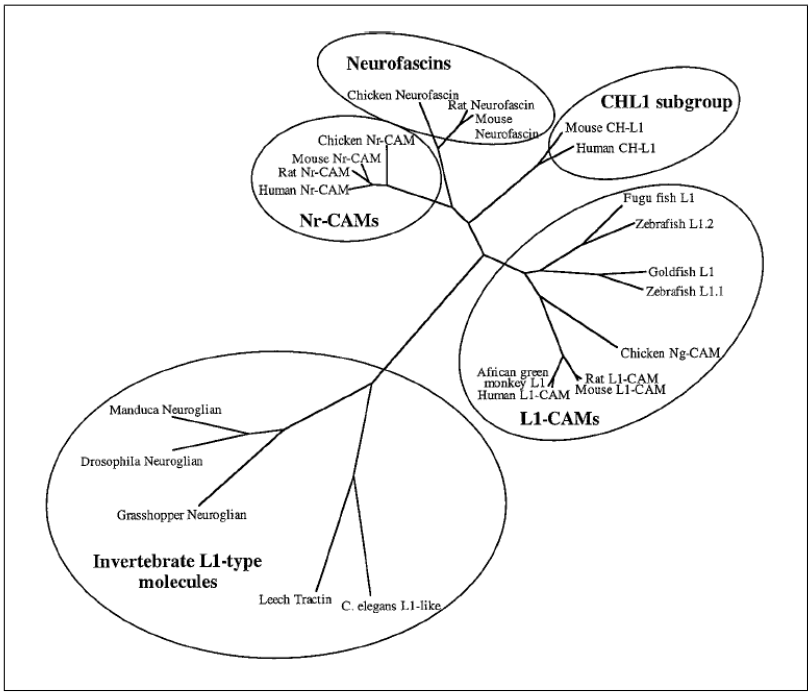

Figure 1.2.: Hierarchical clustering (phylogenetic tree). The image was taken from http://www-personal.umich.edu with the author's permission.

1.1.1. Non-hierarchical approaches

There are various approaches for computing a non-hierarchical clustering. One example was already introduced, the k-means clustering. In this case an object is usually represented by a vector. The objects to cluster can already be on hand in such a representation or in a representation similar to it. Usually the data has to be transformed in such a vector representation. An example is weather data used for instance to forecast rain. Each record consists of values like air pressure, humidity and temperature leading to a table as illustrated by figure 1.3. Each column of the table is interpreted as a column vector, with the column header as the vector identifier.

	Sample 1	Sample 2	Sample 3
Air pressure	970 hPa	1013.6 hPa	1002.1 hPa
Humidity	20.0%	15.5%	72.3 %
Temperature	20°C	17°C	24°C

Figure 1.3.: A representation of weather data. The set contains three samples. Each sample can easily be transformed into a vector, namely by interpreting the value of each property p_i as a coordinate value on the i-th axis. It is therefore a vector in three-dimensional space.

Sometimes the transformation is just not possible, or cannot be computed efficiently enough. A popular example related to biology is comparing protein or DNA sequences. A task may be finding out whether two proteins from different species have a common function. A very high similarity between their sequences can indicate this. To compute the similarity of sequences, they are represented as strings. Since such sequences might have different length it is not reasonable to transform them into a vector and compute the distance between them. Instead of this, a so-called alignment of the strings is performed, which will also deliver a similarity value.

This value is an indicator for the number of deletions, insertions and substitutions of characters which are performed on one sequence string, to make it identical to the other one. There exist various approaches to solve this problem. However, aligning sequences is not the focus of this thesis and the interested reader should refer to the richly available information about this topic. Interesting for this thesis is that aligning sequences delivers a similarity score. Comparing a number of sequences pairwise thus leads to a matrix of similarity values. Figure 1.4 shows a small similarity matrix resulting from the pairwise alignments of three DNA sequences.

S1: AGCAATAAATTAAGTGATGCAACTCTATAAGATATCGTAAGATGATGAAATGTTTAATAAACTAATTAA
S2: AAAATAAGCAACTGATTCAATTTTGTATCAAATCATACACTTAAAATTATCTCTTTTAAAAA
S3: TTTCCTTTTCATTTGTACCCAAAATTTGATGTTTCACTAGCTCTTACTGGATAATTGGAAGAAAAACAAGTGTGA

	S1	S2	S3
S1	0	15	8
S2	15	0	30
S3	8	30	0

Figure 1.4.: Three sequences and the similarity matrix after computing their pairwise similarity. It is obvious that each sequence has the similarity 0 to itself, indicating identity. The higher the score, the less similar two sequences are in this example. It has to be noticed that this example is imaginary. Comparison of these sequences with a common alignment algorithm will lead to different values.

Such a matrix can also be used to compute a clustering. The basic underlying idea is the same: objects which are rather similar should be in the same cluster and objects which are different should be in different clusters. Contrary to the vector representation we do not use the geometrical distance of vectors of the objects but pre-computed similarities from the alignment. We cannot make any use of the location of vectors in space as the k-means clustering does, since we just do not have such a vector representation of the sequences. Most of the clustering approaches using pre-computed similarities are graph-theoretical. They interpret the set of objects as a set of nodes, and the similarity matrix is interpreted as a matrix of weighted undirected edges, each (i, j)-entry (respectively (j, i), since the graph is undirected) indicating the weight of the edge which connects i and j. How such a representation could look like is illustrated in figure 1.5.

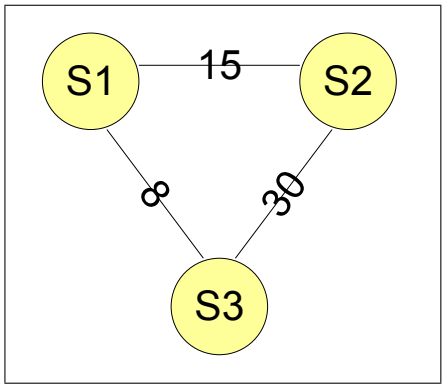

Figure 1.5.: The graph representation of the matrix of figure 1.4.

The task is then to find a decomposition of the graph into disjoint sub graphs. Each sub graph is then interpreted as a cluster. This separation has to be done in a way that the edges between nodes in the same sub graph (the so-called *intra-cluster* distances) have on average a low weight, and the edges between

two nodes of a different sub graph (the so called *inter-cluster* distances) have on average a high weight. Figure 1.6 illustrates such a graph-theoretical clustering by cutting edges to achieve disjoint sub graphs. Not that in this example the graph is not complete. If we transform a matrix of pairwise similarity values into a graph, the graph will always be complete. In general graph clustering algorithms also work on incomplete graphs.

The algorithms focus on several measures, e.g. they try to minimize the cluster diameter (which is the maximum distance from a node n_i to a node n_j in the same cluster). A recent approach was proposed by Flake et al. [8] using so-called cut trees. This approach is described in more detail in section 5 since it is a candidate for being used to perform a hybrid clustering, in the same way as the graph clustering algorithm used within this work. The resulting clusterings gained by the current hybrid approach and a hybrid approach unsing the graph clustering algorithm described in [8] could then be compared.

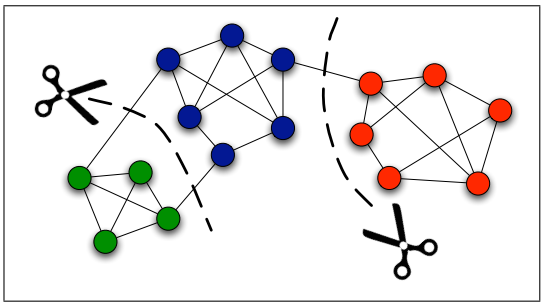

Figure 1.6.: Illustration of a graph clustering. The image was taken from http://www.physik.uni-wuerzburg.de with the author's permission.

1.1.2. Hierarchical approaches

Hierarchical approaches differ from the approaches introduced in the former section. The result of such approaches is not a partition of the data set, but a dendrogram. There are two ways to perform a hierarchical clustering, top-down (also known as divisive) and bottom-up (also known as agglomerative).

In the top-down approach the whole set is at the beginning treated as one cluster containing all elements and iteratively decomposed into smaller clusters. The bottom-up approach starts with as many clusters as there are objects to cluster, each cluster containing one element. In each step, the two most similar clusters are melted to one new cluster consisting of the elements the two former clusters contained. This step is also performed iteratively. Part of the problem is computing the distances between two clusters. Obviously, it mainly depends on the distance between the elements the clusters contain. There are three common approaches to compute the distance between two clusters:

- **Single-link:** The minimum distance d_{min} between two elements of the clusters C_1 and C_2, calculated as

$$d_{min}(C_1, C_2) = \min\{d(x, y) | x \in C_1, y \in C_2\}$$

- **Complete-link:** The maximum distance d_{max} between two elements of the clusters C_1 and C_2, calculated as

$$d_{max}(C_1, C_2) = \max\{d(x, y) | x \in C_1, y \in C_2\}$$

- **Average-link:** The average distance d_{avg} between two elements of the clusters C_1 and C_2, calculated as

$$d_{avg}(C_1, C_2) = \frac{\sum_{x \in C_1, y \in C_2} d(x, y)}{|C_1| \, |C_2|}$$

Such a hierarchical clustering leads then to a tree, with the root node representing the cluster containing all elements and the leaf nodes representing a single element. A hierarchical clustering can always be transformed into a non-hierarchical clustering by cutting the tree at a certain level, leading to a set of sub trees. Each root node of a sub tree then represents one cluster, as illustrated by the figures 1.7 and 1.8.

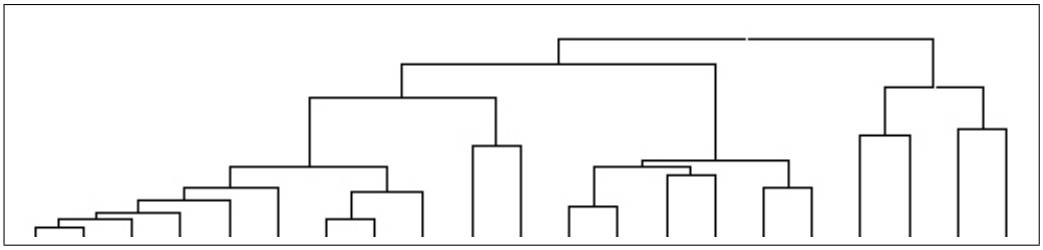

Figure 1.7.: A tree of a hierarchical clustering, computed with *HCE*

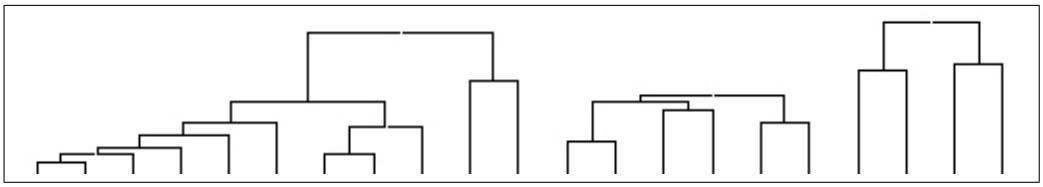

Figure 1.8.: The clustering tree from 1.7 was cut, delivering three disjoint main clusters.

1.2. Biological data

The main task was developing and evaluating a novel clustering using biological data. The data used in this thesis contains information about genes and proteins. This data has many different features which will shortly be introduced.

The idea is to make use of different featuers of the data for the clustering. This idea is related to information fusion, but while information fusion focuses on various features at the same time (it "fuses" the values of the features to an average value), this approach focuses only at one feature at the same time, but it is performed iteratively, focusing on another feature at each step.

It is often the case that some objects are less similar regarding one feature, but more similar regarding a different feature and thus it is reasonable to firstly cluster according to the former "rough" feature, and use the latter more fine-grained feature as a refinement. However, doing it the other way around may also deliver interesting results since there might be elements in the data set behaving different from the rest. Such members can be identified by clustering in another way than the majority of the data suggests. The following sections introduce the features being used in this thesis.

1.2.1. Sequence

The sequence of amino acids or the sequence of nucleotides is a feature which is very basic in protein and DNA analysis. The sequence is usually represented as a string, either consisting of characters representing one of the amino acids (proteins) or a nucleotide (DNA). For instance

$$\text{AT1G68550: } ACAACTTTTAGTTAGCTCAATTTTATTT$$

is such a sequence. *AT1G68550* is a locus identifier (a locus is a fixed region on a chromosome, here the position of a gene) referring to a gene. This specific gene encodes a transcription factor in the species *Arabidopsis thaliana*. Additionally, the transcription factor is a member of of the *ERF* (ethylene response factor) subfamily B-6 of the *ERF/AP2* transcription factor family (see glossary for further information). *Arabidopsis* is a family of small flowering plants related to cabbage and mustard, as mentioned in [1] and [22]. The sequence shown here is a short section from the gene's promoter region. The characters within the sequence refer to the four DNA nucleotides (A=Adenine, C=Cytosine, G=Guanine, T=Thymine).

1.2.2. Gene expression

When a gene is activated, it is transcribed into an mRNA which is thereafter translated into a protein. The amount of mRNA in the cell reflects the expression level of the gene. A common approach to measure gene expression is using Microarrays. Here, DNA spots (either bounded cDNA or synthetical oligonucleotides) are put on a surface. After that, DNA from the object to analyse is labeled with fluorescent dye and put on the array. This DNA binds now to the spots on the array with the matching complementary sequence. After washing the array, only the hybridized parts remain on it. The fluorescence signal is then read with a laser and translated to the level of gene expression.

1.2.3. Sequence motif

A sequence motif is a short nucleotide or amino acid sequence pattern (usually between 5 and 30 nucleotides, respectively amino acids) which is conjectured or known to have an important biological role. On the one hand, the occurrence of a motif in the exon of a gene may indicate that it encodes a certain *structural* pattern. On the other hand, the occurrence of a motif in the promoter region of a gene may indicate that certain transcription factors bind to it. Sometimes there is a certain variability in the sequence motif and thus a common technique is to use consensus sequences. For instance, the motif

$$\text{HD-Zip 2: } CAAT(G/C)ATTG$$

is such a consensus sequence motif. The middle term (G/C) indicates that at this position either Glycosine or Cytosine can occur, this is variable. The motif is related to the *homeodomain-leucin zipper* (HD-Zip) factors, a class of proteins that seems to be peculiar to plants. Sessa et al. state in [20] that the study of the DNA-binding properties showed that the *HD-Zip 2* recognizes exactly the pseudo palindromic pattern above (pseudopalindromic because of the complementary relation between left and right). Sequences which contain this motif are according to [20] candidates where the *HD-Zip 2* transcription factor binds to.

1.3. Clustering software

As part of this thesis a clustering software was implemented. This section introduces two common clustering programs used in biology, namely *HCE* and *Clustal*. The current situation in biology is illustrated, depicting that there are various programs involved in the clustering process. In the last section the clustering software requirements from the user's point of view are listed and explained.

1.3.1. Common clustering software

There are various programs involved in clustering and many of them are especially designed for analysing biological data. A popular example is the *Blast* family. *Blast* collects tools for analysing data, among others tools for searching proteins sequences in data bases. A common clustering software is the *Hierarchical Clustering Explorer* (*HCE*) . This program was developed at the University of Maryland. It provides various functionality, e.g. different similarity measures (Euclidean distance, Pearson correlation or Manhattan distance), different cluster-distance measures (Single-, complete-, average-linkage), cutting the cluster tree to get a non-hierarchical clustering and so on.

Figure 1.9 is a screenshot of *HCE*. Here a clustering of raw data, containing gene expression data from a part of the *Arabidopsis* family, was performed.

The result is a hierarchical clustering represented by a tree. It can be observed that a bar is used to cut the clustering tree, leading to six clusters. This clustering can be printed to a file in plain-text format.

Figure 1.9.: HCE (Hierarchical clustering explorer). In the main window, the resulting clustering tree of the computation is shown, which was already cut to gain six clusters. Below is a matrix with different colors indicating the gene expression levels.

Another program is *Clustal*. *Clustal* computes multiple sequence alignments under use of pairwise alignments and a hierarchical clustering. In the first step of the computation, *Clustal* computes pairwise similarities of the sequences. The result is used to compute a hierarchical clustering represented by a tree. This tree is then used as a guide tree to compute a multiple alignment. Such a clustering tree computed by Clustal is shown in figure 1.10. In contrast to *HCE*, *Clustal* focuses on the sequences respectively sequence alignments for the computation.

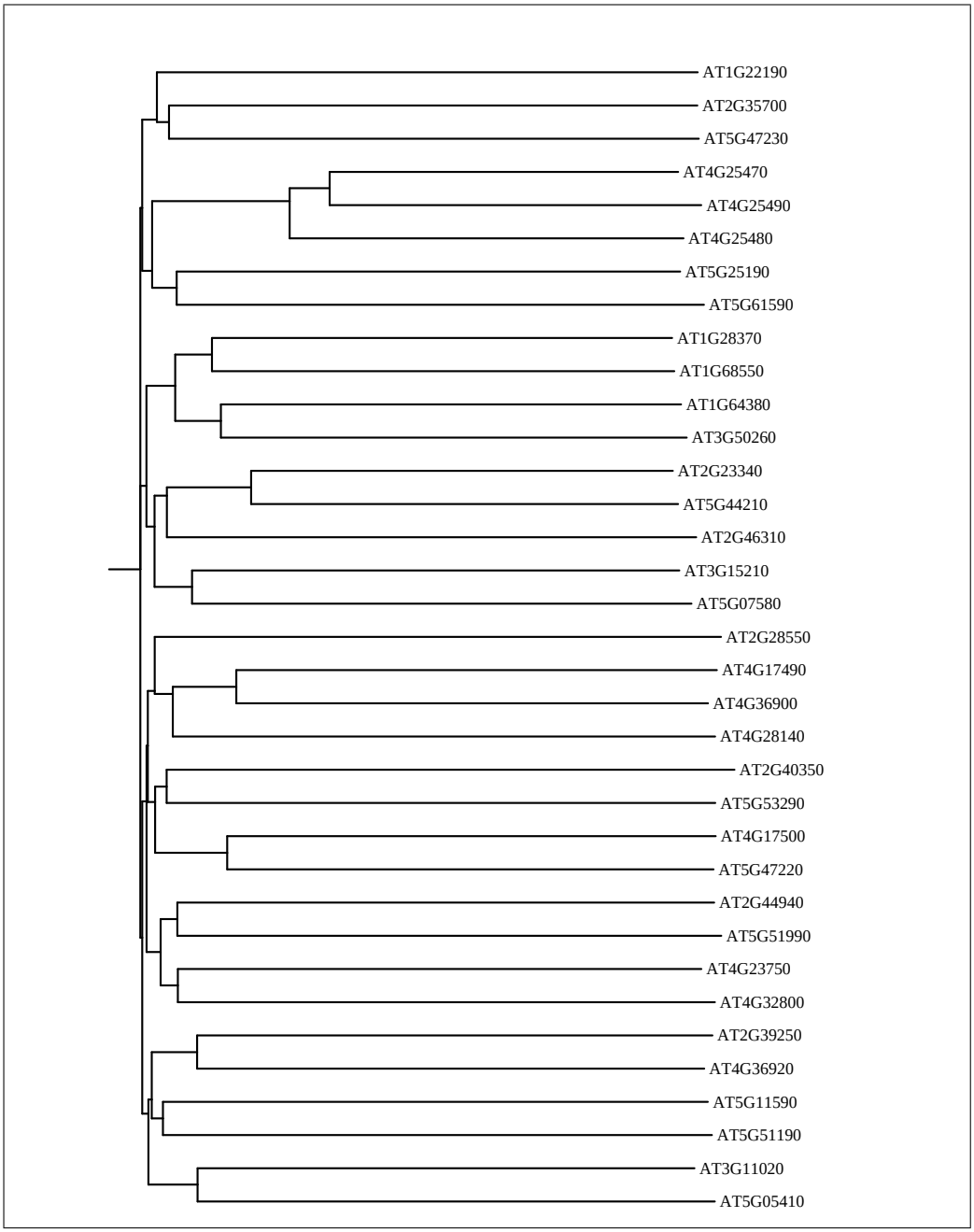

Figure 1.10.: A clustering computed with Clustal. The lengths of the branches indicate the similarity of the sequences. The longer the branch, the less similar the sequences.

1.3.2. Current situation in biology

There are numerous programs involved in the clustering process in biology as well as in general. They focus on several tasks, some of them are only used for specific tasks (e.g. only computing pairwise similarity values of sequences) and some of them cover almost the whole clustering process. They also vary in

appearance, there are full programs including a graphical user interface, command-line based programs and programs accessible via a web interface. The user has to handle all of the programs he needs as well as the data being involved in the clustering process. Different programs may store the produced data at different locations. If the data produced by one program should be used as input for another program, sometimes this cannot be done directly due to format conflicts, and thus many users also have script files transforming the data into the desired format. Figure 1.11 illustrates the current situation with the user handling various software applications as well as different data produced by these programs.

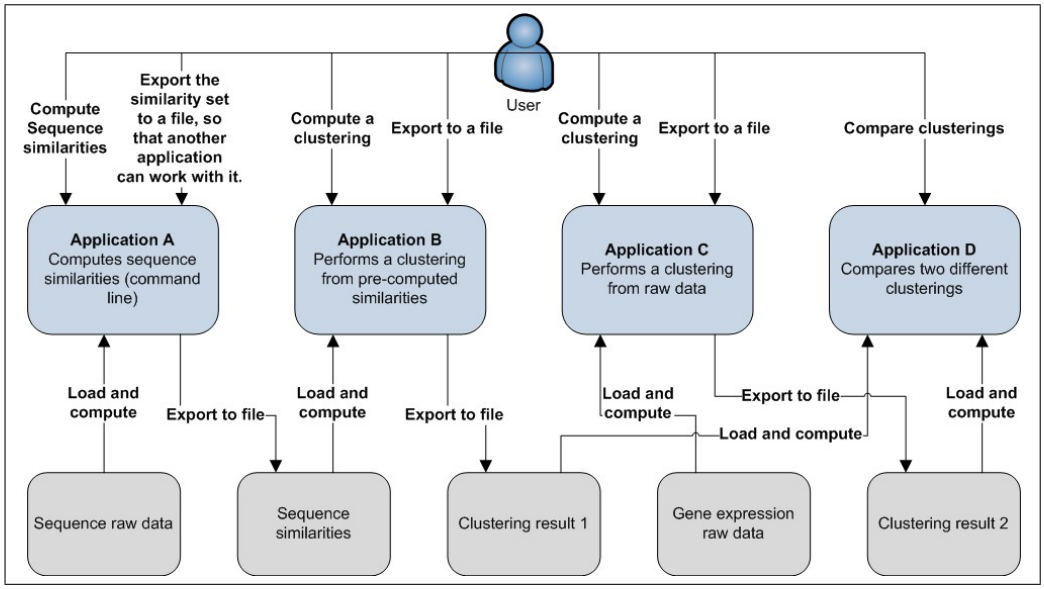

Figure 1.11.: The current situation of clustering in biology with the user handling several software applications. To communicate between applications, the user has to store, load and transform the data. This makes it challenging to keep the overview on all the files, since they may be distributed over the whole file system on the hard disk.

1.3.3. Software requirements

This section gives an overview of the identified user requirements (identified by the prefix "UR") of the clustering software. How the requirements are fulfilled will be shown in chapter 3. The software was implemented as a framework. Developing a framework leads to additional requirements. These requirements are important for people interested in modifying or extending the framework and thus listed in the appendix as framework requirements (identified by the prefix "FR").

UR1:

Three different kinds of data should be distinguished: Raw data, data containing similarities and data containing clusterings.

All this data is involved in the clustering process. Raw data (for instance gene expression data) can be used to compute pairwise similarities between elements. These similarities are used to compute a clustering.

UR2:

Different views to visualise the data sets should be available.

Data containing a clustering is different from raw data or data with similarity information. Consequently there should be specific visualisations. There are also different ways of representing data (e.g. represent pairwise similarities as a matrix, a weighted graph or just as a list).

UR3:

It should be possible to import data sets from foreign software.

It should be possible to import for instance similarity sets computed by other software applications.

UR4:

An export function should make the computed data accessible for foreign software.

Consequently, data produced by the software should be usable by foreign software applications (e.g. software comparing or measuring clusterings).

UR5:

Tools for analysing the data should be included in the software.

Tools can be measuring or comparing clusterings computed by the software.

UR6:

Different users working on the same computer should have their own configuration and data home directory to work on.

If two or more users work on the same computer but on different data the data should be separated i.e. one user should not see the other users' data.

UR7:

The hybrid clustering algorithm should be integrated in the software.

The hybrid clustering algorithm should be part of the clustering software.

UR8:

A wizard should help the user configuring and launching the algorithm.

This should help making the process more efficient. E.g. possible input sets for the clustering should be listed by the wizard so that the user does not have to search for them manually.

UR9:

All the data the user currently works on should be listed in a browser-like view.

The user wants to have a good overview of the data being involved in the current task.

UR10:

A tutorial should be part of the software.

A tutorial should help the user getting started with the software.

1.4. Evaluation

The main aim of this thesis was to develop a software application for clustering data following a novel approach and evaluate it under the use of biological data. On the one hand it is more helpful to have a small data set with carefully selected elements to point out noticeable peculiarities. On the other hand the clustering approach in this thesis differs from the common approaches and should also be evaluated under mathematical aspects.

The tasks of pointing out biological observations and evaluating a clustering under mathematical aspects lead to differerent requirements on the data sets. Studying specifics under biological aspects may require a smaller data set. Contrary, a mathematical measuring of the clustering leads to the need of a larger data set. In a small data set, a few elements which do not behave in the expected way may lead to a result deviating from the average. This may lead to misinterpretation. Having a larger data set should avoid this case. Figure 1.12 illustrates this in an abstract way. In this example it has to be determined if a set of values has a linear character. Obviously, if the sample is to small the result may be misleading due to incorrect values (which may just result from measurement errors).

An important part of software development is testing and debugging. Semantic errors in the program may not cause obvious errors, e.g. the program will not terminate with an error message. It can happen that the program still seems to do what it is expected to do, but in fact the output is

not correct. In terms of clustering a semantical error can for instance lead to an element being present in two disjoint clusters at the same time while another element is not present in any cluster. The clustering may seem correct, even counting the elements will not reveal the error. Finding such errors is even harder if the data set is very large and thus it is reasonable to use a very small data set for testing and debugging.

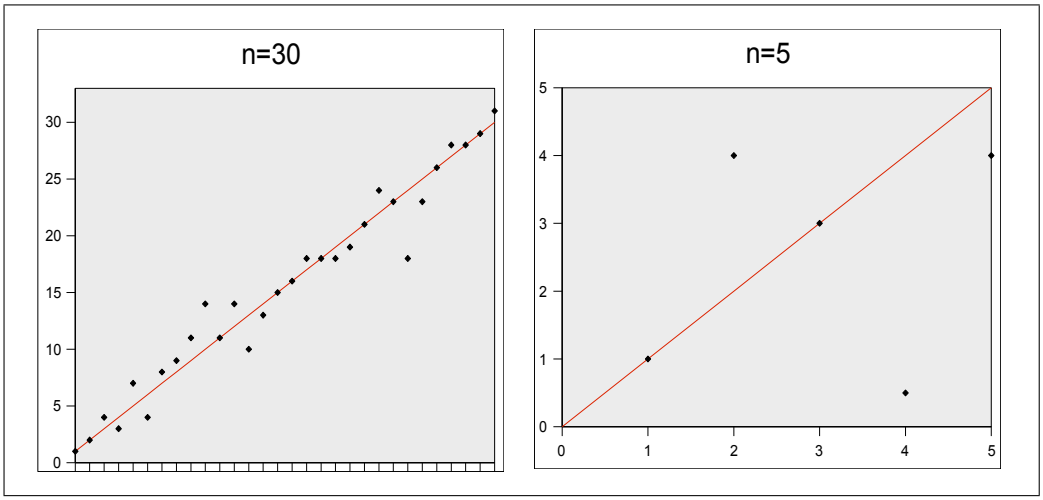

Figure 1.12.: The size of the data sets becomes important when analysing the set mathematical terms. On the left, even if some elements do not lie on the straight line, the linear character of the set is obvious. In the right diagram, it cannot be determined if the set follows a linear order because the sample is too small. Two or three elements behaving unexpectedly may lead to misinterpretation. Using a larger data set is here the better choice.

Four groups of data sets were used for the evaluation. The first group contains only one data set (only one feature) and is very small. It is gene expression data from a cold stress study in *Oryza sativa cv Indica*, to which in the following will be referred just by *Indica*, a cultivar of rice. The data was produced from an in-house experiment at *Göteborg University*. This set is only used for determining if the clustering algorithm works correctly and to help improving the outputs and displays. It can be directly seen if the computed clustering is correct or if the display of a clustering makes sense (or if it should be improved). To be more precisely, the data is from differentially expressed genes in the *WRKY* family. This is a family of transcription factors. For more information about the *WRKY* family the reader should for instance refer to [24].

The second group of data contains data related to *Arabidopsis*, more precisely data from the *AP2/EREBP* family of transcription factors. These transcription factors have a central role regarding abiotic stress in this species, according to the authors of [17]. There are three members of this family being

very similar and having a common function in cold stress resistance, as Gilmour et al. state in [10]. The data contains 34 members out of this family. Three different data sets belong to this group, namely sets with sequence, motif and gene expression data.

The third group contains also data related to *Indica*. It is from the same experiment as the *WRKY* data. In biology, *Arabidopsis* and cultivars of rice are often analysed together. Nakano et al. give in [18] a complete overview of the *AP2/ERF* gene family (which refers to the same transcriptional regulators as *AP2/EREBP does*) in *Arabidopsis*, and perform a comparative analysis between these genes in *Arabidopsis* and rice. The used data group here was larger than the data of the *AP2/EREBP* group, it contains the entire data of the differentially expressed genes, i.e. 1301 elements. Two data sets were used for the clustering, sequence similarities and gene expression.

The last group contains data from the *AtGenExpress* experiment by d'Angelo et al. described in [4]. It contains all the differentially expressed genes in *Arabidopsis*. The set is large, it contains about 2000 genes from different families. It is used to find out if the clustering algorithm might be suitable as a classifier. Three data sets were used, with sequence data, motif data and gene expression data.

2. Hybrid clustering

This chapter describes the ideas behind the novel approach. A common clustering algorithm was taken and performed iteratively to compute a hierarchical clustering and thus the approach is a hybrid one. In each iteration the underlying clustering algorithm gets another data set as input reflecting a different feature of the data. The algorithm itself is introduced and discussed in the first section. Section two describes the underlying algorithm - a non-hierarchical clustering using a graph algorithm. The third section then analyses the algorithm with respect to time and space consumption. The last section illustrates how the different features introduced in section 1.2 are used as input for the hybrid clustering algorithm.

2.1. The hybrid clustering approach

As mentioned in section 1.1 there are mainly two different approaches in clustering. Biological data consists of various features which may all be useful. The idea behind the approach from this thesis is to take a non-hierarchical clustering algorithm and cluster the objects according to a certain feature, leading to a set of clusters. Each cluster is then taken as input for the algorithm again, but now the algorithm clusters according to another feature (of course maybe also according to the same feature again, to achieve a better refinement; but this is not of a high interest here, since it basically leads to a classical hierarchical clustering focusing on a single feature). This may be repeated for a number of times leading to a hierarchical clustering in a top-down fashion as illustrated by figure 2.1.

Figure 2.1 depicts that the hybrid clustering uses a classical clustering algorithm which is executed for several times to achieve a hierarchical clustering. The innovation is that the algorithm focuses on a different feature in each iteration. The clustering tree also looks a little bit different from the results of "classical" hierarchical approaches which are used in bioinformatics. The result of common hierarchical clustering algorithms such as the *Neighbour-Joining* or the *UPGMA* algorithm is always a binary tree (it may even be an unrooted tree, but this is not of importance here), i.e. each node (except the leaf nodes) has two successor nodes, cf. the clusterings from *Clustal* illustrated in figure 1.10.

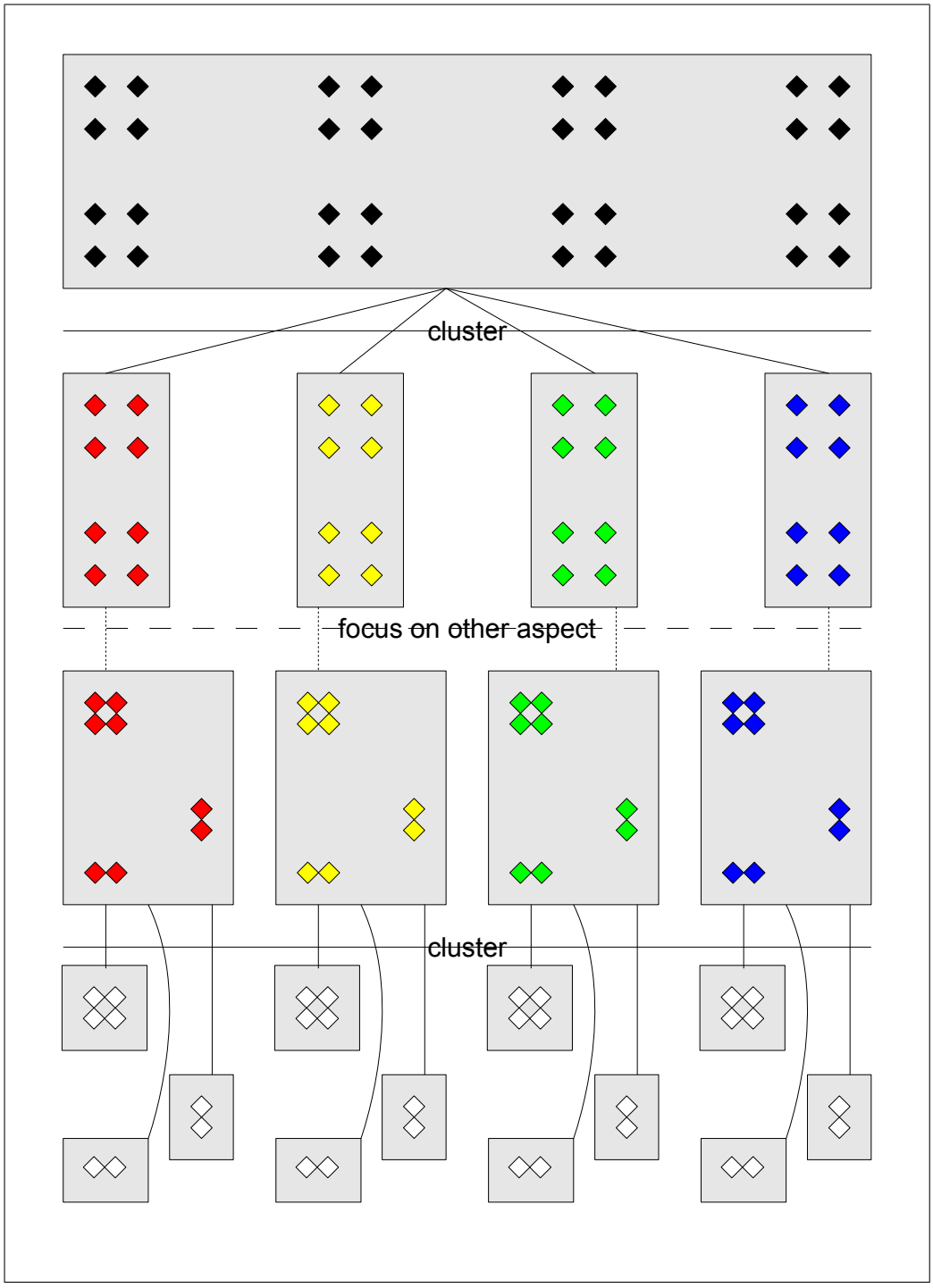

Figure 2.1.: Illustration of the hybrid clustering. After the first step, the focus is changed to another feature. This leads to different similarities than before, illustrated by a different positioning of the objects.

In contrast to the tree shown in figure 1.2, resulting trees from the hybrid approach are always rooted and can have more than one successor (cf. figure

2.2). Here, the root node at level 0 represents the initial set of all objects, with which the algorithm was started. It then clusters according to one certain feature, e.g. gene expression, leading to a clustering consisting of three clusters at level 1. Each of those clusters are then clustered again, but according to another feature. As the result of the middle set at level 1 shows, it may also happen that the set is clustered again to one single cluster, if the objects within that set are already very similar; there is no further refinement in this case.

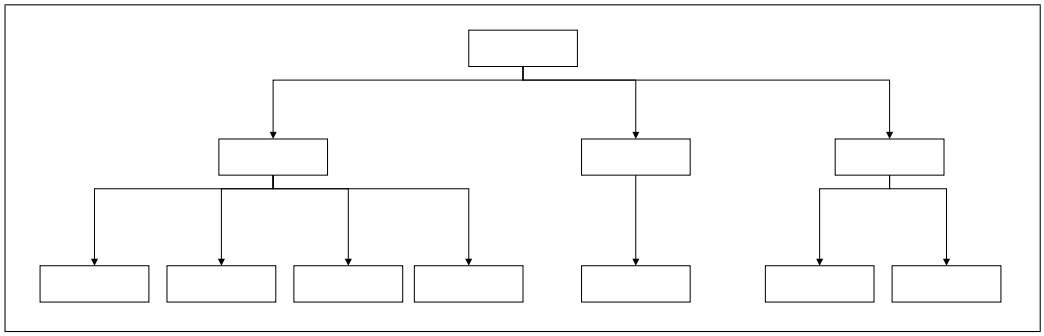

Figure 2.2.: Resulting tree from the hybrid clustering approach.

At each level of the tree a different feature is used for the clustering. Figure 2.3 illustrates how the hybrid works in general by describing the pseudo code of the algorithm.

Input parameters:
O: set of objects to cluster
$M[1..n]$: array of similarity matrices.
($M[i]$ contains the similarities with respect to feature i.)

Variables:
$S_1,\ S_2$: clusterings, therefore sets of sets of objects.
(e.g. $S_1 = \{O_1, O_2\}$ contains two sets of objects.)

$ComputeClustering(O, M)$ returns a set of sets of objects
(clustering according to the similarity matrix M).

HybridClustering($O, M[\,]$):
initialize $S_1 = \{O\}$
for (i = 1..n) {
$S_2 = \emptyset$
forall $(O' \in S_1)$ {
$S_2 = S_2 \cup ComputeClustering(O', M[i])$
}
$S_1 = S_2$
}
return S_1

Figure 2.3.: The hybrid clustering algorithm.

The function $ComputeClustering(O, M)$ can be any clustering algorithm which returns a set of sets of objects. As described in section 1.1.2, a hierarchical approach could also be used, since it can be transformed into a non-hierarchical clustering by cutting the resulting tree (cf. the figures 1.7 and 1.8). The process which has to be performed to use a hierarchical clustering is shown in figure 2.4

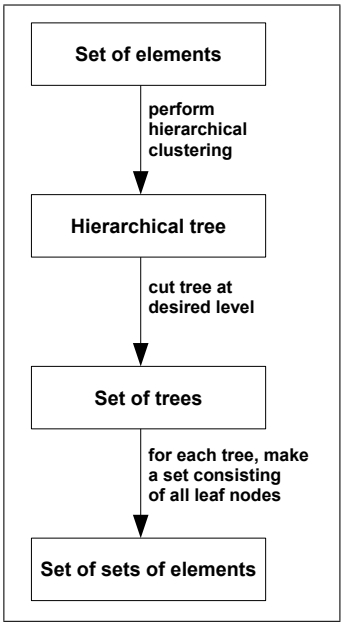

Figure 2.4.: The result of the hierarchical clustering has to be transformed by cutting the tree, leading to a non-hierarchical clustering at the end. This result can then be used for the hybrid approach.

The pseudo code is rather abstract. The described algorithm does not prepare any tree structure, it just clusters all the clusters in iteration i with respect to the similarity matrix belonging to i. This example shows how the different similarity sets, each reflecting one feature, are used. If a hierarchical clustering algorithm is used, each node in figure 2.2 ultimately refers to a tree then (resulting from the cut). For computing the next step, the tree is then reduced to its leaf nodes with the resulting set being clustered according to the desired feature. This in turn establishes the trees, to which the nodes at the next level refer. It has to be noticed that those trees are of course not sub trees of the original tree, since it (the original tree) resulted from a clustering according to the former used feature. The trees at the next level resulted from cutting a hierarchical tree, which was computed according to another feature. It is therefore a transformation of the former tree.

2.2. Underlying clustering algorithm

The underlying clustering algorithm used in this thesis, i.e. the algorithm called by the $ComputeClustering(O, M)$ statement (cf. the hybrid clustering algorithm described in figure 2.3), is a graph algorithm which uses a threshold value. Its similarities can have a different interpretation according to the feature they reflect. E.g. a low *E value* (reflecting the feature "sequence") indicates a high similarity, while for *Pearson Correlation* (reflecting the feature "gene expression") a low value indicates a low similarity. This is abstracted here, so the assumption is always that a low value indicates high similarity.

```
Input parameters:
    E: set of elements to cluster (each initially marked as not visited)
    M: similarity matrix
    t: threshold value

Variables:
    R: a set of clusters
    S: a cluster (set of elements)
    stack: DFS stack

ComputeClustering(E, M, t):
    R = ∅
    forall (e ∈ E) {
        if (e not visited) {
            S = ∅
            initialize stack as empty
            push e on stack and mark as visited
            while (stack not empty) {
                pop e1 from stack
                S = S ∪ e1
                forall (e2 ∈ E) {
                    if ((e2 not visited) ∧ (e1 ≠ e2) ∧ (M[e1, e2] ≤ t)) {
                        push e2 on stack and mark as visited
                    }
                }
            }
            R = R ∪ S
        }
    }
    return R;
```

Figure 2.5.: The underlying graph clustering algorithm.

The algorithm traverses the graph in a depth-first-search manner. It can be observed that the algorithm is not recursive. Instead of this it uses a DFS stack. The algorithm starts with one single node on the stack. For every node (a node represents an object to cluster) which the algorithm currently visits (popped from the stack), it checks the similarities (represented by the similarity matrix) with all neighbour nodes (i.e. all other elements since the graph is complete). If the similarity score is lower than the threshold it pushes the neighbour node on the stack and marks it as visited. When the stack is empty, all the elements which could be reached from the initial node have been marked as visited and thus belong to one cluster. The procedure is repeated with another unvisited initial node while there are unvisited nodes left. The algorithm cuts the edges with a similarity score higher than the threshold and thus the graph is decomposed into disjoint sub graphs.

2.3. Space and run time analysis of the algorithm

Since the set of elements to be clustered can be of a large size, it is necessary to analyse the space and time the algorithm needs with respect to the input size. In the real implementation, the used basic data structures are arrays (a 2-dimensional array for the similarity matrix and a 1-dimensional array for the elements to cluster), hash tables (for the cluster sets) and stacks (for the DFS search, cf. figure 2.5). The used operations are:

- Adding elements for hash tables (adding an element to a cluster)
- Reading an element at aspecified position for arrays (fetching an element, fetching the similarity between two elements)
- Pushing an element on the stack (an element to visit)
- Popping an element from the stack (the element currently visited)

It is known that these operations take constant time, so we take the pseudo code algorithm described in figure 2.3 and figure 2.5 and assume that every statement takes constant time.

2.3.1. Run time analysis

First we analyse the run time of the underlying graph clustering algorithm. The clustering algorithm described in figure 2.5 is decomposed into an inner and outer part.

Outer part:
ComputeClustering(E, M, t):
    $R = \emptyset$
    forall ($e \in E$) {
        if (e not visited) {
            InnerPart(R, e, E, M, t)
        }
    }
    return R;

Inner part:
InnerPart(R, e, E, M, t):

```
S = ∅
initialize stack as empty
push e on stack and mark as visited
while (stack not empty) {
    pop e_1 from stack
    S = S ∪ e_1
    forall (e_2 ∈ E) {
        if ((e_2 not visited) ∧ (e_1 ≠ e_2) ∧ (M[e_1, e_2] ≤ t)) {
            push e_2 on stack and mark as visited
        }
    }
}
R = R ∪ S
```

Assume that the size of the input set E is n, i.e. the number of elements to cluster. In each run of the inner part a number of nodes in the graph is visited. Let k_i be the number of elements visited in the i-th launch of the inner part. All these elements were then pushed on the stack during this process. The *while*-loop is therefore iterated for k_i times. In each iteration of the *while*-loop the *for*-loop iterates over all the n elements. Within the *for*-loop the operations take the constant time c.

The time for the i-th launch of the inner part is then

$$T(''run\, i\, of\, inner\, part'') = k_i \cdot n \cdot c$$

Now we look at the outer part. When the outer part comes the i-th time into the *if*-structure the inner part visits k_i elements, which takes $k_i \cdot n \cdot c$ time. Since k_i elements were visited in one iteration of the outer part, there are $k_i - 1$ more iterations of the outer part where the inner part is not launched (the *if*-clause delivers *false* because they were already visited in the inner part). Thus to every i-th launch of the inner part clustering k_i elements belong $k_i - 1$ launches which take only constant time.

Let l be the number of iterations where the inner part is launched and starts visiting the reachable, unvisited nodes, which then represent one cluster. l is therefore equal to the number of clusters the algorithm will return. Consequently, we can decompose the number of elements n into

$$n = \sum_{i=1}^{l} k_i$$

When $n = |E|$, the number of elements, then the overall run time with respect to the input size n is:

$$T(n) = \sum_{i=1}^{l} [\ \underbrace{(k_i - 1)c_1}_{inner\, part\, not\, launched} + \underbrace{k_i \cdot n \cdot c_2}_{inner\, part\, launched}\ ]$$

$$= \sum_{i=1}^{l} [k_i(c_1 + nc_2) - c_1] = (c_1 + nc_2) \sum_{i=1}^{l} k_i - \sum_{i=1}^{l} c_1$$
$$= (c_1 + nc_2)n - lc_1 \in O(n^2)$$

For the whole hybrid clustering algorithm (cf. figure 2.3) assume we use r features. In each iteration $i = 1..r$ we have a set C_i of l_i disjoint clusters $C_i = \{C_{i,1}, C_{i,2}, ..., C_{i,l_i}\}$ resulting from former clusterings (respectively the whole set containing all elements at the beginning). Let $k_{i,j}$ be the number of elements in $C_{i,j}$ (the j-th cluster of iteration i). Thus for every iteration i

$$\sum_{j=1}^{l_i} k_{i,j} = n$$

meaning that the elements of the sub clusters of the i-th iteration sum up to n since all clusters are disjoint. Since the time for clustering n elements with the graph clustering algorithm $T(n)$ is in $O(n^2)$ and

$$\sum_{i=1}^{l} k_i^2 \leq n^2 \Leftrightarrow \sum_{i=1}^{l} k_i = n$$

we can assume that the overall time for iteration i is in $O(n^2)$, i.e.

$$\sum_{j=1}^{l_i} \underbrace{T(k_{i,j})}_{\in O(k_{i,j}^2)} \leq \underbrace{T(n)}_{\in O(n^2)} \Leftrightarrow \sum_{j=1}^{l_i} k_{i,j} = n$$

meaning that clustering disjoint sub sets C_i of a set C takes less time than clustering the whole set C (since the graph clustering runs in $O(n^2)$ time). Thus the time for the hybrid graph clustering with n elements and r iterations/features is in $O(r \cdot n^2)$.

2.3.2. Space analysis

For the graph clustering algorithm, on the stack there can never be more than n elements, same for the cluster sets R and S. This is by far less than the space the similarity matrices consume, since their space consumption lies in $O(n^2)$ (each pair of elements has a similarity value). Thus the space consumption of the hybrid graph clustering algorithm with n elements and r different features (i.e. r different similarity matrices) lies in $O(r \cdot n^2)$.

2.4. Features for the hybrid clustering algorithm

This section shows how the several features can be used for the hybrid clustering. It was shown in section 2.1 that the algorithm uses similarity matrices. In section 1.2 different features were introduced. The following sections describe how the data was transformed to gain similarity matrices which can be used by the hybrid clustering algorithm.

2.4.1. Sequences

For the sequences *Blast* was used to calculate *E values*. In general *Blast* searches for a query sequence within a data base. The *E value* of the query sequence q and the matching sequence m indicates the expected number of other sequences in the data base having a higher similarity to q than m. Thus a low *E value* close to 0 indicates that the two sequences are very similar. If a set of n elements should be clustered according to the sequences, the n sequences of the elements are taken as the data base and each sequence is taken once as query sequence. The output *Blast* computes is a set of tuples (s_1, s_2, e). s_1 is the identifier of the query sequence q, s_2 is the identifier of the matching sequence m and e is the *E value* of q and m. This output can directly be transformed to a matrix and used by the hybrid clustering algorithm. Table 2.1 illustrates a *Blast* output.

Query ID	**Sequence ID**	**E value**
AT1G22190	AT1G22190	0.0
AT1G22190	AT4G28140	2e-017
AT1G22190	AT5G44210	3e-013
AT1G22190	AT4G23750	3e-007
AT1G22190	AT5G53290	4e-006
AT1G22190	AT5G11590	4e-006
AT1G22190	AT2G46310	2e-005
AT1G22190	AT1G64380	7e-005
AT1G22190	AT4G25480	0.004
AT1G22190	AT2G44940	0.004
AT1G22190	AT4G32800	0.016

Table 2.1.: A *Blast* output.

2.4.2. Gene expression

Gene expression values result from *Microarray* experiments. For each target we get a set of gene expressions at several time points. Table 2.2 illustrates how the raw data looks like. The pairwise similarities of two elements were computed as the *Pearson correlation* of their gene expression data rows. This was done by a *PHP* script.

ID	GE(0h)	GE(0.5h)	GE(1h)	GE(2h)
AT2G23340	0.988	0.891	0.877	1.0275
AT4G17490	0.964	1.21	3.187	10.1085
AT4G17500	0.974	0.228	0.532	0.459
AT4G36920	0.999	0.69	1.018	0.919
AT4G36900	1	1.286	1.011	1.0055
AT5G25190	0.979	0.969	1.557	1.5395
AT5G61590	0.999	0.778	0.797	0.5805
AT5G53290	1	1.058	1.09	2.942
AT5G51990	1	1.048	1.249	20.6245
AT5G51190	0.998	1.05	1.526	1.7785

Table 2.2.: Raw gene expression data which has to be transformed to pairwise similarities.

2.4.3. Motifs

For a set of motifs, the number of occurences of each motif in the promoter region of a sequence was computed. For a set of sequences this leads to a matrix similar to the gene expression matrix shown in table 2.2. But in this case for each column we do not get a gene expression at the corresponding time point but the number of occurences of the corresponding motif. For each gene or protein sequence we get a data row not of gene expression at different time point but of occurences of different motifs, illustrated by table 2.3. Like the gene expression data, pairwise similarities were computed as the *Pearson correlation* of their motif data rows.

ID	#Occ.(M1)	#Occ.(M2)	#Occ.(M3)	#Occ.(M4)
AT1G22190	9	0	0	0
AT1G28370	8	0	0	0
AT1G64380	10	0	0	0
AT1G68550	12	0	0	0
AT2G23340	4	0	0	0
AT2G28550	6	0	4	2
AT2G35700	13	0	0	2
AT2G39250	10	0	0	0
AT2G40350	13	0	0	0
AT2G44940	15	0	2	0

Table 2.3.: Raw motif data which has to be transformed to pairwise similarities.

3. Software

This part discusses in detail the software which was written for the purpose of this thesis. In the first section the main ideas are introduced, providing solutions for the requirements. Section two explains the architecture of the software. It describes the software on a large scale, in order to answer questions as "How should the software be decomposed into small components, to keep it clearly and reasonably arranged?", or "How can it be accomplished that the software can easily be modified in the future?". More detailed information about the system architecture is given in the technical documentation, which is part of the appendix.

3.1. Solutions to the requirements

This section describes the ideas, which mainly result from the user requirements (UR). The solutions are therefore in the same order as the order of the requirements given in section 1.3.3.

UR1

Three different kinds of data should be distinguished: Raw data, data containing similarities and data containing clusterings.

Solution: All the data which belongs to an experiment can be bundled in a so-called *project*. A project in turn contains three folders for raw data, similarity sets and clusterings belonging to the objects which are to be analysed in the experiment, e.g. the *AP2/EREBP* transcription factor family. All the kinds of data are treated as very abstract, for instance raw data can be a DNA sequence as well as gene expression data which belong to a transcription factor.

UR2

Different views to visualise the data sets should be available.

Solution: If the user selects a set in the browser, the framework collects all the classes from the plug-ins which provides displays for this data set and adds the displays in the tab page on the right part of the main window. There are different plug-ins which provide several views for similarity sets and cluster sets: matrix views and info views for the similarity sets, tree views, measure views and info views for the cluster sets.

UR3

It should be possible to import data sets from foreign software.

Solution: When the user wants to import a set, the framework displays a file browser where he can select the desired file. The framework checks all the installed plug-ins for possible importers. When more than one importer accepting the file is found, a list of the conflicting importers is shown, and the user has to select one. After that, the framework launches the selected importer, and adds the imported set to the right folder in the browser.

UR4

An export function should make the computed data accessible for foreign software.

Solution: One of the plug-ins provides a view where the user can cut a clustering tree from the hybrid clustering. When the user has cut the tree, he can print the delivered clustering to a file. The format is similar to the format of the files that *HCE* creates.

UR5

Tools for analysing the data should be included in the software.

Solution: One of the plug-ins provides a display which computes four measures of a clustering which resulted from cutting a cluster tree. It measures the average Intra-Cluster distance, the Inter-Cluster distance, the *Jagota* measure and the *Davies-Bouldin* measure. Furthermore, another program was written which can take the prints of a clustering mentioned before as well as clusterings from *HCE*, and compare them by calculating the *Rand Index* and the *Adjusted Rand Index*. The measures are described in detail in section 4.2.

UR6

Different users working on the same computer should have their own configuration and data home directory to work on.

Solution: The framework has a system which is similar to the workspace system of *Eclipse*. This is a software framework for software development. The workspace points to a certain directory on the file system, defined by the user. This workspace directory contains all the data the user works on. He can just switch the workspace if he wants to. A user can therefore also have more than one workspace, and different users can have different workspaces. The system recognizes also the current system user, so if different users work on the same computer and have different windows accounts, the framework will always display the matching workspace for the current user.

UR7

The hybrid clustering algorithm should be integrated in the software.

Solution: The hybrid clustering algorithm is included in a plug-in bundle which also defines the classes for clusters and displays for this class. These displays are basically the clustering tree and a matrix which visualises elements in the same cluster, but they also include tools for cutting the clustering tree, plotting the result and computing several measures.

UR8

A wizard should help the user configuring and launching the algorithm.

Solution: The wizard for the hybrid clustering algorithm is included in the same bundle as the algorithm itself. With this wizard, the user can select the elements to cluster and the number of iterations. For each iteration, he can select the desired similarity set for the computation, and a threshold value.

UR9

All the data the user currently works on should be listed in a browser-like view.

Solution: There is a browser view in the application which shows all the current projects the user works on, including the contained sets in the corresponding folder.

UR10

A tutorial should be part of the software.

Solution: The tutorial can be opened via the main menu bar, or by clicking on the shortcut item on the tool bar. It displays the names of the available tutorial pages in a list. When clicking on an item, it displays the corresponding tutorial page. This can be a HTML page, a PDF file, an image etc.

3.2. Software architecture

This section describes the structure of the software at a large scale. Two main patterns within software engineering where used, the so-called *Model-View-Controller* (*MVC*) pattern, which is an architectural pattern, and the *Observer* pattern, which belongs to the class of design patterns. These patterns are discussed in the first two parts. The software is a framework providing basic functionality. It is able to adapt to various tasks related to clustering. The specific functionality (functionality concerning the hybrid clustering) is implemented as plug-ins extending the framework. The third part illustrates how the framework integrates the plug-ins.

3.2.1. System architecture

The architectural design is a variation of the *MVC* pattern. This is used especially to make it easier to exchange the graphical user interface (GUI). The idea of *MVC* is to separate the view from the model and the controller so that the model is not dependent on view and controller. In the selected programming language *C#* the technique of partial classes assists the separation of view an controller. A disadvantage is that the view and the controller e.g. of a formula are grouped together by the developing environment and cannot be distributed in different folders. Therefore a folder called "ui" groups the classes with the view and the controller part together. Figure 3.1 shows the dependencies and the access between the three parts of *MVC*. The classes of the model are collected in the folder "model". There are also other parts of the system located outside the *MVC* logic. A more detailed description of the architecture is given in the technical documentation, which is part of the appendix.

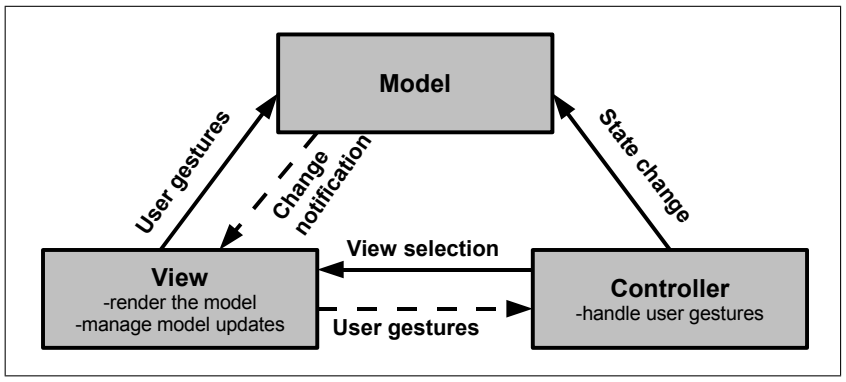

Figure 3.1.: Model-View-Controller

3.2.2. Design patterns

The major design pattern is the *Observer* pattern. It is used to separate the model from the view. This is reasonable, because if they are not separated, every change on the representation (view) would then imply a change of the model as well. This is usually not desired. The pattern therefore becomes important whenever the model changes. It is often used in combination with the *MVC* architectural pattern. As depicted by figure 3.1 if the model changes it does not directly update the view. In this case the model would be dependent on the view. Instead of this, it sends an abstract change notification which is independent from the concrete implementation of the view. The concrete view takes the information given by the model and updates itself. The minor design pattern used in the software is called the *Singleton* pattern. It is especially useful whenever only one singleton instance of a class is desired and it should be prohibited to have multiple instances of the same class. For the detailed description of the patterns (including the concerned parts of the software), refer to the technical documentation, which is part of the appendix.

3.2.3. Framework concept

The application is designed as a software framework providing the basic functionality and the interfaces to extend it. These interfaces are called *extension points*. All the concrete models for the sets, the displays, algorithms, their configuration wizards, set importers and the file system managers for the sets are provided by the plug-ins as *extensions*. A complete overview of the extension points is given by figure C.1, which is part of the technical documentation. Figure 3.2 illustrates how plug-ins extend the framework.

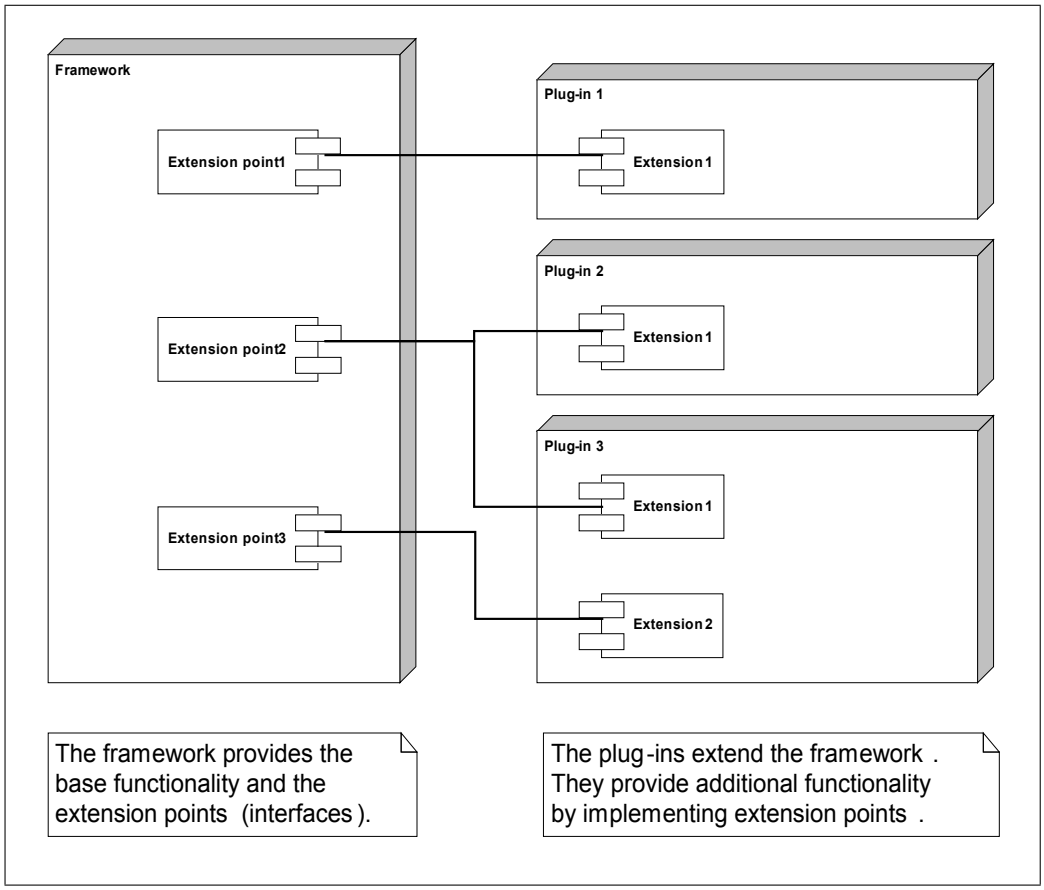

Figure 3.2.: Framework concept

4. Evaluation

This chapter describes the evaluation of the software respectively the hybrid clustering algorithm. The first section gives information about the data which was used. It describes the biological background and specifics, as well as information about the origin of the data. Section two and three discuss formal and informal methods used for the evaluation of the clusterings. This includes measures for analysing the quality of a clustering, measures for comparing two clusterings (which may be the result of two different algorithms) and the informal methods of inspecting the clusterings. The fourth section compares the hybrid clustering approach with two common approaches focusing only on one aspect, namely *Clustal* and *HCE*, as well as with another approach focusing on various features. It is an information fusion approach described by Kasturi et al. [14]. The results are presented in the last section.

4.1. Data sets

This section describes the data sets used for the evaluation. Four groups of data sets were used. They differed in size in order to make facilitate debugging, observe biological specifics and analyse the quality of the clusterings.

4.1.1. WRKY transcription factors

This is a family of transcription factors which have according to Wu et al. [24] a significant role in responses to biotic and abiotic stress. An overview on the family is given by Zhang et al. [28] and Eulgem et al. [7]. More information about the family and its members can be found at `http://www2.mpiz-koeln.mpg.de/~somssich/`. The data group was mainly used to test the application, especially to find semantical errorsduring the system tests. The group is rather small, containing just 26 transcription factors. There is only one similarity set for this group with *Pearson correlations* between gene expression profiles.

4.1.2. AP2/EREBP transcription factors

AP2/EREBP refers to a family of TFs (transcription factors). These TFs have an important role, among others in *Arabidopsis*. This is a family of small flowering plants related to mustard and cabbage. Within biology, this family is very important, since a member of this genus is thale cress, known as *Arabidopsis thaliana*. According to the information given by arabidopsis.org [1] it is widely used as a model organism in plant biology. As

stated by the on-line article from wikipedia.org [22], *Arabidopsis thaliana* was the first plant of which the entire genome was sequenced. There is plenty of information available for this organism, e.g. at http://www.arabidopsis.org/. As mentioned in [1] the genome sequence was completed in 2000. The site also contains a couple of more references to articles related to Arabidopsis, to which the interested reader can refer.

The data used for the evaluation contains data from the *AP2/EREBP* family. Due to this TF family a process is initiated which helps the plant to resist abiotic stress (i.e. environmental stress, not caused by living creatures, like wind, temperature, etc.). Especially three TFs are interesting, namely, *AT4G25470*, *AT4G25480* and *AT4G25490*. These names are the locus identifiers for which http://www.arabidopsis.org can be browsed. Other names for them are (in this order) *CBF2*, *CBF3* and *CBF1* as well as (also in this order) *DREB1C*, *DREB1A* and *DREB1B*. As stated by Kizis et al. [17] they are induced by cold or water stress and their transcripts accumulate at high levels shortly after stress treatment. Furthermore Gilmour et al. [10] come to the conclusion that they are in fact functionally identical.

Three data sets containing similarities between the TFs were used. The first set contains sequence similarities computed by *Blast*. The cut-off value was 10.0 which is the standard value *Blast* uses. All similarities which score a higher *E value* than 10.0 are discarded. How the clustering algorithm handles the missing values is described in the technical documentation (see section C.5.3). The second set was computed from gene expression data from a microarray experiment. The raw set contains a series of gene expressions at several time points. To calculate the similarity, the Pearson correlation was used. The third set was computed from motif data. A set of motifs was taken from the *PlantCare* and the *PLACE* data bases. The number of occurences of each motif was computed leading to a series of motif occurences on a sequence, as described in section 2.4 (table 2.3). As for the gene expression, the *Pearson correlation* was computed to obtain pairwise similarities. The data sets contain similarities between a set of 34 TFs. In appendix C a list of the elements is shown, more precisely a table which also includes their gene expression raw data, from which the *Pearson correlation* was computed (figure B.1).

4.1.3. Indica

Oryza sativa cv Indica is a cultivar of rice. It is quite popular in experimental studies of plant genetics. Here, it is often used together with Arabidopsis. Wu et al. [24] analysed the role of the *WRKY* family of TFs in rice and *Arabidopsis*. The data was primarily used to analyse the clusterings using mathematical measures and contains 1301 genes. Like the data from the *WRKY* group, the data comes from a cold stress study in *Indica*. It was produced from an in-house experiment at Göteborg University. In the experiment, rice seedlings (*Oryza sativa*, *cv Indica*, *v Jumla Marchi*) were

grown under controlled conditions having 25°C day/20°C night temperatures, with 14h-light (intensities of 250μmol m2 s-1)/10h dark cycle. 15 days after germination, seedlings were exposed to cold stress at 4°C. Leaf tissues were harvested after 30min, 2h, 4h, 8h and 24h stress treatment. Control plants were harvested at the same day as the stressed plants. Two biological replicates for each time point, i.e. two biological samples were harvested for each cold stress time point.

Two kinds of similarity sets where used, each reflecting a different feature. The first one contains gene expression similarities. These similarities where generated by computing the Pearson correlation of gene expression raw data from a cold stress study. The second set contains sequence similarities from Blast. A lower cut-off value as for the *AB2/EREBP* group was chosen to save space, namely *e-10*. Every *E value* higher than this value is discarded. For this set this was quite important, because the size was very large. On the one hand, the cut-off value therefore avoids an excessive waste of memory. On the other hand this leads to missing similarity scores. How the clustering algorithm solutes this problem is shown in the appendix C.5.3.

Just to briefly outline how much more efficient it is to use a cut-off process for the computed similarity set, one must compare the different sizes of the sets on the hard drive. Without a cut-off on the gene expression similarities the set size was 12,132,227 Bytes, approximately 12 Megabytes. In contrast the size of the similarity set computed by *Blast* using of the cut-off value e-10 was 165,880 Bytes. This is approximately 1.37% of the size of the gene expression set. Figure 4.1 shows that a massive number of similarities were discarded leading to a much smaller size. On the other hand this leads to problems when calculating measures as will be shown in section 4.5, because no "distance" value can be retrieved for the affected pairs.

For the gene expression normalized expression values were derived using the GC-RMA algorithm as described by Wu et al. [25] implemented in *GeneSpring* version 7.3. Statistically significant differentially expressed genes were detected using a 3-fold change in combination with a FDR p-value <0.05 (as given by Benjamini et al. [2]). There is no published paper at hand yet, the writing of the report is still in progress. The *Pearson correlation* of the gene expression was computed using a cut-off value of 0.7 in order to save space.

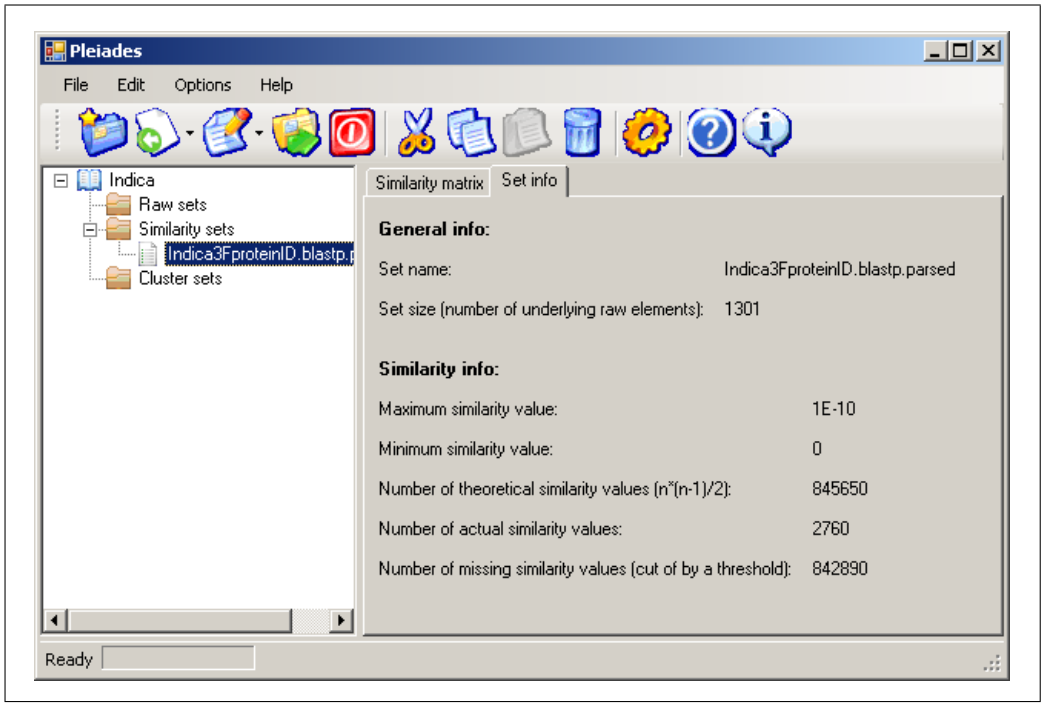

Figure 4.1.: A screenshot of the application. It shows the similarity set imported from a *Blast* result with a cut-off value of e-10.

4.1.4. AtGenExpress

This is a cold stress experiment performed by D'Angelo et al. [4]. The group contains about 2000 genes from different families. The relevant data for the experiment is available at `http://arabidopsis.org`. According to [4], plant material from 18 days old *Arabidopsis thaliana* plants was analysed. Shoot and root tissue were analysed separately. For the gene expression the material was harvested at the time points 0.5h, 1h, 3h, 6h, 12h, 24h. For each time point two samples serving as replicates were collected. This was done by the researchers involved in the experiment. Furthermore the data group contains sequence data and motif data. The sequence similarities were computed by *Blast* with the cut-off value 10.0, leading to a large set used to determine if the algorithm can handle sets with many elements efficiently (with respect to space and time). The motif occurences were computed in the same way as for the *AP2/EREBP* group, leading to a matrix with data rows as described in section 2.4.3 (cf. table 2.3). Both motif similarities and gene expression similarities were computed as the *Pearson correlation*.

4.2. Formal methods

This section describes formal methods which where used for evaluating the application. The first part describes measures which where used to calculate similarities between clusterings, while measures in the second part indicate

how similar elements in the same cluster respectively how dissimilar elements in different clusters are.

4.2.1. Comparing two clusterings

These indices are measures for the similarity of two clusterings. The two clusterings should both be sets of disjoint subsets, forming equal unions. The clusterings are clusterings of the same set of elements S with the size $n = |S|$. The *Rand Index* of two clusterings C_1 and C_2 is then, according to Yeung et al. [26], [27] defined as the value

$$R(C_1, C_2) = \frac{a+b}{a+b+c+d} = \frac{a+b}{\binom{n}{2}}$$

a is here the number of pairs of elements of S which belong to the same cluster in C_1 and in C_2. b is the number of pairs of elements of S which belong to different clusters in both C_1 and C_2. The sum of these values if called the number of "agreements". c and d are the numbers of pairs which are, for c, in the same cluster in C_1, but in a different cluster in C_2. For d is is the other way around. The sum $c + d$ is therefore called the number of "disagreements".

The *Rand Index* is then the number of "agreement pairs", in relation to all possible pairs. In information retrieval there is a very similar value called "accuracy". It is the sum of correctly retrieved documents and correctly not retrieved documents (i.e. the number of documents, which where not retrieved, and are indeed not relevant) in relation to all documents.

As stated by the authors in [27], the *Rand Index* has the problem that it does not take a constant value on two random partitions. They suggest to use another score called the *Adjusted Rand Index*. This index is defined as the value

$$R_{Adjusted}(C_1, C_2) = \frac{\sum_{i,j}\binom{n_{ij}}{2} - [\sum_i \binom{n_i}{2} \sum_j \binom{n_j}{2}]/\binom{n}{2}}{\frac{1}{2}[\sum_i \binom{n_i}{2} + \sum_j \binom{n_j}{2}] - [\sum_i \binom{n_i}{2} \sum_j \binom{n_j}{2}]/\binom{n}{2}}$$

The value n_i indicates the number of elements in the i-th cluster of C_1, and n_j is the number of elements in the j-th cluster of C_2. n_{ij} is then the number of elements which are both in the i-th cluster of C_1 and the j-th cluster of C_2.

4.2.2. Measuring a cluster

Four measures were used for the clusterings computed by the hybrid clustering algorithm. The *average intra-cluster distance* measures the average distance between elements of the same cluster while the *average inter-cluster distance* measures the average distance between elements in different clusters. Another measure, as stated by Gerber [9] proposed by Jagota, is the value

$$Q = \sum_i^k \frac{1}{|C_i|} \sum_{x \in C_i} d(x, \mu_i)$$

$d(x, \mu_i)$ is here the distance from element x of the cluster C_i to the cluster centroid μ_i of C_i. Since we do not have points in the space here, for which this kind of measure is actually made, the distance from x to the centroid is calculated as the average distance from x to all other elements in the same cluster.

The last measure is called the *Davies-Bouldin Index* defined by Davies et al. [5] as the value

$$DB(C) = \frac{1}{k} \sum_{i=1}^{k} \max\{\frac{\Delta(C_i) + \Delta(C_j)}{\delta(C_i, C_j)}\}$$

$\Delta(C_i)$ is here the *intra-cluster distance* of C_i, and $\delta(C_i, C_j)$ is the *inter-cluster distance* between C_i and C_j. For the computation of the *intra-cluster distance*, the *average intra-cluster distance* was used. For the computation of the *inter-cluster distance* analogue, the *average inter-cluster distance* was used analogously.

It has to be said that the results of these measures have to be taken with caution. The measures are actually designed for a set of objects having a distance in a space with a metric, e.g. the *Euclidean metric*. Here, we just have similarity scores at hand. Those scores have a different character, e.g. compare *Blast E values* from sequences (small value indicates high similarity) to *Pearson correlations* from gene expression (high value indicates high similarity). Furthermore some of the similarity values might have been cut off during the computation. In this case, there is actually no "distance" value for the affected elements at hand. The algorithm which was implemented tries to solute this by introducing penalty terms for missing values, under the assumption that a similarity score which was cut off during computation, was on average twice as bad as the worst similarity score which was not cut off. The reader who is interested in the algorithm can request the code from the author.

4.3. Informal methods

The formal methods introduced in section 4.2 are useful to express the quality of a clustering or the similarity of two clusterings, according to a certain measure. However, in some cases it might be hard either to adapt the measures to the given data set or to interpret the results of the measure. One clustering might, according to a measure, be not as good as another clustering, but can nevertheless point out a conspicuousness clearer than another clustering does, due to the characteristic of the data. In each case it is indispensable that the clusterings are, regardless of the high quality a measure indicates, inspected and interpreted. It is important that this process is performed in a structured way with well-selected tools assisting it.

4.3.1. Inspecting the clustering tree

The application shows the result of a clustering in a tree view, as depicted by figure 4.2. A clustering with many very small clusters or singletons is probably not very useful, though e.g. the *intra-cluster similarity* would indicate a high quality. If there is knowledge about the data at hand, this information should also be used. If it is known that there is a group of elements within this data which is supposed to have a high similarity, then a close look on those elements might also give important information. This can also be of importance when the clustering was performed according to another feature. This means that it is for instance known that five elements of the set are known to have a high sequence similarity but the clustering is performed according to the gene expression similarity.

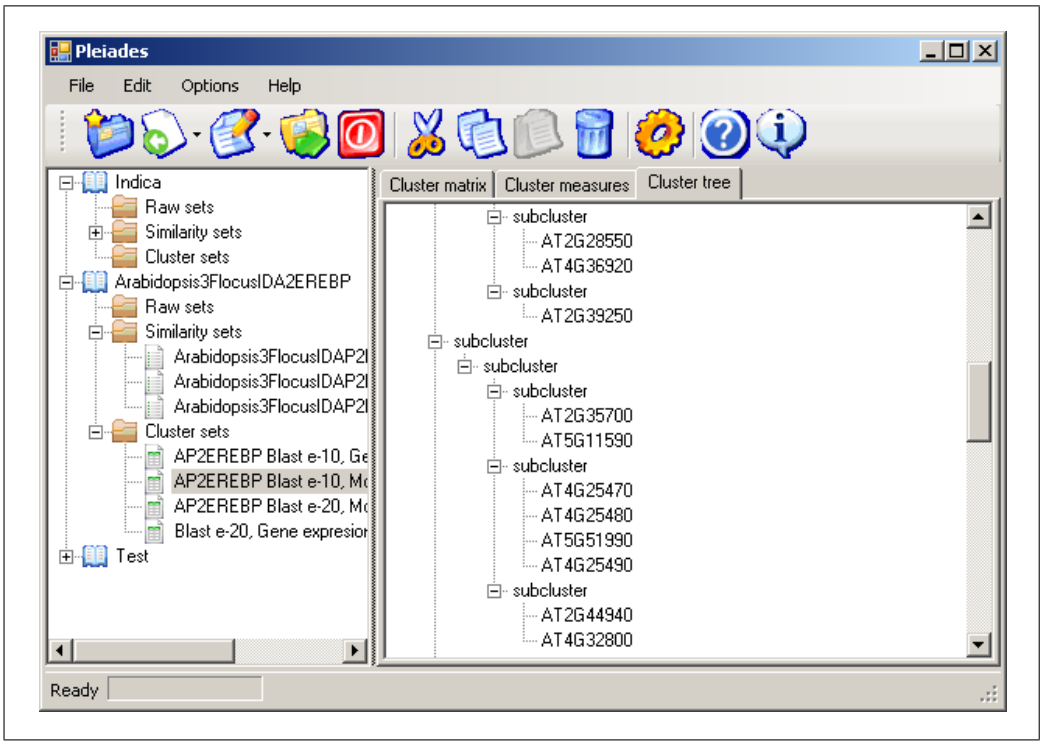

Figure 4.2.: The tree view of a clustering result, computed by the hybrid clustering approach.

4.3.2. Comparing different levels of the clustering tree

It was mentioned above that, when there is knowledge at hand about the relation between some elements according to a certain feature, the clustering according to another feature can provide interesting information. It might be of importance that some elements are very similar in sequence, but not in gene expression or occurence of motifs in the promoter region, especially when the majority of the other elements in the set does not show this behaviour. When knowledge about some elements according to a feature is at hand, it is

quite easy to check whether they show the same behaviour according to another feature or not. On the other hand, when this knowledge is not available it is much harder to determine such a conspicious behaviour of a group of elements. The hybrid approach can assist this process. The following example should illustrate that:

- A group g of elements in a set S has a high similarity in feature A, but is very dissimilar with respect to feature B.
- The set S is now clustered according to feature A in the first round.
- The elements of g should then end up in the same cluster.
- Each of the clusters from the first round is now clustered according to feature B in the second round.
- The elements of g (in which the elements are very dissimilar to each other with respect to feature B) will then be in a different cluster at the next level of the clustering tree.
- If this behaviour is significant for the group, the majority of the other elements does not show this behaviour, i.e. other pairs of elements are either in the same cluster after the first and the second round, or are in different clusters in the first and the second round.

As mentioned in section 1.1.2, a hierarchical clustering can always be transformed into a non-hierarchical one, by cutting the hierarchical clustering tree. We produce now to clusterings from this tree by cutting it at the first and the second level. In the case of the example above, the clusterings would be quite similar, except for the elements in g. Therefore the two clusterings resulting from the cuts at the first and the second level should be inspected and compared. To assist this the application provides a matrix view of non-hierarchical clusterings. Figure 4.3 shows a clustering result from a hybrid clustering with three rounds. The tree was then cut at the second level, resulting in the non-hierarchical clustering shown in the tree on the left. In the matrix view on the right, each row and column indicates one element. If the elements i and j are now in the same cluster, the (i, j) entry of the matrix is black as well as the (j, i) entry and thus the matrix is symmetrical.

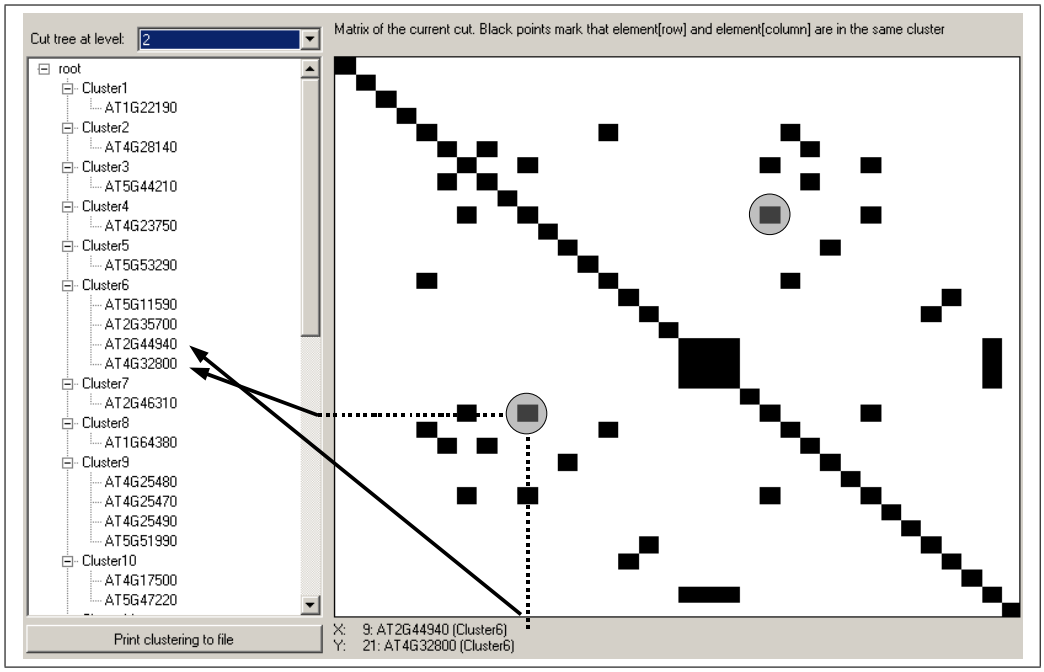

Figure 4.3.: A cut of the clustering tree leading to the non-hierarchical clustering shown on the left and the matrix view shown on the right. The two marked entries in the matrix indicate the entries $(9, 21)$ and $(21, 9)$ The arrows show that for the entry $(9, 21)$ 9 refers to *AT2G44940* and 21 refers to *AT4G32800*. The entry in the matrix is black because both belong to the same cluster.

When there is now a group of elements which shows a significant behaviour, then the majority of the elements should not change (pairs either remain in the same cluster or in different clusters) when the tree is cut at the next level, while the significant elements do change (the pairs of the group are in the same cluster in one clustering, but in different clusters in the other clustering). This is illustrated by the matrix of the cut. When there is a significant group the two matrices should look quite similar, only a few entries which reflect the elements of the significant group should change. Figure 4.4 shows two matrices which resulted from cutting a tree at different levels. This result does not seem to be very significant, since approximately half of the elements remain in the same cluster and half of the elements split up at the next level. However, a look on the affected elements might give important information.

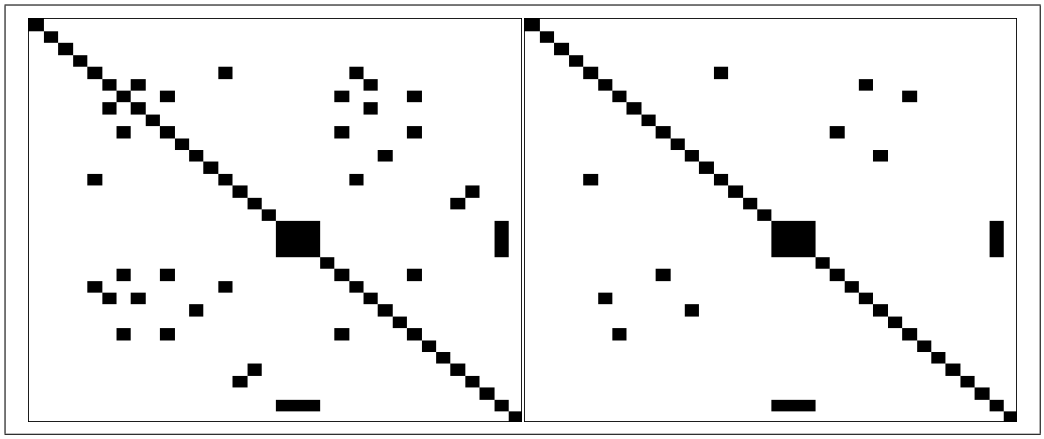

Figure 4.4.: Two matrices which resulted from cutting the clustering tree at different levels.

4.4. Comparison with common clustering approaches

4.4.1. HCE and Clustal

To compare the hybrid clustering with *HCE* and *Clustal* the *AP2/EREBP* group was used. Both of the clusterings focus on raw data. They were chosen because *HCE* clusters with respect to gene expression while *Clustal* uses the sequences. For the clustering with *HCE*, the used similarity measure was the *Pearson correlation* and the linkage method was *average-linkage* (see section 1.1.2 for the definition of *average-linkage*). The resulting hierarchical tree was cut at two different positions leading to two non-hierarchical clusterings with two respectively three clusters. The result is shown in figure 4.5.

To compare *HCE* with the underlying graph clustering algorithm, the hybrid clustering was performed only for one round with the gene expression similarities. This was done for two times, with the thresholds 0.7 respectively 0.8. The result is shown in figure 4.6. It can be observed that the clustering computed using threshold 0.7 contains two clusters while the clustering computed with the threshold 0.8 contains three clusters.

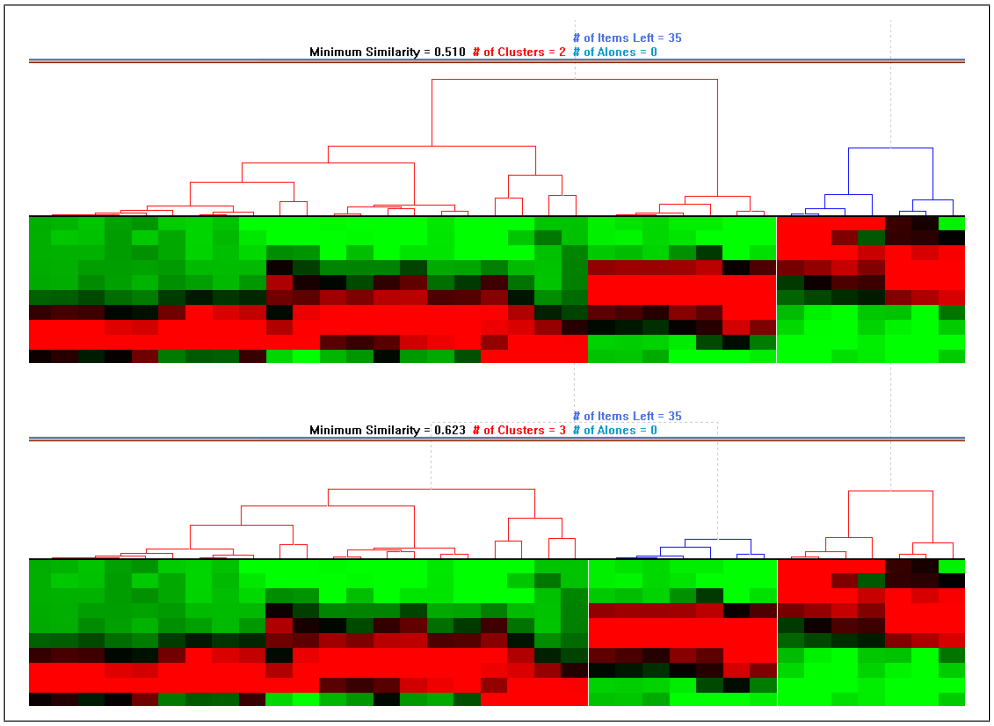

Figure 4.5.: The hierarchical clustering tree computed with *HCE* was cut at two positions leading to two clusterings consisting of two (top) respectively three (bottom) clusters.

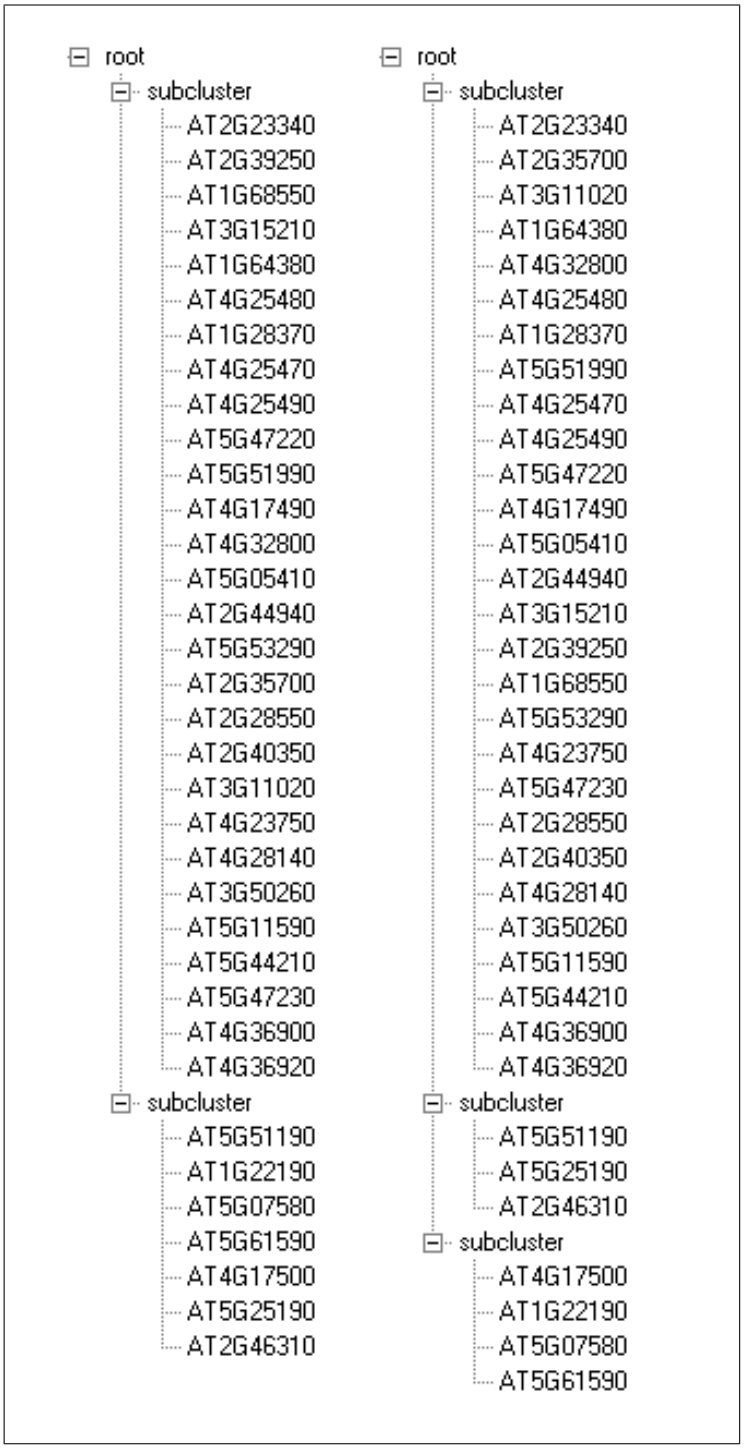

Figure 4.6.: The two clusterings computed by the graph clustering with respect to gene expression similarity using the thresholds 0.7 and 0.8.

The two clusterings from *HCE* as well as the two clusterings from the graph clustering were printed to a file to compare them using the measures introduced in section 4.2. The two clusterings containing two elements were

compared as well as the two clusterings containing three elements. As depicted by figure 4.7 the two clusterings containing two clusters each are identical, indicated by the *Rand Index* and the *Adjusted Rand Index* being 1 (left part of the figure). The clusterings containing three clusters each computed by the graph clustering algorithm and *HCE* differ (right part of the figure). The two clusterings containing three clusters were then compared with the quality measures. Figure 4.8 illustrates that the result from the graph clustering algorithm is slightly better with the *Jagota measure* and the *average inter-cluster distance* is better, while the clustering computed by *HCE* is better with respect to the *Davies-Bouldin Index* and the *average intra-cluster distance* is better.

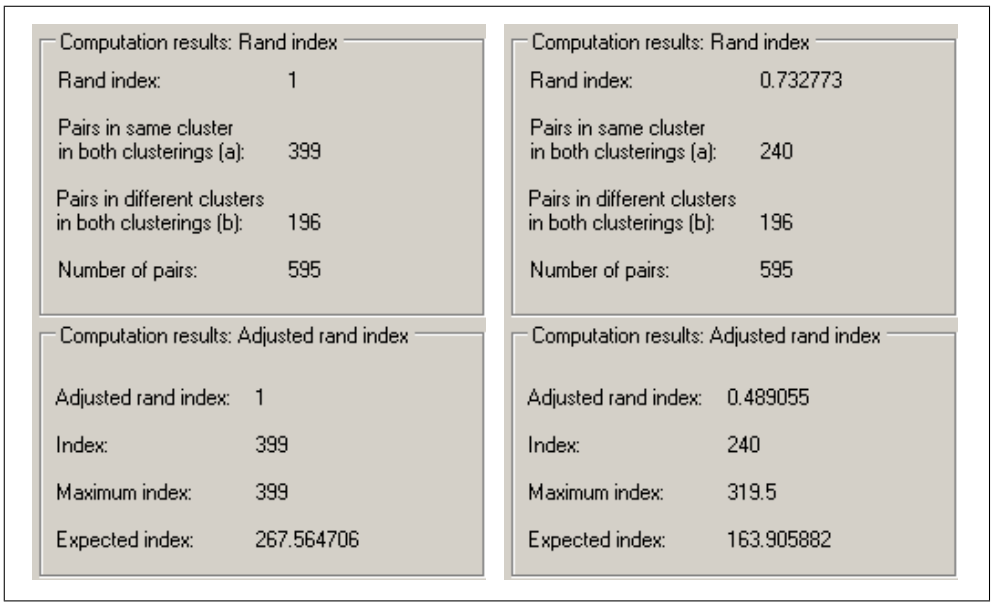

Figure 4.7.: The comparisons with *Rand Index* and *Adjusted Rand Index* of the clusterings computed by the graph clustering algorithm and *HCE*. The comparison of the two clusterings consisting of two clusters each is shown on the left, while the comparison of the clusterings containing three clusters each is shown on the right.

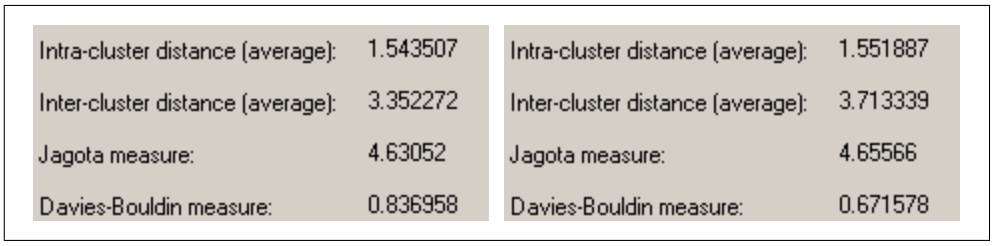

Figure 4.8.: The results from the measure computation for the two clusterings computed by the graph clustering algorithm (left) and *HCE* (right).

Figure 4.7 also illustrates that for dissimilar clusterings the *Rand Index* scores

higher than the *Adjusted Rand Index*. Yeung et al. state that, in a work from Milligan and Cooper from 1986, the authors recommend the *Adjusted Rand Index* as the index of choice [26].

HCE has a lot of functionality which is currently missing in the application from this thesis. First of all, *HCE* can cluster from raw data like gene expression data. The application from this thesis supports that in general due to the framework concept, but the functionality (i.e. adequate plug-ins) is still missing. Furthermore, *HCE* has an efficient way to cut clustering trees by using a cut bar and can also compute *k-means* clusterings, as illustrated by figure 4.9. The cut bar can be dragged up and down with the mouse, and the result of the cut is dynamically displayed by showing the sub trees resulting from the cut. The *k-means clustering* can both cluster columns and rows of the data set, in this case it was gene expression data.

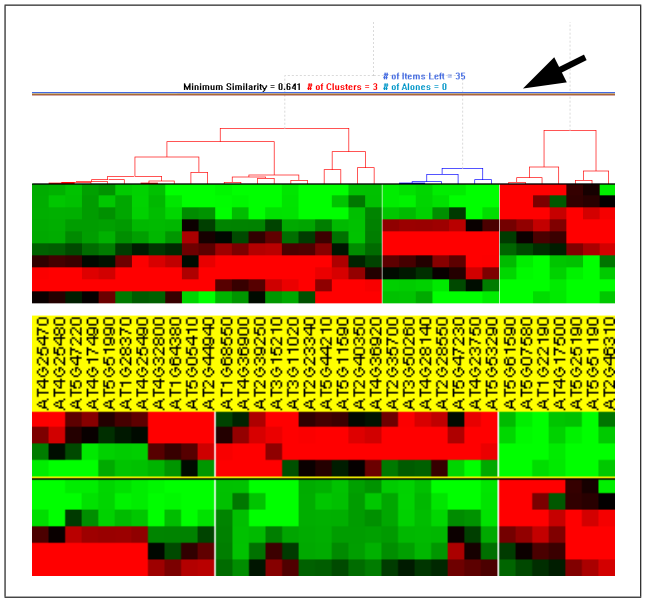

Figure 4.9.: *HCE* provides a lot of functionality. A cut bar (top) helps to efficiently cut hierarchical clustering trees. The software can also compute *k-means* clusterings (bottom).

To illustrate the benefit of the hybrid clustering approach figure 4.10 shows the clustering trees of *Clustal* and *HCE* together. *Clustal* clusters according to the sequences while *HCE* uses the gene expression for the computation. It can be observed that according to gene expression there are seven elements being rather similar. Four of those elements are highlighted in the figure, namely *ATG25490*, *AT5G51990*, *ATG25480* and *ATG25470*. If we only look at the sequences as computed by *Clustal*, only *ATG25490*, *ATG25480* and *ATG25470* are rather similar. If we only look at the gene expression we can assume that *ATG25490*, *AT5G51990*, *ATG25480* and *ATG25470* and three additional elements are rather similar. The hybrid clustering now shows that *ATG25490*, *AT5G51990*, *ATG25480* and *ATG25470* are rather similar

compared to the other elements with respect to both gene expression and sequences. If we look at both features they differ from the other elements, as depicted by figure 4.11. How the hybrid clustering is configured is discussed in the latter sections when the results are presented.

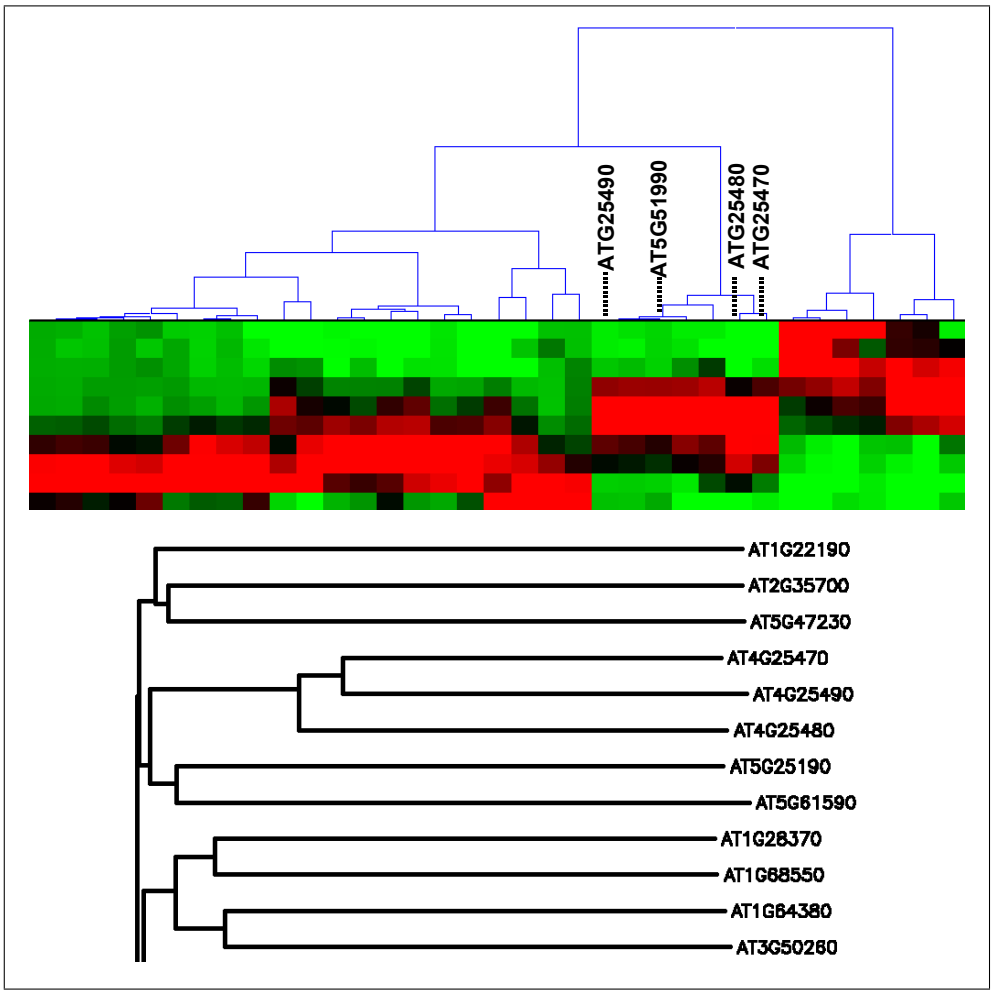

Figure 4.10.: Two clusterings computed by *HCE* (gene expression, top) and *Clustal* (sequences, bottom).

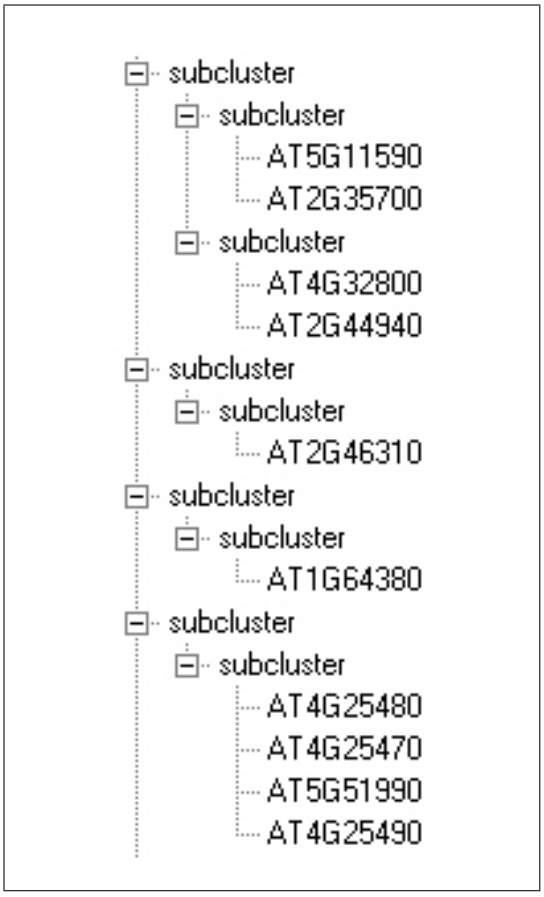

Figure 4.11.: The hybrid clustering shows that *ATG25490*, *AT5G51990*, *ATG25480* and *ATG25470* are rather similar according to both gene expression and sequences. They remain in the same cluster at the second level of the clustering tree.

4.4.2. Information fusion approach

Another approach also using multiple features of the data is described by Kasturi et al. [14]. The main difference between the approach of this thesis and the approach in [14] is that the approach described by Kasturi et al. focuses on all the aspects at the same time, while the hybrid clustering focuses on one aspect at each iteration step. The algorithm which Kasturi et al. describe is an information fusion approach. Their method extends a *self-organizing map* (SOM). This is an *artificial neural network*. It is trained using unsupervised learning. Further information about self-organizing maps is for instance given by Haykin [11], while Dayan gives an overview on unsupervised learning [6]. For the information fusion approach the authors of [14] weight the information which is given by the different aspects. Among other things, they also use gene expression data and motif data. Assume that a set of genes $g_1, g_2, ..., g_n$ is given. Furthermore for each gene there is data of k categories at hand. The algorithm then iterates until convergence. In each iteration, one gene g and one category r is randomly selected. The distance

from g to each cluster centre according to the category r (e.g. gene expression) is calculated. g is assigned to the closest cluster and after that, the weights of this cluster are updated using a learning rule.

They use two different distance functions for the gene expression and the motifs. For the gene expression, they use the *Relative entropy* or *Kullback-Leibler divergence* which is according to the authors [14] given by

$$D(p\|q) = \sum_{x \in X} p(x) log_2 \frac{p(x)}{q(x)}$$

The authors state that it was shown by Kasturi et al. [15] that this measure performs better on gene expression than the standard distance measures such as *Pearson correlation*. For the motifs, they use the *Extended Jaccard Similarity Coefficient* which is according to the authors given by

$$SDist(x, y) = 1 - \frac{\sum_{i=1}^{N} min(x_i, y_i)}{\sum_{i=1}^{N} max(xi, yi)}$$

After analysing the results the authors come to the conclusion that "genes with similar function might possibly share a common expression pattern under a certain experimental condition and might share a common motif" [14] - an assumption which supports the use of multiple aspects of the data. As already mentioned, the approach from this thesis and the approach from Kasturi et al. have in common that they both focus on several aspects of the data. As shown in [14] this seems to be reasonable, providing new insights which might not be obtained when focusing only on one aspect. On the other hand, the hybrid approach leads to a hierarchical clustering while the information fusion approach leads to a non-hierarchical clustering. In their experiments, the authors also repeated their clusterings with different input parameters i.e. different weights of the aspects, to get another view on the data. A method which was also used in this thesis, cf. for instance section 4.5.1. As the information fusion approach described in [14] is related to this thesis, it could be of interest to run the hybrid clustering algorithm on the data Kasturi et al. used for their experiments and compare the results. Furthermore the two distance measures the authors use are interesting. Since both of them are used for data which was also used in this work (gene expression and motifs), they could be adapted for the hybrid approach. Even the algorithm itself might be a candidate for being used as base algorithm for a hybrid clustering.

4.5. Results

This section shows the results from the evaluation and is divided into four subsections. The first part shows biological specifics which were observed on the results of the *AP2/EREBP* group. The second part analyses the clusterings. It measures the clusterings which were computed from the larger data group of *Indica*. Furthermore, it compares clusterings from this algorithm with clusterings from *HCE*. In the third part the results from

clustering the data group of *AtGenExpress* are illustrated. It was also analysed if the hybrid clustering could be used for classification. The results of this analysis are described in the last part.

Before the results are presented it should first be described how the algorithm was used. Most of the clusterings which were computed consisted of two or more iterations. In each iteration another similarity set reflecting a specific feature was used, for instance it was first clustered with respect to sequence similarity. Each cluster resulting from this procedure was then taken again and clustered in the second iteration according to another feature, for instance gene expression. As figure 4.12 depicts, it is also necessary to set a threshold for each round. The graph clustering algorithm needs this threshold since it cuts off all the edges representing similarities violating the threshold, as it was illustrated by figure 1.6. The graph, represented by the similarities, is by that decomposed into clusters which are retrieved by a *depth-first-search*. The interpretation of the threshold depends on the character of the similarity set (compare *E values* and *Pearson correlation*) and has to be selected by the user as well.

Selecting good thresholds is a task requiring experience as well as knowledge about the data. If the user does not know about the interpretation of *E values* for instance, it would be really hard to select good thresholds. It is also helpful if the user knows for instance that the elements in the data set are known to be rather similar with respect to the feature of choice. In this case the threshold should be set higher (if higher values indicate higher similarity) to avoid that almost all elements will be in the same cluster. The order how the features should be used is also defined by the user. Usually in this work the sequences were chosen as the first feature for the hybrid clustering. The reason for that is that the sequence is the primary structure of a protein or gene mainly defining function and structure. It is very unlikely that two proteins totally different in sequence have the same function, or that the same transcription factors bind to genes totally different in the nucleotide sequence.

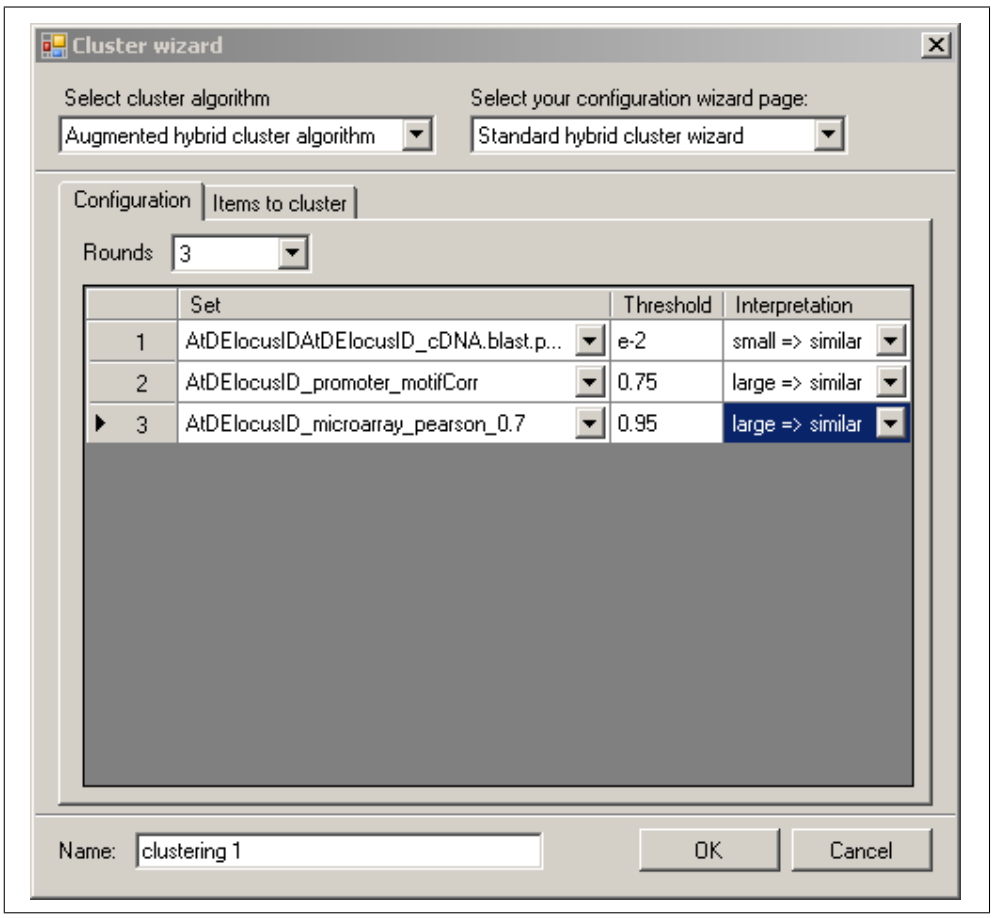

Figure 4.12.: Configuration of a hybrid clustering. For the first round a similarity set computed by *Blast* containing *E values* (sequence similarity) was used. The lower the value, the better the similarity. In contrast, when computing a *Pearson correlation*, a high value (with the upper bound one) indicates a high similarity (cf. the second round). The hybrid clustering is performed with three iterations, with a different similarity set in each iteration.

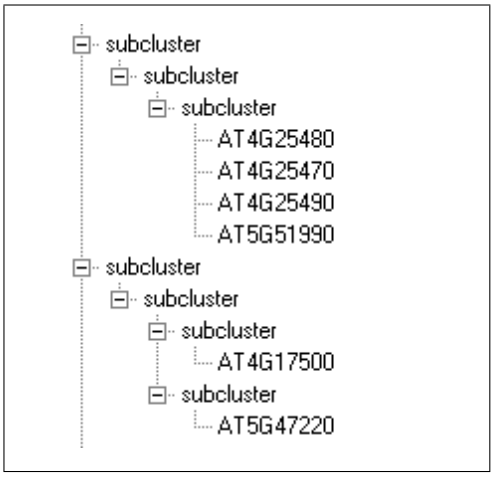

Figure 4.13.: A tree resulting from a hybrid clustering for three rounds. The root (depth 0) represents the set of all elements before the clustering and a leaf node (depth four) represents a single element. The nodes at levels one, two and three represent the clusters of the first, second and third iteration.

4.5.1. Results from clusterings of the AP2/EREBP group

Clusterings of the 34 members of the *AP2/EREBP* were made. For the clusterings, three features were used, namely sequence similarity, gene expression similarity and motif occurence similarity. That means that the hybrid clustering algorithm executed the underlying graph clustering algorithm for three iterations under the use of a different similarity set for each round.

Observation: AT3G11020 and AT5G05410

The first two clusterings belong together. The configuration of the second clustering is a permutation of the configuration of the first clustering, meaning that the features were used in a different order but the thresholds remained the same. In the first computation the set was clustered for three rounds. In the first round, it was clustered according to *Blast E values* with a threshold of *e-10* (maximum threshold). In the second round the motif correlations where used with a threshold of 0.8 (minimum threshold). In the last round, it was clustered with the gene expression similarities and a threshold of 0.9 (minimum threshold).

Figure 4.14 shows a part of the clustering tree resulting from the computation. As figure 4.15 illustrates, when cutting the tree at level two only one more pair of elements is not in the same cluster anymore compared to the cut at level one. The elements are *AT5G05410* and *AT3G11020*. Their motif correlation was thus lower than 0.8, while the motif correlations of all the other pairs in the same cluster was at least 0.8 so they remained in the same cluster at the next level.

In this clustering, it cannot be directly observed if *AT3G11020* and *AT5G05410* would also be rather dissimilar with respect to gene expression since they are already in different clusters at level two. If so, they would be dissimilar with respect to both gene expression correlation and motif occurence correlation. Another clustering was performed, but now in the second round gene expression was used (again with threshold 0.9) while in the third round it was clustered with respect to motif occurence correlation (again threshold 0.8).

The resulting tree was then cut at level two, i.e. at the level of the gene expression clustering. The results from this cut were printed to a file. As can be seeen in the printout in figure 4.16, with respect to gene expression with threshold 0.9 most of the elements are in different clusters, while *AT3G11020* and *AT5G05410* belong to the same cluster, meaning that they have a gene expression correlation higher than 0.9. On the one hand the gene expression similarity of *AT3G11020* and *AT5G05410* is in contrast to most of the other elements rater high. On the other hand their motif occurence similarity is comparatively low, in contrast to the other elements of the data group.

Inspecting the gene expression similarity set shows that the *Pearson correlation* of the gene expression profiles of *AT5G05410* and *AT3G11020* is 0.9231, while inspection of the *Pearson correlation* of motif occurences in *AT5G05410* and *AT3G11020* shows a value of 0.7681, which is much less than most of the other similarities in this set. According to the information found at the *Arabidopsis* site [1], both *AT5G05410* and *AT3G11020* have a role in drought stress resistance, but no comparative analysis of these two elements could be found.

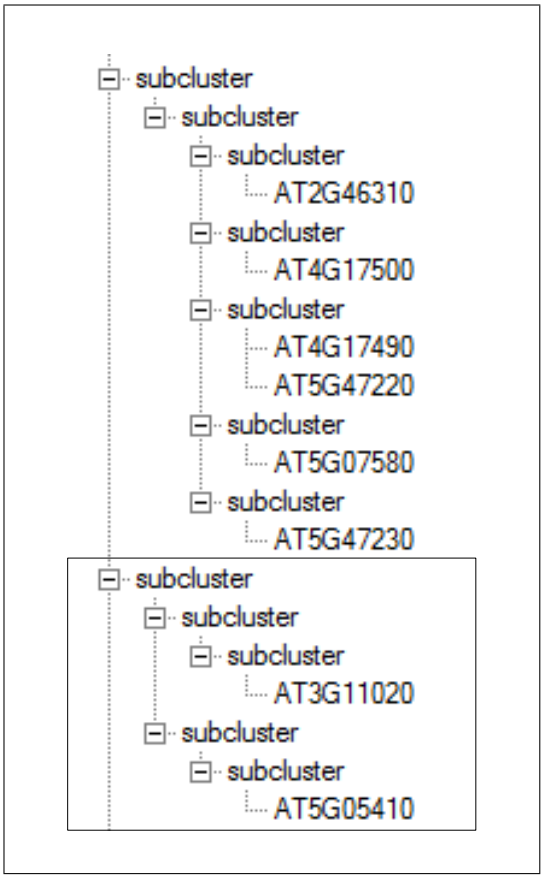

Figure 4.14.: Clustering tree of the first computation. The elements *AT3G11020* and *AT5G05410* are in the same cluster at level one, but not at level two and three.

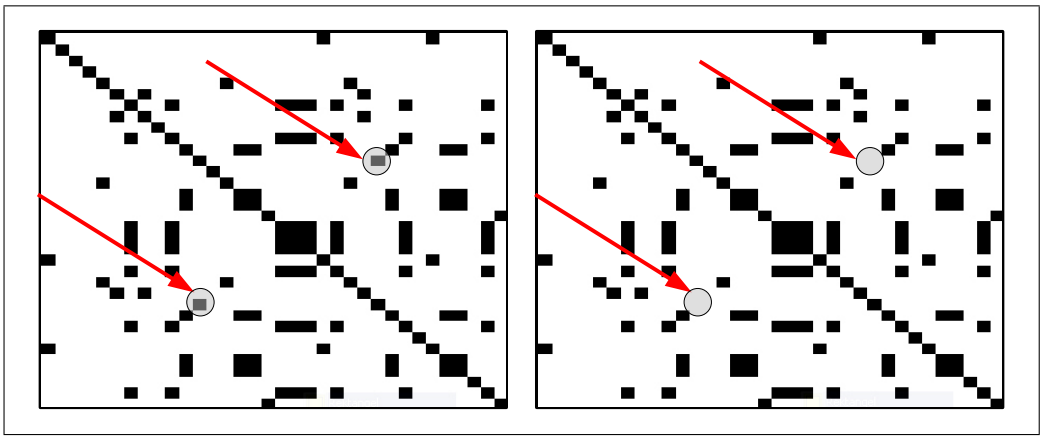

Figure 4.15.: The two matrices resulting from the cut at level one (left) and level two (right). It can be observed that *AT3G11020* and *AT5G05410* belong to the same cluster at level one indicating sequence similarities, but not at level two indicating motif occurence similarities.

CLUSTER1	LEAF0	AT1G22190
CLUSTER2	LEAF1	AT5G44210
CLUSTER2	LEAF2	AT4G28140
CLUSTER3	LEAF3	AT4G23750
CLUSTER4	LEAF4	AT5G53290
CLUSTER5	LEAF5	AT5G11590
CLUSTER5	LEAF6	AT2G35700
CLUSTER6	LEAF7	AT4G32800
CLUSTER6	LEAF8	AT2G44940
CLUSTER7	LEAF9	AT4G25480
CLUSTER7	LEAF10	AT4G25490
CLUSTER7	LEAF11	AT5G51990
CLUSTER7	LEAF12	AT4G25470
CLUSTER8	LEAF13	AT2G46310
CLUSTER9	LEAF14	AT5G47230
CLUSTER10	LEAF15	AT4G17490
CLUSTER10	LEAF16	AT5G47220
CLUSTER11	LEAF17	AT4G17500
CLUSTER12	LEAF18	AT5G07580
CLUSTER13	LEAF19	AT1G64380
CLUSTER14	LEAF20	AT2G23340
CLUSTER14	LEAF21	AT3G50260
CLUSTER15	LEAF22	AT4G36900
CLUSTER16	LEAF23	AT2G28550
CLUSTER16	LEAF24	AT4G36920
CLUSTER17	LEAF25	AT2G39250
CLUSTER18	LEAF26	AT1G28370
CLUSTER19	LEAF27	AT5G51190
CLUSTER20	LEAF28	AT5G05410
CLUSTER20	LEAF29	AT3G11020
CLUSTER21	LEAF30	AT5G25190
CLUSTER22	LEAF31	AT3G15210
CLUSTER23	LEAF32	AT2G40350
CLUSTER24	LEAF33	AT1G68550

Figure 4.16.: Plot created by the clustering software. It resulted from the cut of the tree of the second computation at level two indicating gene expression similarity with threshold 0.9. *AT3G11020* and *AT5G05410* are in the same cluster.

Observation: AT4G25470, AT4G25480, AT4G25490

The former clusterings (see section 4.5.1) were then inspected at the last level to find out if there are elements in the data group rather similar with respect to all three features. Figure 4.17 shows the cut at level three from the second clustering described in the former section 4.5.1 (first round gene expression with threshold *e-10*), second round gene expression with threshold 0.9 and third round motifs with threshold 0.8). It can be observed that there is a cluster consisting of four elements, *AT4G25470*, *AT4G25480*, *AT4G25490* and *AT5G51990*. Especially the first three of those are interesting. They are also known as *CBF1-CBF3* or *DREB1A-DREB1C*. According to Gilmour et al. they have an important role in *Arabidopsis* regarding biotic stress [10].

The authors manipulated *CBF1-3* in plants and set them under cold stress. They observed that the plants with the manipulated *CBF1-3* degenerated, they were smaller and grew much slower [10]. As figure 4.5.1 illustrates most of the clusters contain at level three only one element. It seems to approve what the authors state: *CBF1-3* are very related [10].

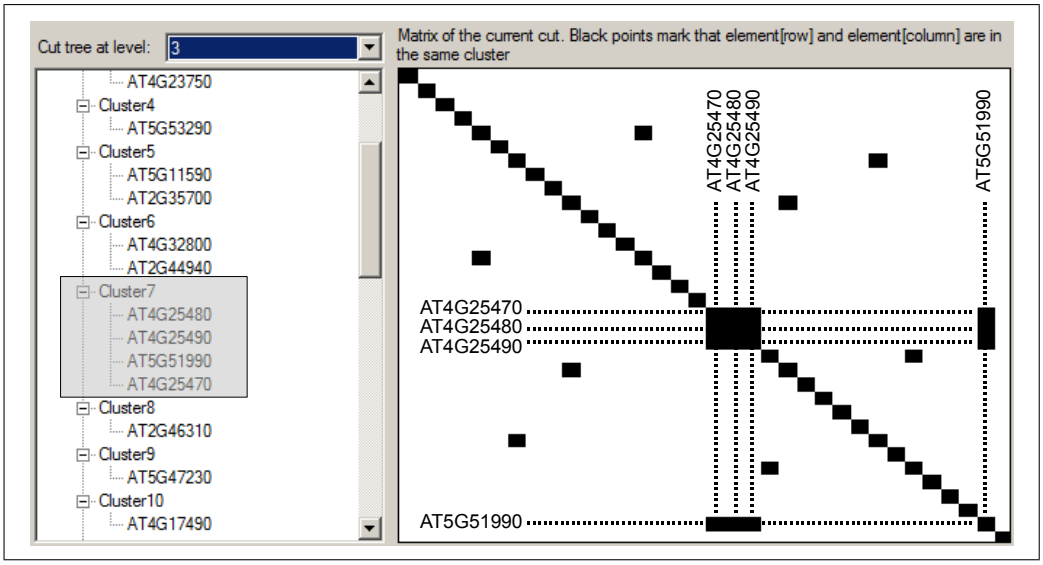

Figure 4.17.: Clustering tree and matrix from the cut of the second clustering described in section 4.5.1 at level three. *AT4G25470*, *AT4G25480*, *AT4G25490* and *AT5G51990* are in the same cluster meaning that they are similar with respect to sequences (*E value* <*e-10*), gene expression (*Pearson correlation* >0.9) and motif occurrence correlation (*Pearson correlation* >0.8).

4.5.2. Results from clusterings of the Indica set

We will now turn to the measures for evaluating cluster quality. The measures refer to distances between objects in a k-dimensional space. Since we have no location at hand, the similarity scores have to be used. Due to cut-off values in the pre-computation of the similarity sets (as for instance used by *Blast*)

some of the similarity scores are missing. A penalty term was introduced for these missing values. Let S be the set of similarity sets which are at hand. The penalty score of two elements for which no similarity value is at hand, is then defined as the value:

$$penalty(e_1, e2) = 2 * \max_{(x,y) \in S} \{distance(x, y)\}$$

The group was firstly clustered according to blast similarities with a threshold of *e-20*. After that, the algorithm clustered according to gene expression similarities with a threshold of 0.8. The resulting clustering tree was then cut at the first and the second level. The purpose was to find out whether the hybrid approach improves the quality of the clustering. If this assumption holds the measure should indicate a higher quality at the second level. Before the results are presented, it has to be noticed that the hybrid approach is not meant to replace common clustering algorithms. Instead of this the hybrid approach is meant to make use of common clustering algorithms (in this case a graph clustering algorithm) by executing them iteratively with a different feature in each iteration. This is not meant to improve the clustering in terms of mathematical quality but to obtain new insights by taking various features into account.

The result is shown in figure 4.18. The clustering consists of 916 clusters while the total number of elements is 1301, meaning that there are a lot of clusters containing only one element. This is due to the *Blast* cut-off value. For a lot of pairs there is no similarity value at hand and thus the graph clustering algorithm treats such pairs as most dissimilar implying that the elements will be in different clusters. The clustering at level two contains 1010 clusters meaning that there are even more clusters containing only one element. Thus it is not surprising that the *average intra-cluster distance* of the second clustering is lower, because for clusters containing only one element the *intra-cluster distance* is 0, and the second clustering has more such clusters. The *average inter-cluster distance* in contrast was nearly the same. The increase of the *Jagota measure* could be explained analogously since this measure only focuses on the *intra-cluster distance*. In the first clustering there are more clusters containing only one element and thus delivering the distance 0 when calculating the distance from the single element to the cluster's centroid - itself.

However, the higher *Davies-Bouldin Index* on the first clustering is surprising. It may be due to the missing values. Since the first clustering contains many small clusters, calculating their *inter-cluster distances* may include a lot of missing values resulting in a high penalty value. This makes the denomiator (cf. the formula for the *Davies-Bouldin Index*) on the average higher. The missing values may lead to a decrease in measure quality. Another reason might be that the measure is more useful to compare clusterings of the same set which have the same number of clusters. To analyse this, a possibility is to

have another clustering approach which delivers the same number of clusters (or at least approximately the same number), but in a manner that the clusters decompose the set in another way than this algorithm. Thus the measures have to be interpreted cautiously if the clustering was computed from a similarity set with missing values, due to a cut-off value in the pre-computation.

Intra-cluster distance (average):	0.287678	Intra-cluster distance (average):	0.119057
Inter-cluster distance (average):	2.168352	Inter-cluster distance (average):	2.188126
Jagota measure:	263.513099	Jagota measure:	120.247212
Davies-Bouldin measure:	2.308511	Davies-Bouldin measure:	1.260617

Figure 4.18.: The result from the measure computation for the cuts at the first (left) and the second level (right). It can be observed that the measures indicate a quality improvement for the cut at the second level although there are more singletons.

To analyse if the measures are more reliable a clustering was performed only for one round with the gene expression similarities and a threshold of 0.9. For pre-computation of the gene expression similarities, no cut-off value was used and thus for every pair there is a similarity value at hand. The result is shown in figure 4.19. The clustering contained only 11 clusters and thus there are not many singletons. Intiutively, such a clustering seems to be more reasonable under mathematical terms (similar elements should be in the same cluster and dissimilar elements should be in different clusters) than a clustering where almost all of the clusters contain only one element (implying that almost all of the elements are in different clusters). The measures approve this assumption, the *Jagota measure* as well as the *Davies-Bouldin Index* indicate that the mathematical quality of this clustering is much better compared to the former clusterings with many singletons.

Intra-cluster distance (average):	0.459014
Inter-cluster distance (average):	2.684998
Jagota measure:	5.049159
Davies-Bouldin measure:	0.756782

Figure 4.19.: The result from the measure computation for second clustering, focusing only on gene expression

4.5.3. Results from clusterings of the AtGenExpress group

The elements of this group were clustered for three iterations. In the first iteration, it was clusterered according to sequence similarities with a threshold of *e-2*. The reason for that is that the data group contains genes from different families. Compared to the members of the *AP2/EREBP* TF family they are less similar and thus a higher threshold was chosen (not that for *E values*, lower values indicate higher similarity). After that in the second iteration the threshold was 0.8 and the motif correlations were used. The motif threshold of 0.8 turned to be a good value in the formerly computed clusterings. In the last iteration the threshold was 0.95 and the gene expression profile correlations were used. The purpose was to have a strong refinement at the last level. As can be observed in figure 4.21 a lot of elements land in one cluster. The other clusters are comparatively small, and thus the total number of clusters is with 1124 rather high. The upper matrix in figure 4.20 illustrates the distribution of the elements. In the second iteration it was clustered according to the motif correlations with a threshold of 0.8. As can be seen in the second matrix in figure 4.20, the situation does not really change, the number of clusters is approximately the same. But the cut at level three reflecting the clustering of the third iteration, according to gene expression with threshold 0.95 shows a change, depicted by the third matrix in figure 4.20. Especially was the large cluster depicted by in figure 4.21 divided into two clusters, as illustrated by figure 4.22, meaning there is a set of elements which all have a pairwise sequence similarity of at least *e-2* as well as a motif correlation of at least 0.8. Clustering them according to gene expression correlation with a threshold of 0.95 divides them into two sub clusters. None of the elements from one sub cluster has a gene expression correlation of at least 0.95 to one element of the other sub cluster.

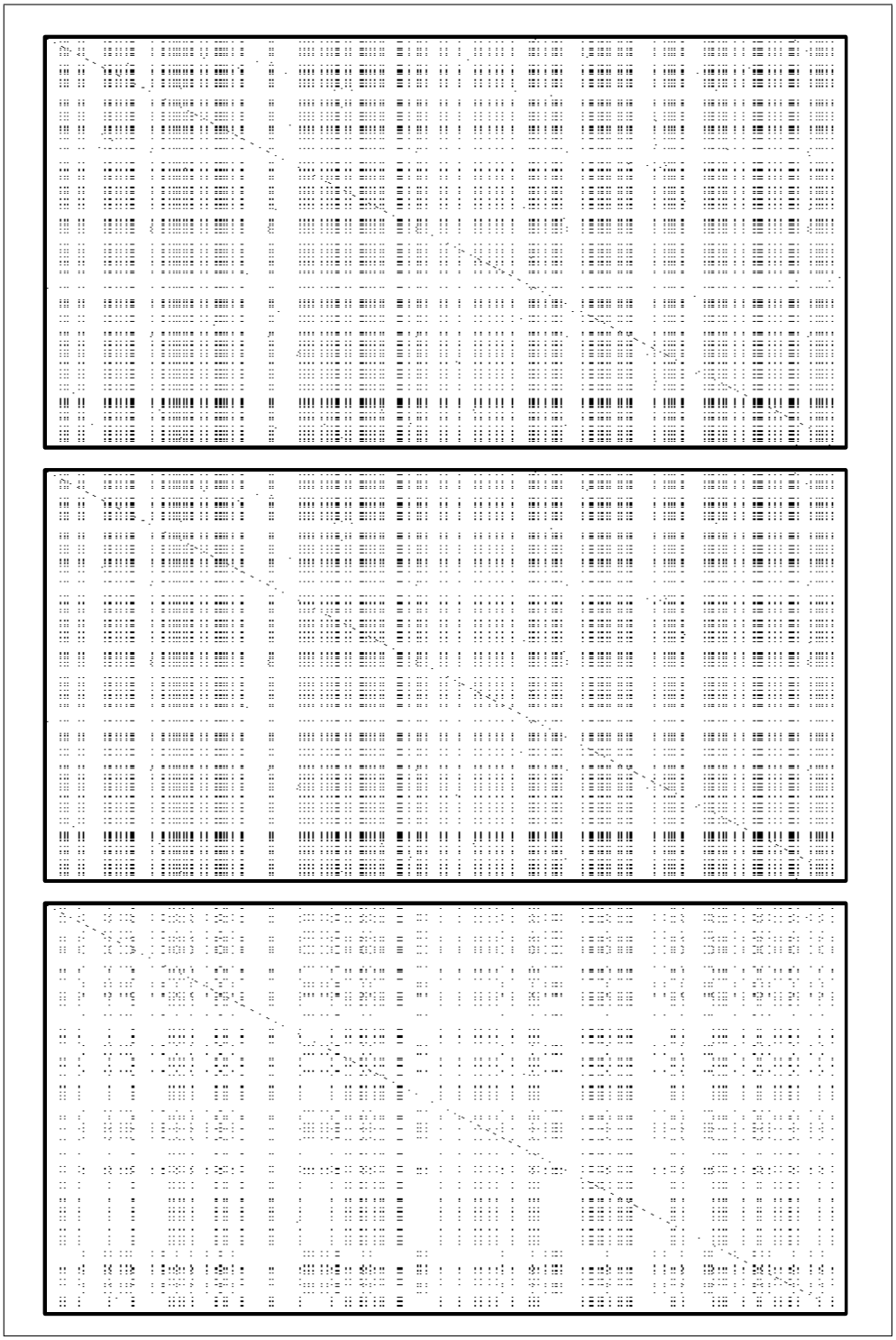

Figure 4.20.: The matrices from the clustering of the *AtGenExpress* group, resulting from cuts at levels 1-3. It can be observed that there is not much of a change between levels one and two, but at level three many clusters were decomposed into smaller sub clusters (less elements are in the same cluster and thus the matrix has less black entries).

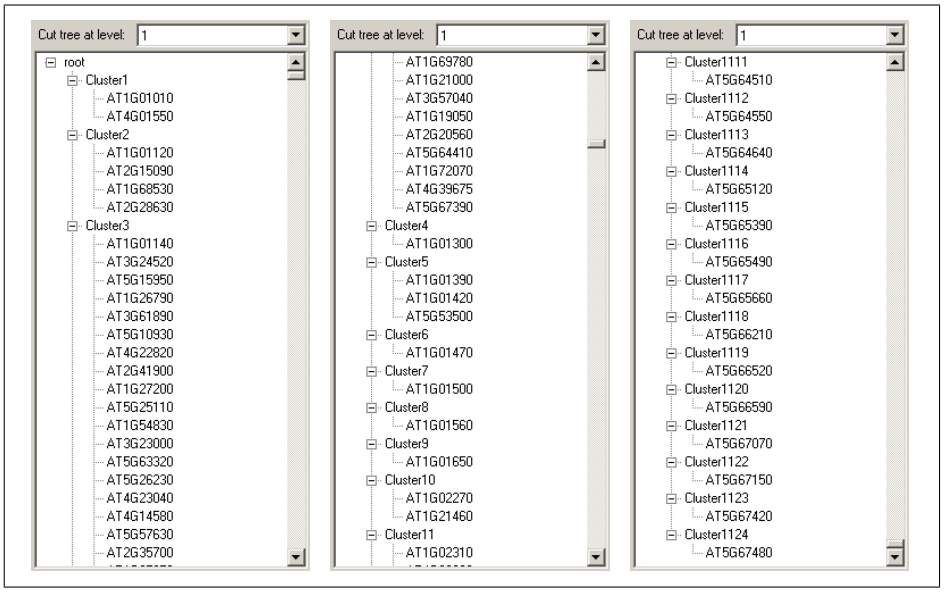

Figure 4.21.: Three parts from the clustering tree, which resulted from a cut at level one. As the leftmost and the middle parts show, a lot of elements land in the second cluster. This can be seen by looking at the position of the scrollbar in the middle picture. But there are also a lot of elements which are the only ones in their cluster, depicted by the right part. Therefore, the total number of clusters is comparatively high, although the threshold of *e-2* is quite relaxed.

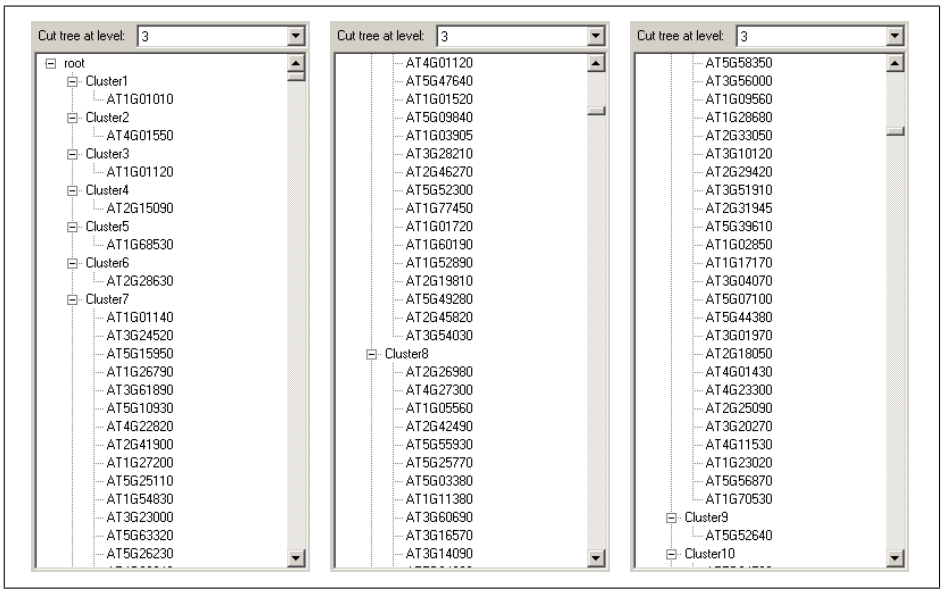

Figure 4.22.: The tree resulting from the cut at level three. The large cluster, shown in figure 4.21, was divided into two large clusters after clustering according to gene expression with a threshold of 0.95.

4.5.4. Hybrid clustering as a classifier

Clustering algorithms can sometimes be used as classifiers. One example is a *k-means* classifier. Assume that we have knowledge about data and come to the conclusion that this data can be distributed into five clusters. Now we take this clustering as the base for our *k-means* clustering, meaning that $k = 5$ and the cluster centroid of a cluster is the mean vector computed out of the vectors of this cluster's members. How this could look like was illustrated in figure 1.1. Now we find a new element, for example a transcription factor, and want to find out to which class the transcription factor belongs. We obtain therefore gene expression data in an experiment. We could then simply assign it to the class with the closest cluster centroid and then recompute this cluster's centroid. This method is of course naive. There are other approaches, for instance *Support Vector Machines*. They are not discussed in detail here, but there is plenty of information, e.g. Kecman explains them in the context of learning from experimental data [16]. A brief overview is given by the article at Wikipedia [23].

To determine whether the hybrid approach may be useful for classifying (in this case family classification) the data from the *AtGenExpress* group consisting of around 2000 members was analysed. The clustering described in section 4.5.3 was used again. Within this group there are some members of the *AP2/EREBP* family. This knowledge was used meaning that this family forms a class within the whole group. If the hybrid clustering could work as a classifier, these members should be in the same cluster, or at least almost all should. Under the assumption that a class has a high similarity, regardless if looking at sequences, motifs or gene expression, it should furthermore not make much of a difference at which level we cut the tree. Since the cuts at level one and level two lead to quite similar results, as described in section 4.5.3, we only look at the clusterings which result from the cuts at level one and three.

The members of the *AP2/EREBP* family were loaded from http://www.arabidopsis.org/browse/genefamily/AP2EREBP.jsp. For each member it was checked whether it occurs in the clustering. Furthermore, an integer value was assigned to each cluster indicating the number of members from the *AP2/EREBP* family it contains. If member m was found in cluster c, then the value of c was increased by 1. If the hybrid clustering is now a good classifier, almost all of the members should be concentrated in one cluster. This condition should also hold if we look at the clustering obtained by cutting the tree from the hybrid clustering at different levels. Figure 4.23 shows the results in two charts. 35 members of the *AP2/EREBP* family were identified within the data set from section 4.5.3. The upper chart resulted from cutting the tree at level one, reflecting the clustering according to sequence similarity in the first iteration. 31 out of those 35 members occurred in cluster three, three members occurred in cluster 78 and one member occurred in cluster 989. 31 out of 35 members seems to be quite good, although four members landed not in cluster three which is about 11%. The

lower chart shows that if the tree is cut at level three, reflecting the gene expression similarities, only 24 members remain in the same cluster.

As shown in figure 4.22, the large cluster at level one and two was split up into two large clusters at level three. The 31 members which were in the big cluster of level one distributed over the two big clusters at level three. Five are in cluster seven and 24 are in cluster eight, as the lower chart in figure 4.23 illustrates, summing up to 29, meaning that two other members are located in none of the big two clusters at level three. Under these observations, the conditions for having a good classifier do not hold. The hybrid approach is not meant to be a new or a better classifier which should replace existing ones. Its use is clearly not to classify data. It should help to analyse data from different perspectives which are reflected by the various features the data offers, e.g in biology like gene expression, sequences, motif occurences and so on.

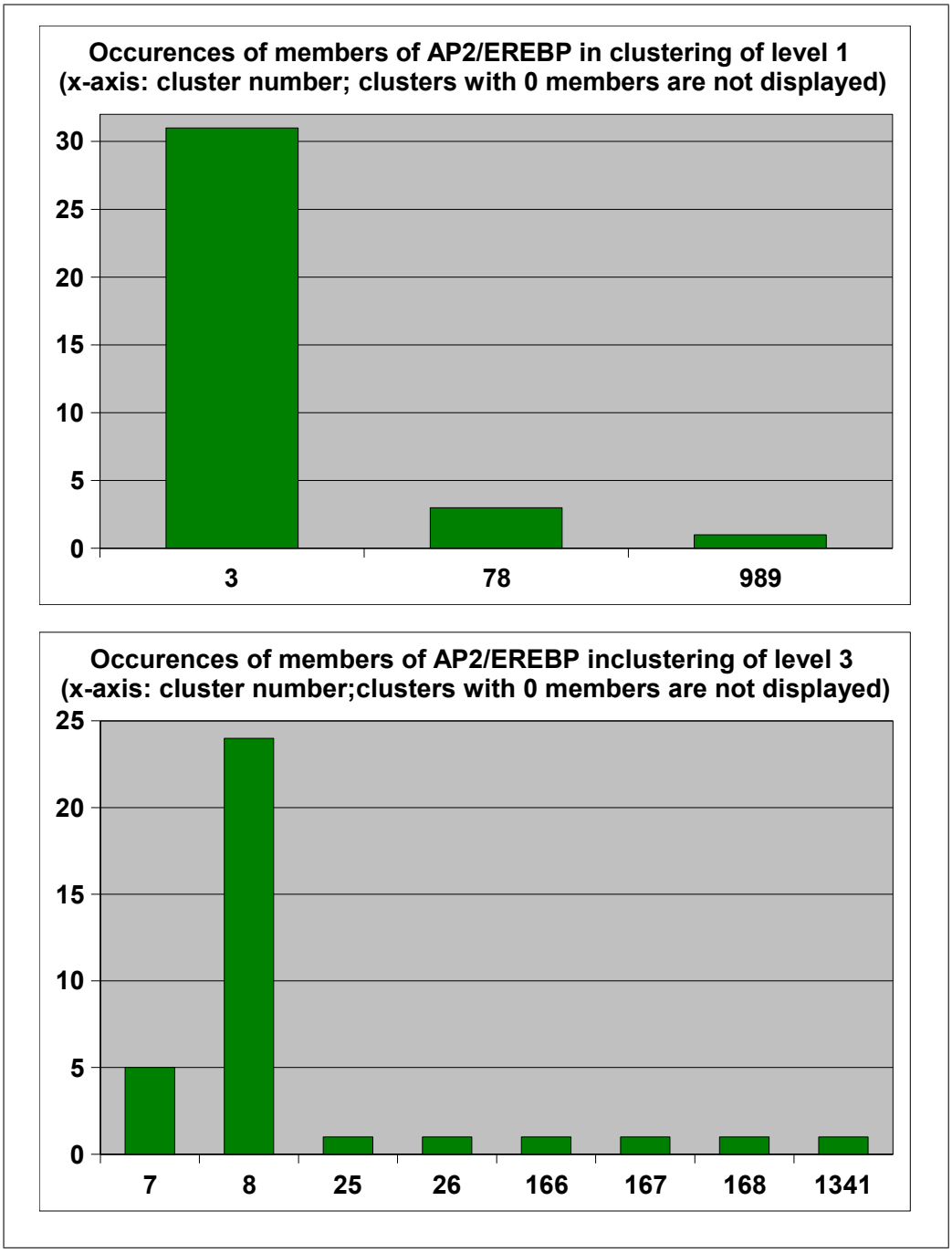

Figure 4.23.: Charts resulting from counting the number of occurences of *AP2/EREBP* members for each cluster. The upper chart shows the occurences in the clustering resulting from cutting the tree at level one, while the lower chart shows the occurences in the clustering resulting from cutting the tree at level two. The configuration of the clustering is the same as for the clustering described in section 4.5.3.

5. Future work

5.1. AT3G11020 and AT5G05410 from the AP2/EREBP family

As observed in section 4.5, these elements show a high motif occurrence similarity but a low gene expression similarity. The members from the *AP2/EREBP* family analysed for this thesis, tend to have a high gene expression similarity, but a low motif correlation if they are rather similar in sequence (*E value* computed by *Blast* was lower than *e-10*). Further comparative analysis of these two members of *AP2/EREBP* might be an interesting task for people related to biology. As mentioned in section 4.5.1, these members of the *AP2/EREBP* family have a role in drought stress. This can be found out by browsing the *Arabidopsis* data base (http://www.arabidopsis.org).

5.2. Genome-mean expression profile

The *Genome-mean expression profile* (GMEP) is an approach introduced by Chiang et al. [3]. Their work is closely related to motifs and transcriptional regulation. According to the authors, "many methods have been described that identify sequence motifs enriched in transcriptional control regions of genes that share similar gene expression patterns" [3]. Therefore most of the approaches identify the motifs by a "group-by-expression" method, i.e. they first identify genes with similar expression patterns and analyse their transcriptional control regions for the presence of shared sequence motifs. A review of these approaches is given by Ohler et al. [19]. The assumption behind these approaches is according to the authors of [3] that "genes with similar expression patterns are likely to be regulated by common factors, and thus should share binding sites for these factors in their non-coding regions" [3]. However, as the authors state there is one major problem with these approaches: As stated by Holmes et al. [12] they do not take into account that the process of regulation is in general influenced by multiple independent factors. As the authors of [3] state it often depends on the several conditions in an experiment whether two genes would be identified as e.g. co-expressed or not. They might be co-expressed under one set of conditions, but differentially expressed under others. Motivated by this observation Chiang et al. try an alternative approach. They analyse the behaviour of a motif in several conditions by calculating the so-called Genome-mean expression profile. Assume that for a motif m, G will be the set of genes which contain the motif in their TCRs (transcriptional control

regions). The Genome-mean expression profile of a motif m under the condition j is then defined as:

$$GMEP(m)_j = \frac{\sum_{g \in G} w_{mg} \cdot D_{gj}}{\sum_{g \in G} w_{mg}}$$

w_{mg} is the number of occurrences of motif m in the sequence of the transcriptional control region of gene g. D is a data matrix with r rows, each indicating a single gene, and c columns, each representing a single condition. D_{gj} indicates therefore the expression of gene g under condition j. The reason for the use of the *GMEP* is the following assumption the authors make: If a sequence motif carries transcriptional information (i.e. it is bound by a transcription factor) they "expect the expression pattern of genes containing this motif to have non-random features that reflect the activity of the corresponding transcription factor" [3]. The expression pattern of such genes differs therefore significantly from those of the entire population. Since the *GMEP* of a motif is the weighted mean expression of the genes containing the motif, it should differ significantly if the motif carries transcriptional information. The authors calculated the *GMEP* for a set of motifs under various conditions like abiotic stress. By that they obtained a data row of *GMEPs* for each motif and clustered the motifs according to this. It can be observed in [3] that there is indeed a significant behaviour of the motifs in the same cluster. For each cluster the authors calculated the cluster correlation as the *Pearson correlation* for all motifs within the cluster, showing that the intra-cluster-correlation (the correlation between *GMEPs* of motifs in the same cluster) is rather high.

The *GMEP* approach is not directly related to the hybrid clustering of this thesis since the main focus of the work of Chiang et al. lies on finding significant motifs. Clustering is what they did to show the use of the *GMEP*. However the *GMEP* has a high importance for sequence motifs and transcriptional regulation. In this thesis motifs were also used to cluster data. The use of the *GMEP* might help to improve the quality of calculating motif occurence similarities. In this thesis the motifs were taken from to data bases, *PLACE* and *PlantCare*. By the *GMEP* functionally related motifs can be identified, furthermore the *GMEP* helps to identify important motifs for a certain condition. If data is for instance to be clustered to obtain new insights in a drought stress study, using the *GMEP* can help to identify motifs having an important role in drought stress. The task of analysing how the *GMEP* can improve the motif data might be of interest especially for people related to data mining and knowledge engineering as well as for biologists.

5.3. Information fusion approach

In the approach introduced by Kasturi et al. [14] the authors clustered data related to yeast which they took from several experiments on budding yeast. In their work they describe precisely where the data comes from and how they

prepared the data for using it with their approach. As mentioned in section 4.4.2 the authors come to the conclusion that "genes with similar function might possibly share a common expression pattern under a certain experimental condition and might share a common motif" [14]. Under the use of the data they used as well as with other data, and maybe in combination with the *Genome-mean expression profile*, it could be analysed if this observation can be confirmed with the hybrid approach. This is also a task especially for people related to data and knowledge engineering and biologists.

5.4. Another graph clustering algorithm

It would be interesting to use another clustering algorithm than the current one, and to compare the results. This is maybe of interest for people related to algorithmics. Like the underlying clustering algorithm used in this thesis, the algorithm from Flake et al. [8] is a graph-based clustering algorithm. To explain the algorithm in short: it creates an artificial node, the authors call it the "artificial sink". It then connects this artificial node to every other node, with the weight α. It then computes a minimum cut tree of the new augmented graph. The artificial node is then deleted again as well as all its adjacent edges, decomposing the graph into independent sub graphs. The value α has an important role in the algorithm. The authors state that if α increases the number of clusters is non-decreasing. The best value can according to the authors be computed e.g. by a binary search. The central point is the computation of the minimum cut tree. Turau [21] gives a suggestion for the implementation of calculating minimum cuts. It is, unfortunately, in German. However, people interested in this suggestion for future work can contact the author of this thesis if they would like to get an "english version" of Turau's algorithm.

5.5. Several software tasks

The suggestions of this section might be of interest for people related to software engineering. An outline of the main points is given below. The suggestions include writing plug-ins to provide new functionality as well as making changes to the framework itself, because there is still room for improvement.

- **Display modification:** Right now, the display method is that the framework collects all the displays for the element on which the user clicks in the browser view. Each panel the displays provide is shown on a different tab page. This has two main disadvantages. Firstly, the user might want to see the same display type (e.g. a similarity matrix) of two different similarity sets at the same time, and then arrange them side by side to compare them. With the tab page solution, this is not possible. The second disadvantage is that all possible displays are always loaded, no matter if the user is interested in only a few. To load all of them also

takes time if the data set is large. The suggestion is that when the user right-clicks on a data set, the framework then collects all the relevant displays and shows their names in a context menu. The display itself is then opened in a sub window, so the tab panel should be replaced by a window container.

- **Registries:** At the moment the plug-ins have to register their new functionality at the framework's corresponding registries. Sometimes this cannot be avoided (e.g. for the storage managers, since there must be a unique storage manager for one set), but in many cases it is possible by searching the plug-in assembly for the desired interface implementation. Then the registration is not necessary anymore. The developer just implements the interface and loads the plug-in in the application. The rest is done by the framework.
- **Raw sets and algorithms for computing similarities:** Since the raw data was not the focus of this work, the framework does not contain any implementation for modelling several raw sets, e.g. sets with sequences. Therefore, it also does not provide any algorithms which can compute similarity sets out of raw sets yet. Modellings of raw sets and algorithms can be implemented as plug-ins to be integrated by the framework.

5.6. The vision of the software

One of the main tasks of this thesis was writing a software including the novel hybrid clustering approach. As mentioned in section 1.3.2 there are numerous common software applications, and some of them are text based and lack a graphical user interface. It would be good to also have a graphical user interface for them. A clustering can be performed from the "raw" data of the objects itself (where objects are for instance interpreted as a vector), but also from pre-computed similarities also based on the raw data. Even if the raw data itself is not necessary for the hybrid approach which was implemented here, the raw data is still connected somehow to the pairwise similarities and the clustering itself.

Some features of biological data do in fact allow a vector representation (e.g. gene expression under certain time points) so that the raw data may also be used for the hybrid approach in the future. In any case it would be much better if all the data, the raw data, similarity sets of the raw data, and clusterings of the raw data are handled through the same application. It may be less confusing to use one single application to compute the pairwise similarities of a set of raw data, and then cluster according to these similarities, instead of using program A to compute pairwise similarities, export them, and then import them into program B to compute the clustering. Furthermore the outputs from different programs are sometimes in a different format, which makes it hard to compare the results of two different clustering programs.

The goal of the development process was not only to write a software with a clustering algorithm, including a graphical user interface. Instead of this, the software comes along as a framework for computing similarities and clusterings of objects. The hybrid clustering algorithm is a plug-in set extending the framework, as well as different visualisations of the clustering results and tools to manipulate, measure and print them out.

In the future it is desired that many approaches are integrated in this framework. For command-line based clustering programs which lack a graphical user interface, the framework can just be extended with a simple wrapper window taking the configuring the command-line parameters and parsing them to the command-line based program. The result computed by this command-line based program is then automatically loaded into the clustering software by an importer. All the available tools and visualisations which the framework provides can then be used with this imported clustering created by the foreign program, e.g. visualise them as a cluster tree, cut the tree, compute several measures of the clustering and so on.

The advantage for the user is obvious. Instead of using a various number of programs, he would use one single application. All the visualisations of raw data, similarity data and clusterings are then consistent, and the same tools and measures can be applied to all of them. The comparison of the vision depicted by figure 5.1, with the current state described in section 1.3.2 and illustrated by figure 1.11, emphasises the benefit for the user.

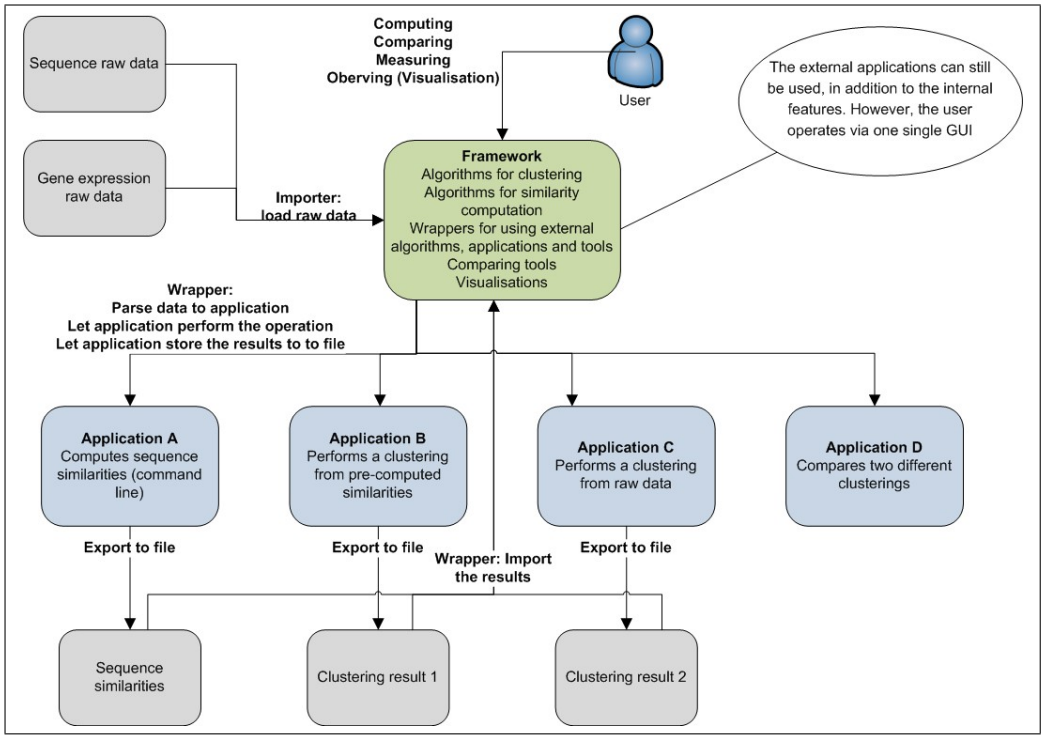

Figure 5.1.: The vision. The user can still use the old applications but does not recognise it. The framework either communicates with external applications (through wrappers), or uses internal functionality, and provides a consist displaying of the results. Several tools for analysing, as for instance measures, are also used via the framework.

6. Conclusions

In this thesis a new clustering approach was implemented and evaluated with biological data. The application was implemented as a software framework. From the software engineering view, it can be said that the language of choice *C#* provides many useful techniques like static classes, which were helpful during the implementation. The framework is now extendable with respect to various aspects, including the model, the graphical user interface and the algorithms. As pointed out in chapter 5 there are several possibilities how the software can be improved. While evaluating the clustering approach two interesting observations were made, on two groups of members within the *AP2/EREBP family*. One of these groups, namely the *CBF1-3* group, is well-studied, but a detailed comparative analysis of *AT3G11020* and *AT5G05410* could be interesting. As mentioned in section 4.5.1 the two groups show a behaviour which differs from the behaviour of the other members within the *AP2/EREBP* family of transcription factors. There is plenty of information available about the *AP2/EREBP* family. Kizis et al. [17] studied their role in gene regulation during abiotic stress. They come to the conclusion that "Our knowledge of the molecular mechanisms underlying the responses of plants to environmental stresses such as drought is still rather limited, but an increasing number of genes have been identified in recent years that mediate those responses.". Gilmour et al. [10] state that the three members *CBF1-3* of the *AP2/EREBP* family have redundant functional activity. It was highlighted during the evaluation that they show in fact a noticeable behaviour. They are rather similar in sequence similarity, gene expression and motif occurences. As mentioned by the Chiang et al. [3] multiple factors are involved in transcriptional regulation and especially motifs strongly depend on the conditions of the experiments. A careful selection of the motifs and the use of the *GMEP* suggested in [3] could lead to a further improvement.

The hybrid approach and the technique of cutting the tree and displaying the result in a matrix view was helpful to identify significant behaviour quickly. It would be interesting to build the hybrid clustering approach with other underlying clustering algorithms, to compare the quality between two hybrid clustering approaches with different underlying base clustering approaches. One interesting candidate was suggested by Flake et al. [8]. This algorithm is also a graph clustering as well as the underlying algorithm used in this thesis. Turau [21] gives suggestions for the implementation of the underlying minimum-cut tree computation. The algorithm described in [8] is a more recent one. Furthermore there are plenty of other, well-established algorithms, like *k-means* clustering. Some of them work directly on raw data, like the

algorithms used in *HCE* for instance. In contrast the current version of the hybrid approach uses pre-computed similarity sets. Thus the quality of the clustering depends on the quality of the underlying similarity set (its computation). However, the ideas of the hybrid clustering do not exclude the use of underlying clustering approaches which make direct use of the raw data. It was shown in section 2.1 that also hierarchical clusterings might be used as underlying algorithms of the hybrid approach.

From the viewpoint of software engineering, a new framework was developed providing basic functionality related to the process of clustering and analysing data. People interested in technical questions about the framework can refer to the technical documentation in the appendix or contact the author. It was shown that the *MVC* architectural pattern as well as the *Observer* design pattern assist the process of framework development. The two framework concepts *Eclipse* and *Lucene* gave ideas how to implement a framework which is on the one hand easy to extend (*Lucene*) but on the other hand still flexible enough to satisfy the various requirements coming up when extending a large end-user application (*Eclipse*).

After evaluating the data from *Indica* and measuring the clustering, it came out that the chosen measures have to be interpreted cautiously under the special conditions of this kind of clustering (it is rather different from common clustering approaches). In future, other approaches might come up focusing also on various aspects of the data and are therefore better to compare. Furthermore, as more such algorithms are developed there also might come new measures designed especially for comparing clustering algorithms focusing on different aspects of the data. However, comparisons between clusterings can and should be made, based on the results they deliver. As stated in section 5, the authors of the information fusion approach described in [14] describe precisely the data they used for the experiments. Comparing the two algorithms under the use of the same data could lead either to an "agreement" with the results and the conclusions or not. In each case, the insights obtained by such a comparison could help for designing new and better algorithms.

As emphasised in section 4.5, new insights could be obtained under the use of the hybrid clustering. Observations were made which seem to confirm some of the conclusions made in related work, cf. the observations Gilmour et al. [10] made concerning *CBF1-3* with the results of the hybrid clustering. It may be possible to improve the similarity values regarding motif occurrences, the *GMEP* introduced by Chiang et al. [3] could be helpful. Another approach also focusing on multiple aspects of the data is the information fusion approach from Kasturi et al. [14]. Since the amount of information available not only in biology but in many fields of research is growing and growing, approaches which focus on all or at least many of the features the information provides attract more and more attention. Due to this there might be plenty of new algorithms coming up in future, designed especially for focusing on multiple features of the data. A comparison between the hybrid approach

introduced in this thesis and other related algorithms like the information fusion algorithm described in [14] could give new insights into algorithm development as well as into data analysis.

Bibliography

[1] arabidopsis.org. About Arabidopsis. http://www.arabidopsis.org/portals/education/aboutarabidopsis.jsp, [Online; accessed March 2008].

[2] Y. Benjamini, E. Kenigsberg, A. Reiner, and D. Yekutieli. FDR adjustments of Microarray Experiments. 2005. http://www.math.tau.ac.il/~ybenja/Software/fdrame.pdf, [Online; accessed March 2008].

[3] D.Y. Chiang, P.O. Brown, and M.B. Eisen. Visualizing associations between genome sequences and gene expression data using genome-mean expression profiles. *Bioinformatics*, 17 (1), 2001.

[4] C. D'Angelo, J. Kilian, J. Kudla, O. Batistic, and S. Weinl. Experiment: AtGenExpress: Cold stress time course. http://arabidopsis.org/servlets/TairObject?type=expression_set&id=1007966553, [Online; accessed April 2008].

[5] D.L. Davies and D.W. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1 (4):224–227, 1979.

[6] P. Dayan. Unsupervised learning. *Appeared in Wilson, RA & Keil, F, editors. The MIT Encyclopedia of the Cognitive Sciences.* http://www.gatsby.ucl.ac.uk/~dayan/papers/dun99b.pdf, [Online; accessed March 2008].

[7] T. Eulgem, P.J. Rushton, S. Robatzek, and I.E. Somssich. The WRKY superfamily of plant transcription factors. *Trends in Plant Science*, 5 (5):199–206, 2000.

[8] G.W. Flake, R.E. Tarjan, and K. Tsioutsiouliklis. Graph Clustering and Minimum Cut Trees. *Internet Mathematics*, 1 (4), 2003.

[9] G. Gerber. Power point presentation: Clustering. page slide 41. http://www.mit.edu/~georg/papers/lecture6.ppt, [Online; accessed January 2008].

[10] S.J. Gilmour, S.G. Fowler, and M.F. Thomashow. Arabidopsis transcriptional activators CBF1, CBF2, and CBF3 have matching functional activities. *Plant Molecular Biology*, 54, 2004.

[11] S. Haykin. *Neural networks - A comprehensive foundation*, chapter 9. Self-organizing maps. Prentice-Hall, 2 edition, 1998. ISBN 0-13-908385-5.

[12] I. Holmes and W.J. Bruno. Finding regulatory elements using joint likelihoods for sequence and expression profile data. *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, pages 202–210, 2000.

[13] T. Kanungo, D.M. Mount, N.S. Netanyahu, C. Piatko, R. Silverman, and A.Y. Wu. An Efficient k-Means Clustering Algorithm: Analysis and Implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24 (7):881–892, 2002.

[14] J. Kasturi and R. Acharya. Clustering of diverse genomic data using information fusion. *Bioinformatics*, 21 (4), 2001.

[15] J. Kasturi, R. Acharya, and M. Ramanathan. An information theoretic approach for analyzing temporal patterns of gene expression. *Bioinformatics*, 19:449–458, 2003.

[16] V. Kecman. Learning and Soft Computing - Support Vector Machines, Neural Networks and Fuzzy Logic Models. 2001. ISBN 0-262-11255-8.

[17] D. Kizis, V. Lumbreras, and M. Pages. Role of AP2/EREBP transcription factors in gene regulation during abiotic stress. *FEBS Letters*, 2001.

[18] T. Nakano, K. Suzuki, T. Fujimura, , and H. Shinshi. Genome-Wide Analysis of the ERF Gene Family in Arabidopsis and Rice. *Plant Physiology*, 140:411–432, 2006.

[19] U. Ohler and H. Niemann. Identification and analysis of eukaryotic promoters: recent computational approaches. *Trends in Genetics*, 17:56–60, 2001.

[20] G. Sessa, G. Morelli, and I. Ruberti. DNA-binding specificity of the homeodomain-leucine zipper domain. *Journal of Molecular Biology*, 274:303–309, 1997.

[21] V. Turau. *Algorithmische Graphentheorie*, pages 88–92 and 222–233. Oldenburg, 2004. ISBN 3-486-20038-0.

[22] wikipedia.org. Wikipedia article about Arabidopsis. http://en.wikipedia.org/wiki/Arabidopsis, [Online; accessed March 2008].

[23] wikipedia.org. Wikipedia article about Support Vector Machines. http://en.wikipedia.org/wiki/Support_vector_machine, [Online; accessed March 2008].

[24] K.L. Wu, Z.J. Guo, H.H. Wang, and J. Li. The WRKY Family of Transcription Factors in Rice and Arabidopsis and Their Origins. *DNA Research*, 12:9–26, 2005.

[25] Z. Wu, R.A. Irrizarry, R. Gentleman, F.M. Murillo, and F. Spencer. A Model Based Background Adjustment for Oligonucleotide Expression Arrays. *Working Papers Working Paper 1*, 2004. http://www.bepress.com/cgi/viewcontent.cgi?article=1001&context=jhubiostat, [Online; accessed March 2008].

[26] K.Y. Yeung and W.L. Ruzzo. An empirical study on Principal Component Analysis for clustering gene expression data. *Technical Report UW-CSE-01-04-02*, 2001.

[27] K.Y. Yeung and W.L. Ruzzo. Details of the Adjusted Rand index and Clustering algorithms. *To appear in Bioinformatics*, 2001.

[28] Y. Zhang and L. Wang. The WRKY transcription factor superfamily: its origin in eukaryotes and expansion in plants. *BMC Evolutionary Biology*, 5 (1), 2005.

Appendix

A. Glossary

Artificial neural network
A mathematical model that is based on biological neural networks. It is usually an adaptive system, i.e. it changes its structure due to external or internal information that flows through it.

AP2/EREBP
A family of transcription factors which have a central role for Arabidopsis and rice. They have an important role in resistance to abiotic stress, according to [17] and [10]

AP2/ERF
This refers to the same family as AP2/EREBP

Arabidopsis
A family of small flowering plants related to cabbage and mustard. The AP2/EREBP Transcription factors which are related to this family, were used for the evaluation. More information can be found at http://www.arabidopsis.org

AtGenExpress
In this experiment, gene expression under cold stress within Arabidopsis is studied. More information about the experiment from D'Angelo et al. can be found at [4].

Blast
A collection of programs for analysing biological data. It is used for matching DNA or protein sequences to a data base, aligning sequences etc. The web site is http://www.ncbi.nlm.nih.gov/blast/Blast.cgi.

Clustal
Clustal is a program for computing multiple sequence alignments. To achieve this, Clustal computes pairwise similarities and a clustering. There are command-line based versions of Clustal as well as versions including a graphical user interface. Downloads of the program can be accessed at http://www.clustal.org/.

E value *Expected value*
This is a value which Blast delivers when comparing sequences. Roughly said, if the value s measures the similarity between two sequences, then the E value is the expected number of sequences which score an equal or better similarity value with one of those two sequences, when performing a database search by chance. Therefore, the lower the value, the more significant the score is.

Eclipse
An open source software framework. Eclipse is mainly used for developing software in Java, but not necessarily, as shown e.g. in the ISTP project (see http://wagner.st.informatik.tu-darmstadt.de/se2006/eos2oft/, where Eclipse was used for assisting software support.

GeneSpring
This is a visualization and analysis tool designed for use with gene expression data from Agilent Technologies (http://www.chem.agilent.com).

HCE
Hierarchical Clustering Explorer. A program for computing hierarchical clusterings with various measures, e.g. average linkage and single linkage, euclidean distance and Pearson correlation. The resulting tree can be cut to retrieve a non-hierarchical clustering, which can be printed to a plain text file. The web site is http://www.cs.umd.edu/hcil/hce/.

Indica *Oryza sativa cv Indica*
A cultivar of rice. A large data set from this family was used for the evaluation of the application.

Lucene
An open source software framework for indexing data and searching within indexed data.

Microarray
This is a large-scale technology for measuring gene expression.

Neighbor-Joining algorithm
A bottom-up hierarchical clustering approach. In this algorithm, the two closest cluster (the clusters with the best similarity score) are fused together to one new cluster. After that, the distance from the new cluster to all other clusters is re-computed. The algorithm is similar to the UPGMA method, but Neighbor-Joining doesn't calculate the new distances under the assumption of the molecular clock (i.e. that all taxa evolve with the same, constant change rate)

Pearson correlation
A correlation coefficient which indicates the strength and direction of a linear relationship between two variables. It is often used to measure the similarity between gene expressions. There, the empirical correlation coefficient can be used, which is

$$r_{xy} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2}\sqrt{\sum(y_i - \bar{y})^2}}$$

PLACE
A database, similar to PlantCare, which contains Cis-acting regulatory elements. The web site is http://www.dna.affrc.go.jp/PLACE/.

PlantCare
A database for plant promoters and their cis-acting regulating elements. Cis-acting elements are, in the context of transcription regulation, DNA sequences which regulate the expression of genes on the same strand. The web site is
http://bioinformatics.psb.ugent.be/webtools/plantcare/html/

SOM *Self-organizing map*
An *artificial neural network* that is trained using *unsupervised learning*.

SVM *Support Vector Machines*
Support Vector Machines are methods for supervised learning and used for classification. There is plenty of information on the web available, for instance the online article at wikipedia.org [23].

Unsupervised learning
Unsupervised learning belongs to the field of machine learning. In contrast to supervised learning, there is no "teacher" showing how to perform a task by giving a set of labeled examples. P. Dayan gives an introduction to it in [6].

UPGMA
Another bottom-up hierarchical clustering approach. It works like Neighbor-Joining, but after fusing the two closest clusters to a new one, the distance from the new cluster to the other clusters is computed in a different way than in the Neighbor-Joining approach.

WRKY
Another family of transcription factors in Arabidopsis and rice. A small subset from this family was taken during the evaluation, but mainly for testing the application. More information about this family can for instance be found in [24].

B. AP2/EREBP members used for the evaluation

Figure B.1 shows the members of the *AP2/EREBP* transcription factors, which were used for the evaluation of the software. This is a raw set with gene expression data. Out of this data, the Pearson correlations where computed, leading to a similarity matrix which was used as an input parameter for the clustering.

	0	0.5	1	2	3	4	6	8	12	24
AT2G23340	0.99	0.89	0.88	1.03	1.18	5.53	14.24	25.93	49.3	12.05
AT4G17490	0.96	1.21	3.19	10.11	17.03	14.55	9.59	6.74	1.05	0.27
AT4G17500	0.97	0.23	0.53	0.46	0.39	0.29	0.11	0.09	0.07	0.07
AT4G36920	1	0.69	1.02	0.92	0.82	1.13	1.75	3.02	5.55	2.46
AT4G36900	1	1.29	1.01	1.01	1	1.2	1.61	2.28	3.61	3.25
AT5G25190	0.98	0.97	1.56	1.54	1.52	1.15	0.4	0.29	0.06	0.03
AT5G53290	1	1.06	1.09	2.94	4.79	10.57	22.12	19.88	15.39	2.95
AT5G51990	1	1.05	1.25	20.62	40	31.51	14.54	10.12	1.27	3.44
AT5G51190	1	1.05	1.53	1.78	2.03	1.41	0.17	0.14	0.08	0.07
AT5G47220	0.98	0.78	1.8	2.87	3.94	3.45	2.47	2.23	1.73	0.23
AT5G47230	0.95	0.93	1.81	2.54	3.27	2.99	2.43	3.28	4.99	1.45
AT5G44210	1	0.86	1.01	1.02	1.04	1.56	2.6	3.8	6.19	2.4
AT5G11590	0.99	1.01	1.13	1.28	1.43	1.93	2.94	4.42	7.38	2.29
AT5G07580	1	0.88	0.84	0.67	0.5	0.38	0.15	0.13	0.1	0.08
AT5G05410	0.99	0.5	0.69	14.18	27.68	37.08	55.87	47.36	30.35	23.25
AT3G50260	0.97	0.73	1.21	8.64	16.07	55.28	133.7	158.37	207.7	36.25
AT4G32800	0.97	0.85	0.79	3.35	5.92	9.16	15.66	13.03	7.78	2.35
AT4G28140	1	1.55	1.19	1.4	1.6	10.23	27.49	34.61	48.84	7.44
AT4G25480	1	1.39	40.38	416.04	791.7	801.8	822	676.83	386.5	48.07
AT4G25490	0.96	1.33	42.86	411.38	779.9	639.37	358.3	250.36	34.48	50.22
AT4G25470	1	5.56	113.9	601.95	1090	1020.57	881.7	680.03	276.7	89.15
AT4G23750	0.99	2.49	4.46	5.94	7.42	8.58	10.91	12.11	14.51	2.37
AT1G22190	1	0.6	0.85	0.69	0.53	0.37	0.05	0.04	0.03	0.13
AT3G11020	0.98	1.77	1.41	6.92	12.42	19.37	33.28	32.84	31.97	11.31
AT3G15210	1	0.73	1.33	3.29	5.25	6	7.5	7.03	6.1	7.75
AT1G64380	0.99	1.22	1.92	2.6	3.28	5.52	10.02	8.27	4.76	2.39
AT1G68550	1	1.01	1.61	1.64	1.67	1.86	2.25	3.26	5.28	9.28
AT1G28370	0.96	0.57	1.8	9.12	16.45	13.41	7.34	5.37	1.42	1.59
AT2G40350	1	1.24	1.24	1.17	1.11	3.96	9.66	22.18	47.21	10.43
AT2G28550	1	1.19	1.31	1.55	1.79	2.63	4.31	5.13	6.77	3.32
AT2G35700	0.96	2	2	2.35	2.7	5.05	9.73	14.43	23.81	3.3
AT2G46310	1	1.83	2.61	2.83	3.04	2.5	1.42	1.13	0.54	1.14
AT2G44940	1	0.86	4	8.06	12.11	14.27	18.58	17.31	14.78	5.92
AT2G39250	1	1.54	1.32	1.58	1.84	2.18	2.87	3.01	3.3	3.83

Figure B.1.: AP2/EREBP members used for the evaluation, including the gene expression.

C. Technical documentation

C.1. Technical solutions to the requirements

This part gives further information about the structure of the software, with the purpose of showing how the framework can be extended to satisfy the different user requirements. In contrast, section C.3 showed the ideas for the solutions, but not from the technical view point. The identifiers for the user requirements (UR) are the same as in C.3, where the reader can look up the explanation of the user requirement and the ideas for the solution. Furthermore, there are specific framework requirements (FR) which are not a direct result from the user requirements but indirect results from the extendable nature of the framework. The technical solutions for these requirements is also shown here.

UR1:

The Project class contains three instances of the generic class *SetsContainer<N>*. The class parameter *N* takes the values *AbstractRawSet*, *AbstractSimilaritySet* and *AbstractClusterSet*, which are abstract classes subclassing the basic class *AbstractSet* for the data. For providing e.g. raw data which models DNA sequences, the plug-in should contain a class which extends *AbstractRawSet*.

UR2:

The framework defines an interface *IDisplayProvider*. This set has two relevant methods. The first method *Accept(Object o)* returns true if the display can handle the object. Usually, the parsed object is a subclass of *AbstractSet*, so therefore a raw set, a similarity set or a cluster set. The second method *CreateDisplayPanel(Object o)* creates and returns a display panel, according to the data the parsed object contains. This can for example be a panel which displays a cluster tree if the parsed object was a cluster set. If the user selects a set in the browser, the framework collects all the classes from the plug-ins which implement the *IDisplayProvider* interface. For every provider which delivers true by calling its *Accept* method, the framework adds the panel which they create on a tab page. To provide new displays, the plug-in should contain a class which implements the interface *IDisplayProvider*.

UR3:

The framework provides an interface *ISetImporter* with a method *Accepts(String)* which returns true if the file to which the parsed string points

can be imported by this class. After the user selected the desired file to import, the framework collects all the classes from the plug-ins which implement the *ISetImporter* interface. When the correct importer is identified, the framework calls its *Import(String)* method, which imports the set from the desired file. To provide new importers, the plug-in should contain a class which implements the *ISetImporter* interface.

UR4 and UR5:

These requirements are subsumed since an export functionality is not explicitely integrated in the framework. It can be treated and implemented like a tool. The exporting functionality and tools can be provided either via a display or by a new context menu item. In the case of providing it on a display, it only depends of its implementation and is up to the developer (for providing new displays, see *UR3*). For providing new context menu items, see *FR3*.

UR6:

The framework completely handles the workspace organisation, so there is no need for providing a new functionality here. The plug-ins can neither see nor use the relevant classes of the framework, which are responsible for this part.

UR7:

This is done by one of the plug-ins. However, the classes of this plug-in can be used by another plug-in. A possibility could be to write a subclass of *StandardHybridGraphClusterAlgorithm* which is contained in the plug-in set *PleiadesHybridClusterPlugins*, to provide a new hybrid clustering algorithm which uses another underlying base clustering algorithm, e.g. the algorithm which is described by Flake et al. in [8]. How to provide new algorithms is shown in *FR4*.

UR8:

There are four interfaces, two for wizards for the similarity algorithms and two for wizards for the clustering algorithms. The interface *IClusterWizardPageProvider* describes two methods. *Accepts(IClusterAlgorithmProvider)* returns true if the wizard page which is created by the provider can configure the given clustering algorithm (to be precisely, the given provider which creates the algorithm). The method *CreateWizardPage(Project, IClusterAlgorithmProvider)* parses the given parent project and the provider of the clustering algorithm to the wizard page and returns the wizard page.

The interface *IClusterWizardPage* describes the methods *CheckConfiguration()*, which returns *true* if the configuration is completed and *ConfigureAndReturnAlgorithm()*, which returns the configured algorithm.

If *CheckConfiguration()* returns *true*, the framework retrieves the configured algorithm by calling the *ConfigureAndReturnAlgorithm()* method and launches it.

To provide new clustering wizards, the interfaces *IClusterWizardPageProvider* and *IClusterWizardPage* should be implemented. To provide new similarity wizards the interfaces *ISimilarityWizardPageProvider* and *ISimilarityWizardPageProvider* should be implemented analogously.

UR9:

The framework handles the representation in the browser view, so there is no need for providing new functionality here. The relevant classes of the framework are neither visible nor usable for plug-ins.

UR10:

To provide new tutorial pages, the plug-in should create instances of the class *HelpPageProvider*. The only constructor is *HelpPageProvider(String, Uri)*. The given string represents the topic name of the help page. This topic name will be shown in the list on the left of the help browser. When the user selects the desired topic from the list, the document is shown to which the corresponding URI of the HelpPageProvider points. This can be HTML pages, PDF files, images etc.

FR1:

The kinds of data sets should be extended (e.g. integrating new kinds of raw data).

Solution: The framework defines three abstract classes, *AbstractRawSet*, *AbstractSimilaritySet* and *AbstractClusterSet*. They subclass *AbstractSet*. This class has only two public methods, *GetName()* and *GetParent()*. *GetName()* provides an identifier for the set, which has to be unique within the project folder. E.g. a project cannot contain two raw data sets called "AP2EREPB". But to have a raw set called "AP2EREPB" and a cluster set "AP2EREPB" in the same project is of course not a problem. The method *GetParent()* returns the parent project which contains the set.

Every plug-in which wants to provide new kinds of sets must extend the correct abstract class. For instance if a plug-in provides a class of raw sets which contain protein sequences, it can have a class called *SequenceRawSet* which contains a set of sequences. It has therefore to subclass *AbstractRawSet*, or an already existing class which subclasses *AbstractRawSet*. If it subclasses an already existing subclass of *AbstractRawSet*, all the displays which can handle *AbstractRawSet* instances will also be able to display instances of the new class. If this kind of raw set is totally new (in the example of sequences, there is no class which models sequences yet and therefore no display classes

which can visualise the sequences), it is up to the developer to also provide new display classes which can handle this raw data.

FR2:

Because of FR1, abstract also the process of storing and loading data sets, dependent on the data.

Solution: The framework defines an interface *ISetStorageManager*, which has two methods, *Store(AbstractSet, StreamWriter)* and *Load(StreamReader)*. The store method gets as input the set to handle and writes the data to a file via the stream writer. Set and stream writer are parsed by the framework. The load method gets the stream reader as parameter. The stream reader has already the matching file stream opened. The method reads the stream with the stream reader, creates the data set and returns it.

The plug-ins have to register the storage managers in the framework's static class *SetStorageManagerRegistry*. They have to register a pair consisting of the unique type identifier for set class, and the storage manager which can handle this set. When storing a set, the framework looks in the registry for the storage manager which corresponds to the set's type identifier. It then creates a file with a stream writer. The name of the file corresponds to the set's name. Subsequently, the framework writes the header line of the file which is just the type identifier. Then it launches the *Store* method from the storage manager which completes the file and closes the stream.

When loading a file, the framework opens the file with a stream reader and reads the header line, which contains a type identifier. Then it fetches the storage manager corresponding to this identifier (i.e. the storage manager which can handle the type of set which the type identifier represents) from the registry and calls the manager's *Load* method.

FR3:

Provide a possibility to add new kinds of views and tools.

Solution: As mentioned in *UR2*, the views are handled by the interface *IDisplayProvider*. Plug-ins can implement this interface to provide new views. These views can of course also include some tools. There is one plug-in which provides a view with an integrated tool. It shows the clustering as a tree, and can cut this tree. The result can then be printed to a file.

But there is another way to provide new tools. If the user right-clicks on a set in the browser view, a context menu appears. This context menu is not fix; it is extendable, so plug-ins can provide new entries. For that, the framework defines an interface *IContextMenuItemProvider* which has two methods. The *Accept(Object)* method returns *true* if the context menu item should appear by right-clicking on the parsed object, e.g. *true* if the user right-clicked on a

specific cluster set. The *CreateItem()* method returns the context menu item which should be added to the context menu. When a user right-clicks on a set in the browser, the framework collects all the classes from the plug-ins which implement the interface and return *true* when calling their *Accept* method. For those classes, it adds the context menu item the *CreateItem()* method returns. It is then up to the developer what should happen by clicking on this item, e.g. a file browser can appear with which the user can print the set to a file.

FR4:

Make it possible to add new kinds of algorithms.

Solution: The framework defines two interfaces, *IClusterAlgorithmProvider* and *ISimilarityAlgorithmProvider*. The *IClusterAlgorithmProvider* interface defines two methods. *Accepts(AbstractSet)* should return *true* if the algorithm accepts this kind of set (e.g. a similarity set with gene expression similarities) as input. The *CreateClusterAlgorithm()* method returns then a clustering algorithm, for instance the hybrid clustering algorithm.

When the user then wants to compute a new clustering, the framework collects all the classes which implement the interface and shows them in a list. When the user selects one, the framework shows all the wizards which can configure the algorithm. After configuring via the wizard page, the framework invokes the *CreateClusterSet()* method from the configured algorithm, which performs the computation. For the interface *ISimilarityAlgorithmProvider*, which provides an algorithm for similarity computation, the system is analogue.

FR5:

Since different algorithms need different configurations, also provide an extension point for new configuration wizards.

Solution: The extension points where already described in *UR8*. The framework displays all the available clustering algorithms (it checks all the plug-ins for classes which implement the *IClusterAlgorithm* interface) in a list, and the user selects the desired one. The framework shows then all the wizards which can configure the selected algorithm. When the user choses one, the framework calls the *CreateWizardPage()* method, also defined by the *IClusterWizardPageProvider* interface, which returns a configuration panel. For the *ISimilarityWizardPageProvider*, it works analogue. It provides a wizard page which can configure an algorithm for computing similarity sets (e.g. compute sequence similarities from a set of DNA sequences).

C.2. MVC architecture of the framework

- ***core:*** This part contains core classes which are totally independent from the concrete application itself. They could be used in any other

application which has nothing to do with this one. It lies outside the structure of the *MVC* pattern.

- ***model:*** This is the most important part of the framework. It models the abstraction of projects, raw sets, similarity sets and cluster sets, without any relation to their visualisation or their physical representation on the file system. According to the *MVC* pattern, this is the model, which is independent from the view.
- ***ui:*** This part contains all the graphical elements through which the user interacts with the application, including their controllers. It is therefore both the view and the controller part. However, the separation of them is kept under the use of the technique of partial classes.
- ***algorithms:*** The algorithms for computing clusterings and similaritiy sets. This part lies outside the structure of the *MVC* pattern.
- ***filesystem:*** Everything that manages the communication with the file system, storing sets, loading a project etc. belongs to this part. It also lies outside the structure of the *MVC* pattern.

Some parts of the application are outside the UI and the model. Every GUI element which is contained in the ui part, consists of (at least) two partial classes. If the class name is for instance *Class1*, then there exists one file "Class1.cs" containing the partial class for the controller logic, according to the *MVC* pattern. The file "Class1.Designer.cs" then contains the view. In some classes, the controller logic is divided into more partial classes depend on the aspect. There also may be a partial class in the file "Class1.observer.cs". It contains the controller logic which is related to its *Observer* functionality. According to the *Observer* pattern (an important design pattern of the framework), every change of the model results in notifying the observers which listen on this model part. The observer then performs an *actualise* process, e.g. it actualises the view. This assures that the model is independent from the view. Thus all control functionality related to the *actualise* aspect of the Observer pattern, is situated in the corresponding partial class (in the example, "Class1.observer.cs"). Figure C.1 gives an overview of the parts of the framework, including the extension points.

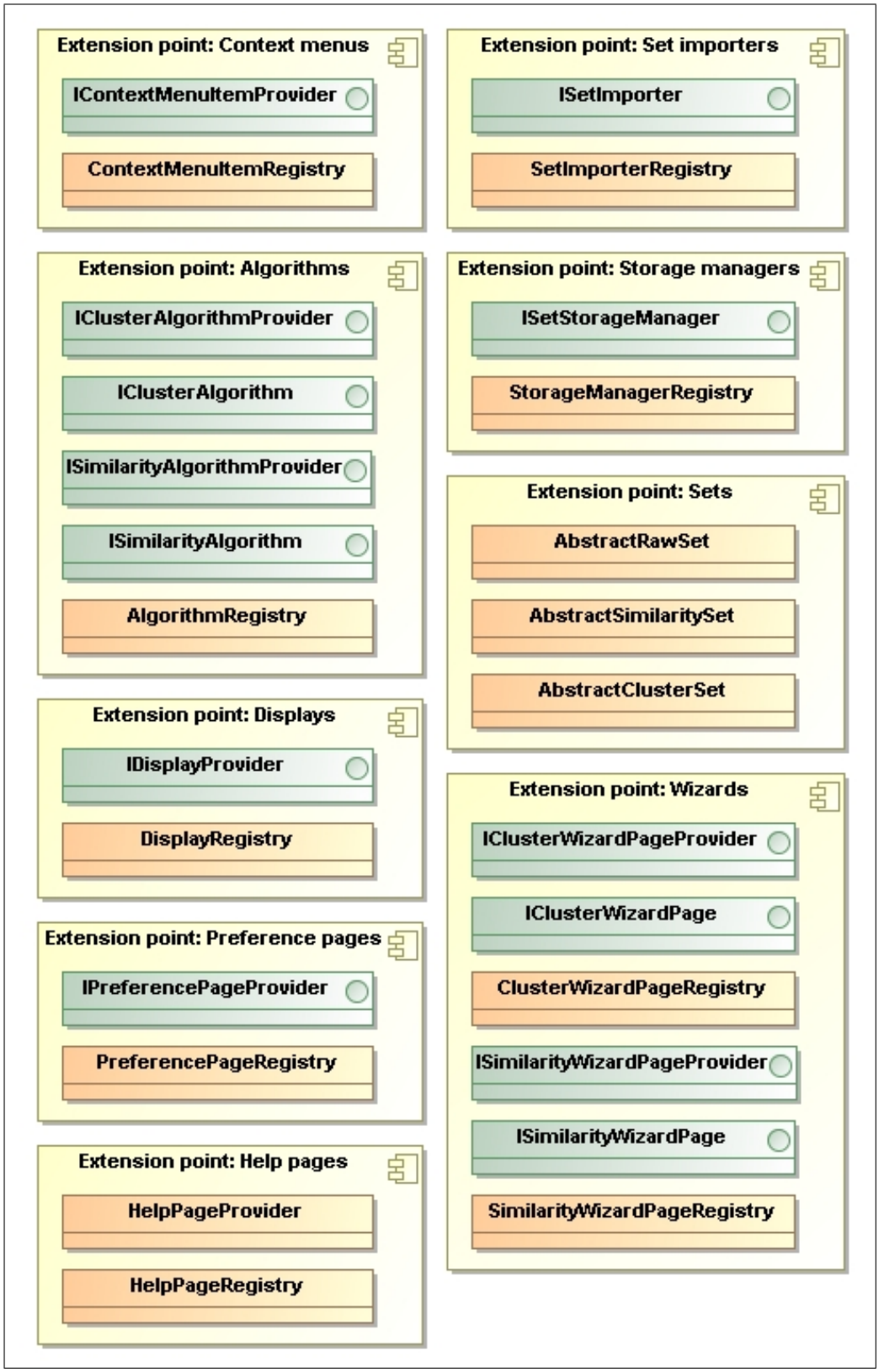

Figure C.1.: The extension points of the framework, including the relevant interfaces and classes.

C.3. Design patterns

This section shows the architecture of the framework from the view point of the leading design pattern, the so-called *Observer* pattern. There is a second pattern with a minor role, namely the *Singleton* pattern which is described in the second part.

C.3.1. Major design pattern: Observer

The observer pattern defines two main parts, the *publisher* and the *observer*. The framework contains therefore two interfaces, *IPublisher* and *IObserver*. The interfacess are described in figure C.2.

```
public interface IObserver
{
        void Actualize(IPublisher p);
}
public interface IPublisher
{
        void AddObserver(IObserver o);
        void RemoveObserver(IObserver o);
        void NotifyObservers();
        IEnumerable<Delta> ChangeInfo
        {
                get;
        }
}
```

Figure C.2.: Interfaces for the *Observer* pattern.

The classes belonging to the "model" part implement the *IPublisher* interface, and the classes containing representations of the model (it doesn't matter if it is a graphical representation with which the user interacts or a physical representation which maps the model e.g. to the file system) implement the *IObserver* interface. An observer providing some representation of the model is then registered in this model by adding it via the *AddObserver(Observer)* method. When the model changes, it notifies all the registered observers by calling their *Actualize(IPublisher)* method. It parses itself as the parameter. Furthermore, the publisher provides a list of changes describing the changes in an abstract way (e.g. a *String* "SET_ADDED") and providing additional information (for instance the cluster set which was added).

The observer can then check the publisher, i.e. the model, by inspecting the list of changes and the provided additional data. According to that it actualises the representation of the model (e.g. it creates a new file for this cluster set and stores the information there, or it creates a new node for the set in the browser view). The following classes of the framework implement the *IObserver* interface:

- **WorkspaceManager:** The main class which handles the mapping of the model to the file system. All the logic which is related to the observer aspect is contained in the partial class in the file "WorkspaceManager.observer.cs".
- **MainMenu:** The main menu of the application contains a browser view on the left. This is the graphical representation of the model, on which the user works (the projects including their raw-, similarity-, and cluster sets).

The classes implementing the *IPublisher* interface are:

- **Model:** The main model which contains all the data the user is working on, the projects including their data. It notifies the observers when changes to the projects occur, such as adding or deleting a project.
- **SetsContainer<N>:** A generic class which contains a certain kind of sets, e.g. similarity sets (in this case, the instance would be of the class *SetsContainer<AbstractSimilaritySet>*. It is part of the *Project* class. The graphical representation of instances of this class is a folder item in the browser view of the main menu, which therefore is therefore registered as an observer. Whenever sets are added or deleted, this model part then notifies the observers which update the view or perform the changes on the file system.

C.3.2. Minor design pattern: Singleton

There is a minor pattern called *Singleton*, which is also used in the implementation of the software. It is minor importance compared to the *Observer* pattern and hence not discussed in detail. The pattern is mainly used if there is only one single existing instance of a class desired. This is important when a system works with a database for instance. Then there must not be two or more connections to the database existing at the same time because this will cause errors.

If there is just one instance of a class needed which should also contain some fields but the class is never assigned to a variable as for instance by the statement

SingletonClass a = SingletonClass.singletonInstance();

but only by call statements such as

SingletonClass.singletonInstance().methodA();

then the language of choice, *C#*, provides technique called *static classes*, making the singleton pattern unnecessary. The technique was used within the applications for a lot of registries through which plug-ins can add their new functionality to the framework.

Nevertheless the *Singleton* pattern was necessary for two classes because the singleton instance of these classes was referenced to a variable pointing to it. This is not possible with static classes. The two classes are the *MainMenu* and the *WorkspaceManager* class. It is obvious that there exists only one instance of them: Only one main menu and only one class handling the mapping of the model to the file system.

The classes implement the *Observer* pattern an thus the singleton instances of them are registered at the publisher site, i.e. the model, by calling the *AddObserver(IObserver)* method and passing them as parameter. If they were static classes the method call would not be possible as static classes cannot be assigned to a variable and thus the *Singleton* pattern was used.

C.4. Coding conventions

The following sections list the several conventions followed while writing the software. A developer who wants extend or modify it should follow these conventions as well, to assure a consistent appearance of the program code.

Name spaces

The namespaces should be all lowercase letters. The should start with a non-digit and must not contain any special characters, only standard lowercase letters (a-z) and digits are permitted. The namespace has to start with "se.his.bioin.pleiades". Normally the name of the folder in which the concerned class is located is concatenated to this prefix, but if there are reasonable rules this can be handled differently.

Folder naming

Folder names always consist of standard lowercase non-digit letters (a-z).

Class naming

The name of a class should always be the same name as the file prefix in which it is contained. E.g. the class in the file "Util.cs" should always be named "Util". Class names always start with a uppercase letter and only consist of standard non-digit letters (a-z, A-Z). The name should contain one lowercase letter at least.

Method naming

- **Public non-static methods** always start with an uppercase letter, followed by standard letters (a-z, A-Z) or digits (0-9). The name should contain one lowercase letter at least.
- **Other non-static methods** start with a lowercase letter, followed by standard letters (a-z, A-Z) or digits (0-9).

- **Static methods** start with an uppercase letter and consist only of uppercase letters (A-Z) and digits (0-9). The only special character permitted is the underscore "_".

Property naming

Same as for public non-static methods.

Variable naming

- **Non-static variables** are named in the same way as other non-static methods.
- **Static variables** are named in the same way as static methods.

Constant naming

Constants are named in the same way as static variables.

C.5. Concrete implementation of the hybrid clustering

The real code of the application was written in the language *C#*, which has many things in common with *Java* (especially the syntax), as well as *C++* (e.g. virtual classes).

C.5.1. Main clustering method

As illustrated by figure C.3 the clustering method *CreateClusterSet()* is defined as *virtual*. This is important with respect to inheritance. When a class extends the class which contains the hybrid clustering algorithm it can override this method. But in this case, the correct method is always called. This is very important regarding dynamic binding, as depicted by the following example:

Assume the class containing the hybrid clustering algorithm was called *HybridClusteringAlgorithm* and the clustering method *CreateClusterSet()* was not defined as *virtual*. Assume also another class *AnotherHybridClusteringAlgorithm* would extend the class *HybridClusteringAlgorithm* and override the method *CreateClusterSet()*. This seems to be correct so far. The problem arises once the main routine instantiates the new class with the statement

HybridClusteringAlgorithm algorithm = new AnotherHybridClusteringAlgorithm();

This will not lead to a compiler error, since the class *AnotherHybridClusteringAlgorithm* subclasses *HybridClusteringAlgorithm*. But when it calls the method *CreateClusterSet()* it will call the method from

the class *HybridClusteringAlgorithm* and not the correct method provided by the class *AnotherHybridClusteringAlgorithm*.

```
virtual public AbstractClusterSet CreateClusterSet()
{
        StandardCluster root = new StandardCluster("");
        foreach (String s in this.elements)
        {
                root.AddLeaf(s);
        }
        clusterDeep(root, 0);
        StandardClusterSet set = new StandardClusterSet();
        set.RootCluster = root;
        return set;
}
```

Figure C.3.: The implementation of the main routine of the hybrid clustering.

The return of the algorithm is a class called *StandardCluserSet*, which subclasses *AbstractClusterSet*. This is due to of the structure of the application. The clustering algorithm is not a part of the main application but rather a part of a plug-in bundle for the framework. The framework only defines the class *AbstractClusterSet* and it is up to the plug-in how a cluster set looks like in detail. In this case, a standard cluster set is used consisting of a set of sub clusters (standard cluster sets again), or, if it is at the lowest level of the tree, a set of leaf objects which refer by their name to the raw data.

Whenever a cluster set is of the class *StandardClusterSet* or a subclass of it, all the displays which accept this class (and thus also its subclasses) can display it. It is up to the developer to write new classes of cluster sets (e.g. a special cluster set class for phylogenetic trees), he only has to subclass *AbstractClusterSet*.

The field *this.elements* contains a collection of strings. Each of these strings identifies the identifier for an object, e.g. a gene or protein. This is therefore the set which is to be clusterd. The method calls then the hybrid clustering method, which delivers the clustering. It has to be noticed that this is the public clustering method, which implements the method from the interface *IClusteringAlgorithm* of the framework. It does not say anything about the character of the clustering algorithm, it only has to return an instance of the class *AbstractClusterSet*.

It is also possible to write plug-ins for the framework providing another clustering algorithm, e.g. a standard non-hierarchical algorithm like *k-means* clustering. In this case the concrete return class may not be of the class *StandardClusterSet*, the developer can define a new class modelling a non-hierarchical clustering. This can also have different representations, for

instance representing a 2d-projection of the clusterings including their centroids. It is again up to the developer to write different visualisations which can display those kind of clusterings.

C.5.2. Hybrid clustering method

The hybrid clustering algorithm gets the parent cluster which contains the set of elements which should be clustered. It calls the clustering algorithm and parses the correct similarity matrix to it, depending on which feature the current round focuses. The result will be a set of clusters. It deletes the leaf nodes of the parent cluster and adds all the clusters from the computation as new sub clusters to the parent cluster. Each of these sub clusters is then clustered again, by iterating the depth (according to the next feature) and calling itself recursively. Figure C.4 shows the implementation of the hybrid clustering.

```
private void clusterDeep(StandardCluster parentCluster, int depth)
{
        if (depth < this.rounds)
        {
                StandardCluster temp = clusterOneRound(
                        parentCluster.Leafs,
                        this.similaritySet[depth],
                        this.threshold[depth],
                        this.smallIsSimilar[depth]);

                parentCluster.ClearLeafs();

                foreach (StandardCluster cluster in temp.SubClusters)
                {
                        parentCluster.AddSubCluster(cluster);
                        clusterDeep(cluster, depth + 1);
                }
        }
}
```

Figure C.4.: The implementation of the hybrid clustering part.

C.5.3. Underlying graph clustering algorithm

In contrast to the pseudo code version it has to be noticed that similarity sets can have a different character. In similarity sets computed with *Blast* low values indicate high similarity while for similarity scores reflecting other features may indicate high similarity when the value is high (for instance *Pearson correlation*). The algorithm hence calls a method which is named *similarityOK(double, double, bool)*. The first parameter contains the similarity

score, the second parameter contains the threshold and the third parameter indicates whether small values or high values indicate high similarity.

Furthermore a default value is used which plays an important role. When computing similarity sets some applications cut off values which violate a threshold value, meaning that the score is so bad that it is discarded. This is done for space efficiency reasons. For instance if a set of $n = 1000$ sequences is compared by *Blast*, this leads to $\frac{1000 \cdot (1000-1)}{2} = 499500$ pairs respectively similarities, but many of them are irrelevant. The scores which are worse (meaning higher for *E values*) than a certain threshold, for instance $e - 1 = 0.1$, are discarded (meaning they will not appear in the *Blast* output). Consequently, the application has no similarity values for some pairs at hand. This is solved by the *defaultValue*, which is either set to *infinity* or 0. If there is no similarity for a pair at hand, the value is then set to *infinity* if small values indicate a high similarity, otherwise to 0. The implementation of the algorithm is shown in figure C.5

The algorithm uses a hashtable *visited<String, bool>* which is an instance of the generic class *Dictionary<N, M>* for keeping track of the nodes which have already been visited. The keys are objects of the *String* class and the values are of the type *bool*. The key refers to the identifier of the element which is represented by a node for the DFS graph search. The value indicates whether the node has already been visited (meaning that it is or was on the stack) or not. The algorithm starts a depth-first-search from each node (element) which has not already been visited. It adds all the nodes (elements) it could reach from this initial node to a cluster. After finishing, it adds the cluster as a sub cluster to the root cluster. Then it re-starts the search from a node which has not already been visited, until all the nodes are visited and assigned to a cluster.

```
private StandardCluster clusterOneRound(
        IEnumerable<String> clusterElements, StandardSimilaritySet set,
        double threshold, bool smallIsSimilar)
{
    Dictionary<String, bool> visited = new Dictionary<String, bool>();

    foreach (String element in clusterElements) {
        visited.Add(element, false);
    }

    Stack<String> dfsStack = new Stack<String>();
    StandardCluster rootCluster = new StandardCluster("");
    StandardCluster subCluster;
    double defaultValue;

    if (smallIsSimilar) defaultValue = double.PositiveInfinity;
    else defaultValue = 0;

    foreach (String element in clusterElements) {
        if (!visited[element]) {
            subCluster = new StandardCluster("");
            dfsStack.Clear();
            dfsStack.Push(element);
            visited[element] = true;

            while (dfsStack.Count >0) {
                String current = dfsStack.Pop();
                subCluster.AddLeaf(current);
                foreach (String element2 in clusterElements) {

                    if ((!visited[element2])
                    && (!element2.Equals(current))
                    && similarityOK(
                            set.GetSimilarity(current, element2, defaultValue),
                            threshold,
                            smallIsSimilar))
                    {
                        dfsStack.Push(element2);
                        visited[element2] = true;
                    }
                }
            }
            rootCluster.AddSubCluster(subCluster);
        }
    }
    return rootCluster;
}
```

Figure C.5.: The implementation of graph clustering algorithm.