

Adding robustness and scalability to existing data mining algorithms for successful handling of large data sets

Written by
M.G. van der Zon

Under supervision of
Prof. Dr. J.N. Kok
Dr. W.A. Kosters

This thesis is submitted for obtaining the degree Master of Computer
Science at the Leiden Institute of Advanced Computer Science (LIACS)

Leiden University

22 December 2006

Abstract

In this master thesis we focus on the robustness and scalability of two existing data mining algorithms. In the first part we show how an existing algorithm for clustering analysis in Self-Organizing Maps is scaled to a computer grid. This existing approach is made more flexible and robust, such that any generic data set can be used. Experiments show how publications are clustered, based on their abstract, with a set of selected keywords. In the second part of this thesis we introduce a method that can mine a very large genetic data set, using existing biological software packages. This method tries to identify genes involved in longevity, using association analysis with haplotypes. For both parts we will show the achieved results in an interactive visualization tool that can be launched from a web browser.

Contents

Introduction	5
I VisualTreeSOM	7
1 TreeSOM Toolset	8
1.1 Self-Organizing Maps	8
1.2 Cluster discovery in SOMs	11
1.3 SOM as a tree	14
1.4 The most representative SOM	15
1.5 TreeSOM tools	16
2 VisualTreeSOM: Visualization of TreeSOM	18
2.1 Problem description	18
2.2 Solution basis	19
2.2.1 Newick tree format	19
2.2.2 PHYLIP package	20
2.2.3 Tree data structure	21
2.2.4 Tree visualization	22
2.3 Implementation	23
2.3.1 The Java programming language	23
2.3.2 Java Web Start	24
2.3.3 SWT: The Standard Widget Toolkit	25
2.3.4 User interaction	26
2.4 Putting it together: VisualTreeSOM	27
2.5 Summary	31
3 Results of VisualTreeSOM	33
3.1 Data set	33
3.2 Keyword Analyzer and Export Tool	34
3.3 GridSOM: SOM on a grid	37
3.4 Results	37

3.4.1	Experiment 1	38
3.4.2	Experiment 2	38
3.5	Summary	39
II	Longevity	41
4	An Introduction to Genetic Analysis of Longevity	42
4.1	Introduction	42
4.2	Problem description	43
5	Association Analysis using Haplotypes	47
5.1	Solution basis	47
5.1.1	Haplotypes	48
5.1.2	Data set	48
5.1.3	Haplotype Reconstruction	51
5.1.4	Haplotype Pattern Mining	52
5.2	Results	54
5.2.1	Preprocessing	54
5.2.2	Haplotyping	54
5.2.3	Frequent pattern mining	55
5.3	VisualSNP	56
5.3.1	Visualizing pattern files	57
5.3.2	Visualizing marker files	58
5.4	Summary	60
III	Conclusion	62
	Conclusion	63
	Acknowledgements	65
	Bibliography	66
IV	Appendices	68
A	Screenshots experiment 1	69
B	Screenshots experiment 2	72

Introduction

The main purpose of data mining techniques is to find hidden information and unknown relations within an amount of data. The size of the data increases vastly, but useful information extracted from the data seems to be decreasing. In order to comply with future demands, new data mining algorithms and concepts have to be developed that can handle the growing data sets and extract more sophisticated information. Existing software packages should be extended, by adding robustness and scalability, in order to successfully handle the large data sets. This last issue will be the main theme of this thesis as two existing software packages will be discussed and extended.

Data mining involves a broad range of different algorithms to accomplish different tasks. Each algorithm attempts to fit a model to the data. The chosen model is dependent on the characteristics of the data and the task that has to be performed. Roughly, the used data mining models can be either *predictive* or *descriptive* in nature. A predictive model makes a prediction about values of data, using historical results found from different data. A descriptive model identifies patterns or relationships in data, by exploring the properties of the examined data. In this thesis we will only concentrate on the descriptive model and in particular on the sub-classes: *clustering* and *association rules*.

Clustering maps data into several groups, also referred to as clusters, such that similar objects are grouped together. The resulting groups are not predefined, but rather defined by the data alone. The clustering is accomplished by determining the similarity among the data on predefined attributes or properties. Machine learning typically regards clustering as a form of unsupervised learning.

An association rule is a model that identifies specific types of data associations. Using this model, relationships among data items can be uncovered as they co-occur frequently within in the data set. It reduces a potentially huge amount of information to a small set of statistically supported items.

This thesis is divided into four parts, where the first two parts are of a descriptive nature. The third part consists of the conclusion, including the discussion and recommendations for future work, acknowledgements, and the bibliography. The fourth and final part contains the appendices of this thesis.

The first part of the thesis consists of three chapters, wherein an existing data mining method is made more robust and is scaled to a computer grid. This existing data mining algorithm consists of a set of tools for clustering analysis in Self-Organizing Maps (SOMs). In the first chapter this toolset is described in detail, including the fundamental outline and principles of this methodology. The tool responsible for training the SOMs is stripped and scaled to a computer grid. After such a map is trained it can be used for cluster analysis. This cluster analysis can be exported to a tree structure incorporating several clustering levels of the same SOM. This tree is displayed by a visualization tool, which shows the tree in an interactive manner. In the second chapter the focus lays on the visualization of this extended method and in particular the implementation of the working solution. The last chapter of this part shows the results acquired by this method and in particular how the results are achieved.

The second part of the thesis consists of two chapters, wherein an innovative method is introduced to mine a data set of 450 million Single Nucleotide Polymorphisms (SNPs). This complex medical analysis is part of a study in the fields of ageing and longevity performed at the Leiden University Medical Centre (LUMC). The overall goal of this study is to identify genes involved in longevity. The first chapter of this part introduces this medical study and familiarizes the reader with the subject by presenting some biological terms. In our analysis the real challenge lays in the quantity and process of the SNPs. The data consists of the measurement of 500,000 genetic variations in 900 subjects, comprising 450 million data points. In the second chapter we present a method that tackles this problem and which is based on association analysis using haplotypes. The initial data is split based on the chromosome number and is haplotyped with an existing software package. The haplotyped data is searched for frequent patterns with another existing software package. The output of this software package is visualized using a tool, which can be easily used by biologists in order to localize and identify the susceptible SNPs.

Part I
VisualTreeSOM

Chapter 1

TreeSOM Toolset

In this chapter we will introduce the TreeSOM [1] toolset, including the basic outline and principles used by this method. This chapter starts with describing the fundamental algorithm concept used in TreeSOM: Self-Organizing Maps (SOMs) [2]. After this concept is introduced the following section explains how this architecture can be used for cluster analysis and especially for cluster discovery. In the successive section it is explained how the SOM can be represented as a tree. Section 1.4 shows how TreeSOM selects the most representative SOM from a constructed set of SOMs. This chapter is concluded with the description of the various tools contained in the TreeSOM toolset.

1.1 Self-Organizing Maps

The TreeSOM toolset is merely based on an algorithmic approach in the field of neural networks, called *self-organizing map (SOM)*, sometimes referred to as *self-organizing feature map (SOFM)* or *Kohonen map*. As the last name implies this approach was first described by Kohonen in the beginning of the 1980s.

The SOM is a competitive unsupervised learning approach, based on a grid of nodes (also referred to as *neurons*) which are connected by direct links. Learning is based on the concept that behavior of a node should impact other nodes and links in its local neighborhood. Weights above each link are initially assigned randomly and adjusted during the learning process to match input vectors in a training set. In each learning step a single node in the grid is selected, which weight vector is closest to the vector of inputs, introducing competition between nodes. The weight of this selected node is adjusted to make it closer to the input vector and also the weights of the

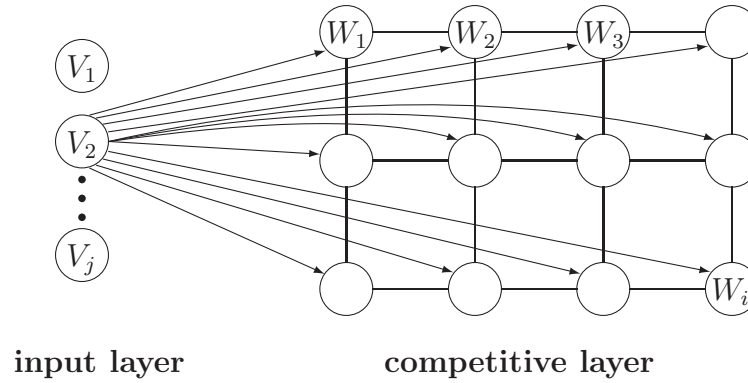


Figure 1.1: Schematic overview of a Self-Organizing Map

neighboring nodes are adjusted, although not so effective as the selected node.

In Figure 1.1 a schematic overview of the network is given, where on the left side the input layer is displayed and on the right the competitive layer. The nodes in the input layer consist of input vectors, containing multi-dimensional data per vector. Each input node is connected to each node in the competitive layer. The competitive layer can be viewed as a two-dimensional grid of nodes, which produces output values that compete. In this example a 4×3 grid creates twelve outputs from which the “best” one is chosen. “Best” is determined by computing a distance measure between the input vector V_j and the *weight vector* W_i for node i , where the dimension of the input vector is equal to the weight vector. TreeSOM uses the Euclidean distance by default, but it can also use the chi-square distance.

Training occurs by adjusting the weights so that the best output is even better the next time the same input is used. The basic outline of the algorithm used for training is displayed below.

Algorithm 1 Training algorithm for a Self-Organizing Map

Procedure Train (for Self-Organizing Map M and input vectors V of dimension k):

- Initialize the weight vector W_i for each node i inside map M to random values
- For a large number of iterations T , repeat:
 - Apply an input vector V_j from all available j input vectors V to SOM M
 - Select node i^* as winner if its weight vector W_{i^*} is closest to the current input vector V_j , using distance measure:

$$\sqrt{\sum_k (W_i^k - V_j^k)^2} \quad (\text{Euclidean distance}), \text{ or}$$

$$\sqrt{\sum_k \frac{1}{V_j^k} \left(\sum_\ell W_i^\ell - \sum_\ell V_j^\ell \right)^2} \quad (\text{chi-square distance})$$

- Update the winners node i^* and its local neighborhood according to the update rule:

$$\Delta W_i = \alpha(t) \Lambda(i, i^*, t) (W_i - V_j)$$

where

- $\alpha(t)$ is the learning rate of the SOM at iteration t
- $\Lambda(i, i^*, t)$ is the neighborhood function based on a Gaussian function

The first step in above depicted algorithm is to randomly initialize the elements of the weight vectors with values between the minimum and maximum value of all elements of the input vectors. After this initialization the SOM is trained by repeating several steps for a large number of iterations. During a single iteration the input vector is selected cyclic from all input vectors, generating a uniformly distributed input. After the winning node is calculated, its neighborhood is updated accordingly. The update rule uses a learning rate and a neighborhood function which both linearly decrease over time. This means for the learning rate that a node will be more “corrected” to the input in the beginning than in the end. For the neighborhood function this implies that the impact radius of a winning node will be greater in the beginning than in the end. The neighborhood function determines how much the elements of node i are changed during the updating of the weight vectors at iteration t .

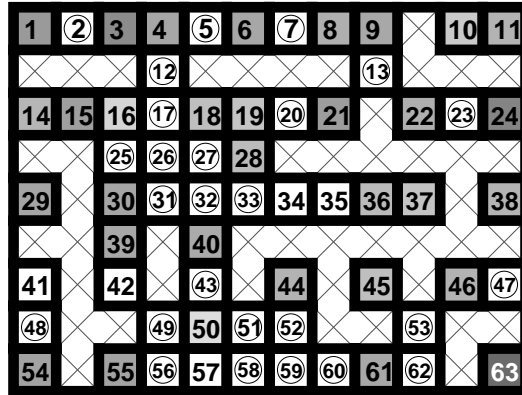


Figure 1.2: SOM clustered at distance threshold 0.16404

During the training process the map organizes itself by ordering the nodes inside the map into clusters based on similarity between them. Those nodes that are closer together are more similar than those that are far apart. After the training phase is finished the mapping process takes place. In this second phase each input vector will be classified to the node in the map, whose weight vector is the closest to the input, based on the same distance measure used in the training phase.

1.2 Cluster discovery in SOMs

In the previous section it is described how a SOM can map high-dimensional data onto a two-dimensional space by placing similar elements close together, forming clusters. So we can define a *cluster* as a group of nodes with short distances between each other and long distances to the other nodes. A distance threshold is used to specify the maximum distance between two adjacent nodes inside a single cluster, defining a clustering level. Changing this threshold will yield different clusterings of the same SOM, where a small threshold corresponds to the most specific clustering and a large threshold corresponds to the most general clustering.

In Figure 1.2 a SOM containing 108 nodes in a 12×9 grid is displayed. This figure shows the most specific clustering, at a threshold of 0.16404, of the 124 data elements mapped on the nodes inside the depicted SOM, forming 63 clusters. Each data item is mapped on the node, which is the most similar to it. Thus some nodes can contain many data elements and others none at all. Nodes that do not have any elements assigned to it are crossed out in this figure.

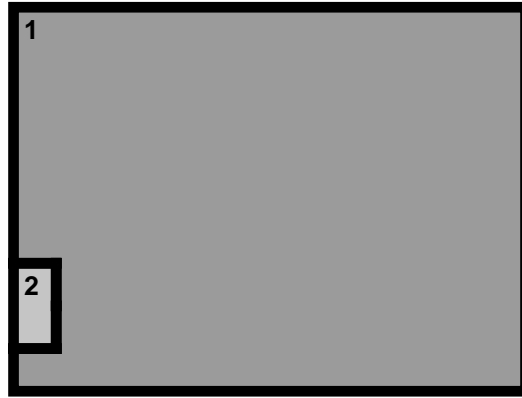


Figure 1.3: SOM clustered at distance threshold 0.61569

Each cluster consists of a number of adjacent nodes in the map, where the cluster borders are displayed in black. The area inside a cluster is shaded and represents the average distance between the data elements mapped within the cluster. In this shading white indicates zero distance (identical elements) between data elements and black the largest distance between any two elements in the data set. Clusters containing only a single mapped data element are marked with a circle and have a white shaded area, since the distance of any data element to itself is zero.

As mentioned in the beginning of this section, different cluster maps are generated for different thresholds. Normalizing the distances between any two adjacent nodes, such that the largest distance equals 1, we can use distance thresholds as values between 0 and 1. The most specific clustering, where the initial mapping is displayed, is already discussed in the previous paragraph. In Figure 1.3 the most general clustering is showed, generated with a threshold of 0.61569. At this cluster level only two clusters are constructed, where the second cluster consists of the merged clusters 41 and 48 from the initial cluster map in Figure 1.2. The first cluster in this figure is shaded more grey that the second cluster, which implies that it has a worse distance density. Also this figure shows the contours of the subclusters contained in each cluster.

In Figure 1.4 the cluster map generated from threshold 0.38028 is depicted. This threshold lies roughly in between the most specific and most general clustering. At this clustering level, the contour of the general cluster (8) is already visible. Almost half of its contained subclusters are single mapped clusters, which are displayed with circles in the initial Figure 1.2.

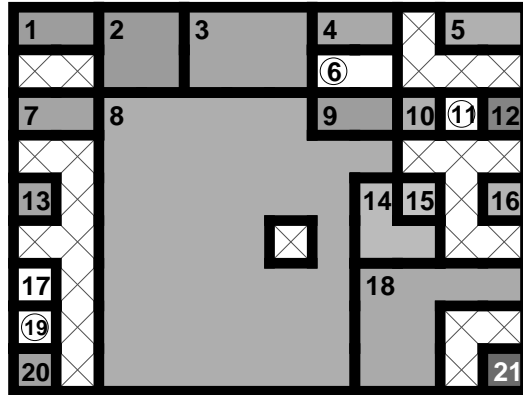


Figure 1.4: SOM clustered at distance threshold 0.38028

The algorithm used for cluster discovery in a trained SOM for a given distance threshold is displayed below, as described in [1].

Algorithm 2 Cluster discovery algorithm for a given distance threshold

Procedure Cluster-Discovery (for distance threshold T):

- Mark all nodes as unvisited
- While there are unvisited nodes, repeat:
 - Locate an arbitrary unvisited node N
 - Start a new cluster C
 - Call procedure *Cluster* for N , C and T

Procedure Cluster (for node N , cluster C and distance threshold T):

- Assign N to C
 - Mark N as visited
 - For each unvisited node A adjacent to N such that the distance $|NA| < T$ call procedure *Cluster* for A and C
-

This algorithm first marks all nodes inside the map as unvisited. After all nodes are marked, an unvisited node is selected and is assigned to a newly constructed cluster. For each unvisited node adjacent to the selected node, it is checked recursively if the distance between the nodes is smaller than the threshold. If this is the case, the node is added to the cluster and the node is marked as visited. The distance between two nodes is defined as the average distance between the data vectors contained in both nodes. This algorithm produces a cluster map, where each node inside a cluster is connected to

at least one other node within this cluster with a distance smaller than the threshold. Note that distances between nodes inside the same cluster can be greater than the distance threshold.

Above visualizations of a cluster map give in a clear view a strong insight in the different clusters on a SOM. The area of a cluster in combination with its shade gives a strong indication about the data density and size of the cluster. In order to get an insight in how the flow of clustering goes from the most specific clustering to the most general clustering, a series of cluster maps can be generated from successive thresholds. These generated cluster maps can be converted into an animated graphical format, showing dynamically the cluster formation on a SOM.

1.3 SOM as a tree

The above described cluster analysis can produce cluster maps for different distance thresholds on a SOM. Using successive thresholds the cluster formation can be shown, where one or more clusters are merged together to a single cluster. This cluster formation on a SOM can be represented as a tree structure as follows.

At each clustering level, with a corresponding scaled threshold between 0 and 1, a cluster can be represented as a node inside the tree structure. In successive clustering levels these clusters will be merged together in a new cluster, which is represented with a new node located further up in the hierarchical tree structure. This new node contains several nodes further down in the tree structure, which correspond to the subclusters in the cluster map.

The root node in the tree structure corresponds to the most general clustering, i.e., 1 cluster, containing all nodes from the cluster map. This is the case between the maximal distance threshold and 1 (for the cluster maps in the previous section this is between 0.61569 and 1). The most specific clustering, between 0 and the lowest distance threshold, shows the individual elements as leaves contained in the corresponding cluster, represented as nodes, inside the tree structure. In the previous section this most specific clustering is between 0 and 0.16404.

In this tree structure, further referred as *cluster tree*, the sum of all branch lengths on the path from the root node to the leaves is the same for each path, and equals the difference between the maximal and minimal threshold

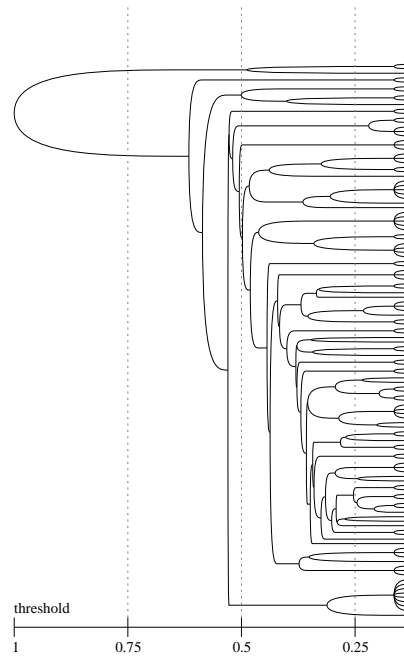


Figure 1.5: Cluster tree of the calibrated SOM depicted in Figures 1.2, 1.3, and 1.4

values: $\Delta_t = t_{max} - t_{min}$. The branch length between a parent node and its child node indicate the additional distance necessary to merge this subcluster to the cluster represented by the parent node. The cluster tree corresponding to the cluster maps depicted in the previous section is shown in Figure 1.5 in a hierarchical representation. Note that a cluster tree is constructed from a large number of cluster maps.

1.4 The most representative SOM

A self-organizing map is randomly initialized and trained with data elements presented in random cyclic order. Thus, different SOMs are constructed for different random seeds. This directly influences the cluster map, where both the most specific clustering as well as the cluster formation may be different. In order to determine the “true” clustering a large number of SOMs has to be produced and their corresponding cluster trees must be analyzed.

From the large number of cluster trees an average or *consensus* tree is constructed using an external tool. This process is described in more detail in Section 2.2.2 of the next chapter. This consensus tree is used to select a single cluster tree from the input SOMs as the best representative of the

consensus. In order to select this best cluster tree, the algorithm below, as described in [1], is used.

Algorithm 3 Distance measure for cluster trees

Procedure Distance-Measure (from reference tree R to tree T):

- Initialize score to 0
- For each node N of R repeat:
 - Search T for a node N_T equivalent to N_R (use *Node-Equivalence-Test* for nodes N_T and N_R)
 - If found, increment the score
- Define similarity of T with respect to R : $S = \frac{\text{score}}{|R|} \in [0, 1]$ where $|R|$ is the number of nodes in R
- Define distance from R to T : $D = 1 - S \in [0, 1]$

Procedure Node-Equivalence-Test (for node A and B):

- Define a set of leaves of a node N to contain all the leaves directly connected to N and all the leaves connected to any of its descendants
 - Nodes A and B are equivalent if their sets of leaves are identical
-

For each input tree the distance is measured to the consensus tree. The cluster tree, which is the most similar to the consensus tree, is selected as the best representative. The similarity is defined as the number of nodes in the input tree, which are “equivalent” to nodes located in the consensus tree, divided by the total number of nodes inside the tree. Two nodes are “equivalent” if the set of the contained leaves inside both nodes are identical.

1.5 TreeSOM tools

In this chapter we described the basic outline and principles used by the TreeSOM toolset. In this final section we will discuss the tools bundled in this software package. The TreeSOM software package consists of the following tools, as described in [3]:

som is meant to train a map from scratch using a set of parameters, located in the **som.conf** configuration file. A trained SOM may then be analyzed for clusters, visualized or saved as a tree file.

clustermap is used for analyzing already trained SOMs. It does the same as **som**, except it can not train a map.

clustertree does tree visualization and calculates distance between trees. Trees can be drawn in a variety of ways: including hierarchical and circular representations.

matrix calculates a distance matrix from the given data file.

The package contains also the shell scripts **manysom** and **mkgif**. The first script is for training many SOMs with the same data and configuration file, but with different random initializations. The second script is used to construct an animated image out of the cluster map visualizations to show cluster formation as a dynamic cluster tree.

Chapter 2

VisualTreeSOM: Visualization of TreeSOM

In the previous chapter we described the TreeSOM toolset in detail. In this chapter the focus lays on the visualization of the TreeSOM toolset and in particular the cluster trees. This chapter starts with describing the problem. After that section the solution basis explains how the introduced problem can be tackled and what kind of data structures are used. The implementation section gives an overview and description of the tools used in order to get a working solution. In Section 2.4 the working solution is described and elaborated with some screenshots. This chapter ends with a short summary, giving an overview of the achieved results and some suggestions for future research.

2.1 Problem description

The TreeSOM toolset, as described in the previous chapter, contains the tools **som** and **clustermap**. Both tools can analyze SOMs for clusters and output this cluster analysis as tree files. For each SOM such a single cluster tree file is produced and saved in Newick tree format. Further details of this Newick tree format can be found in Section 2.2.1.

The visualizations produced by the TreeSOM tools can draw single cluster trees in different ways: as a traditional hierarchical tree representation and as an alternative circular tree representation. From these cluster trees an average or *consensus* tree is build. This consensus tree is used by TreeSOM to select an input cluster tree as the best representative of the consensus.

Unfortunately, TreeSOM does not include consensus tree tools for build-

ing a consensus tree, it can only draw such trees. Therefore the package PHYLIP is used to create a consensus tree from multiple cluster tree files. In Section 2.2.2 a more detailed explanation is given how this package builds such consensus trees.

A disadvantage of the visualizations produced by TreeSOM is that only a static image¹, such as Figure 1.5, can be created from a cluster tree. However, there exists an approach [4], where a consensus tree is visualized in a dynamic and interactive manner. In this approach multiple SOMs are produced by the TreeSOM toolset and from the corresponding cluster trees a consensus tree is built. This consensus tree is visualized in an interactive application, where for arbitrary clustering levels the consensus tree is displayed in a real-time manner. In this way the user is able to change the clustering level and can see almost instantly the result by means of the transformation of the tree.

2.2 Solution basis

In this section several ingredients that form the basis of solution to the problem introduced in the previous section are discussed. Combining these ingredients into a single application will achieve the stated goal. Only the important data structures and models that form the backbone of the application are discussed in this section. Detailed information about the working solution and implementation is in Section 2.3.

2.2.1 Newick tree format

Each trained SOM in the TreeSOM toolset that is analyzed for clusters can be saved in a so-called cluster tree file. This file complies with the Newick tree format [5]. An example of this format is as follows:

$$(A, (B, (C, D)), (E, F));$$

Each tree in the Newick format is represented by at least a pair of parenthesis, possible (nested) internal nodes and/or leaves, and ends with a semicolon. Each internal node is represented by a pair of matched parentheses and may contain other internal nodes or single leaves. A leaf is represented by its name, which can be any string of characters except blanks, colons, semicolons, parentheses and square brackets. Each leaf should have a unique name.

¹TreeSOM can only create an animated image from cluster maps

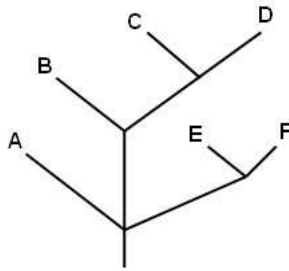


Figure 2.1: Traditional rooted tree

The example tree in the above paragraph is displayed as a traditional rooted tree in Figure 2.1. The Newick standard does not define unique representations of trees, because the order in which the descendants are located affects the representation of the tree. Other examples of the same tree would be:

```
((E, F) , (B, (C, D)), A);
(A , ((C, D), B), (E, F));
```

In the first case the siblings A and (E, F) are swapped, resulting in the same tree as depicted in Figure 2.1.

Branch lengths can be incorporated into a tree by adding a real number after each node, internal or leaf, preceded by a colon. This number represents the distance between a node and its ancestor. For example the tree described above can be represented with branch lengths as follows:

```
(A:0.75 , (B:0.1, (C:0.1, D:0.5):0.2):0.45, (E:0.25, F:0.25):0.5);
```

The above represented tree, including branch lengths, can be displayed as an unrooted tree as showed in Figure 2.2. In this figure the distances between nodes and their ancestors are neatly outlined. In our case this unrooted or phylogenetic tree structure is the default representation of a tree.

2.2.2 PHYLIP package

As mentioned in the previous chapter, each SOM produces a single cluster tree and the TreeSOM approach creates a large number of SOMs. From these input cluster trees a single representative is chosen, which is the most similar to the consensus tree. We use the PHYLIP package [6] to create the consensus tree from all input cluster trees.

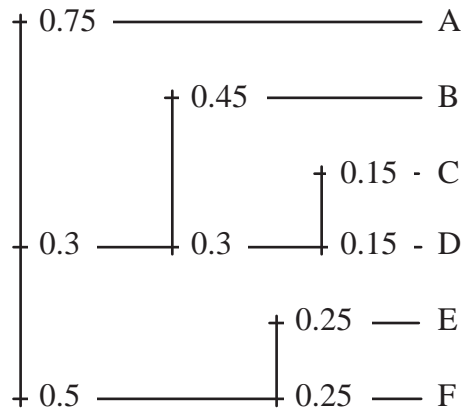


Figure 2.2: Unrooted tree including branch lengths

PHYLIP, the *PHY*Logeny *I*nference *P*ackage, is a package of programs for inferring phylogenies and consists of several tools. From this package we will use the **consense** tool to calculate the consensus tree. This tool computes from all cluster trees the consensus tree by the majority rule method. This method utilizes a simple majority of branches among the original trees. The consensus tree consists of all groups of nodes that occur more than 50% of the time, working downwards in their frequency of occurrence. Each leaf from the original trees should be contained inside such group; otherwise it is directly added to the internal root node.

Each internal node in a cluster tree can be viewed as a cluster, containing several leaves directly or indirectly via nested clusters. A leaf can be viewed as a single input instance of the trained SOM and corresponds to the most specific clustering. The root node corresponds to the most general cluster, containing all leaves. Such a hierarchical structure can be easily visualized at various clustering levels.

2.2.3 Tree data structure

In order to visualize the constructed consensus tree it needs first to be represented in some sort of data structure. For each matching parenthesis in the consensus tree file an internal node is created and for each leaf a leaf node. During the parsing of the tree, the nested internal nodes and leaves are first constructed before the current internal node is created. So the tree is built bottom-up and completed with the construction of the root node.

between node v and w , and $T(v)$ indicates the number of leaves contained in node v .

Advantages of a radial tree above a traditional rooted tree are a conveniently arrangement of nodes (especially for large data sets), the fact that the hierarchy is showed more explicitly and finally the decreasing amount of space needed to represent the tree.

The representation of the nodes from the radial tree is an extension to the previously introduced tree data structure. The following objects are added to the node structure:

- Wedge assigned to this subtree
- Total number of leaves contained in this subtree
- Angle of the wedge

The radial tree is constructed bottom-up from the source root node and finished with the construction of the radial root node.

There exist three different visualizations of the radial tree. In the first visualization all nodes are shown, where the internal nodes are painted gray and the leaves green. In the second visualization only internal nodes and cluster roots are shown, where again the internal nodes are painted gray and the cluster roots blue. In both visualizations the root node is painted black. The third visualization is the same as the first, expect that leaves from the selected family are painted purple. In Section 2.4 the different visualizations are illustrated with screenshots of the working solution.

2.3 Implementation

Now that all important elements which together form the solution basis are discussed, it is time to combine them into a single application. In this section an overview is given what kind of tools and program methodologies are used in order to get a working solution. Each of them is described in detail in the following subsections.

2.3.1 The Java programming language

As stated in the problem description our goal is to create a stand alone application that can be launched from a browser and is not dependent on the operating system platform. For this case Java is by far the most suitable programming language in comparison with other well-known high level

languages, such as the C variants. Java is cross-platform, since it runs on a so-called virtual machine which is available for a broad range of platforms. Secondly, almost every web browser incorporates the JRE, which could be useful to launch the application remotely. Another requirement is that the application should run and operate standalone. Therefore a Java Applet is not suitable for this task. However, Java has another strong alternative for an applet: Web Start.

2.3.2 Java Web Start

The Java Web Start technology [8] provides an easy and simple solution for launching full-featured Java applications with a single click. Users can launch applications without going through complicated installation procedures and Web Start works with any type of browser and any type of web server. From a programmer perspective it provides a robust and flexible solution for deployment, due to the fact that it automatically downloads all the needed files for the application and ensures that the correct JRE is installed, provided that the computer is connected to the internet.

Web Start is designed to make it easy for users to access and run applications remotely. The only thing that a user needs is a browser with an integration of a JRE version. As soon as the user clicks on the Web Start associated JNLP² file, an XML³-based file that tells Web Start how to run the application, the launching process is triggered. It first checks if the target machine has the correct JRE installed, which is specified in the JNLP file. If this is not the case, Web Start will automatically do a request to download the matching JRE and install this new version on the target machine. If the correct JRE is installed, the process continues with determining which resources are needed to run the application. This can be dependent on the target platform, so that different platforms might need different resources. When the needed resources are determined, Web Start checks if the resources are in its local cache saved from a previous run and if so, it checks if there are any application updates available. If the application is not in the cache or there are any updates available, Web Start will download the resources from the server. Now that all needed resources are available and up-to-date the application will be launched and run natively.

Web Start is not only easy for users, it is also easy for developers as

²Java Network Launching Protocol

³eXtensible Markup Language

deployment goes in the exact same way as for other Java programs. An application is deployed by one or more JAR⁴ files, which include all application resources. Developers do not have to bother maintaining consistency between various application versions and different JREs as Web Start automatically downloads the newly updated versions.

Deploying an application involves the following three steps: setting up the web server, placing the application packages on the web server and creating the JNLP file. The web server has to be configured so that the `.jnlp` files are associated with Web Start. This can be simply done by setting the JNLP extension to the `application/x-java-jnlp-file` MIME⁵ type.

The JNLP file specifies which packagers are needed, where these packages are located on the web server and indicates the main class of the application. It also specifies if the application may be launched offline, which saves bandwidth, and it sets some security settings. Each application runs by default in a restricted environment, similar to the Applet sandbox. If the application needs functionality beyond this sandbox, such as in our case where the user has to load the consensus tree from the local file system, then all JAR files need to be signed. The user will be prompted to accept the certificate for unrestricted access the first time the application is launched.

2.3.3 SWT: The Standard Widget Toolkit

As Java is selected to be the most suitable programming language in the previous subsections, we have several options for building the Graphical User Interface (GUI) of this application in this language. The standard option is AWT⁶, which is replaced by the JFC⁷ Swing packages as the new standard for building user interfaces. Both toolkits were developed and maintained by Sun Microsystems. Another option is to develop the GUI using the Standard Widget Toolkit (SWT) [9], which IBM developed for their Eclipse project and which became open source later. SWT is a Java class library that is designed to provide directly access to the user interface facilities of the operating system on which it is implemented.

The main difference between SWT and Swing is that SWT uses native widgets, giving SWT applications a native look and feel and a high level of integration with the desktop. Swing uses its own look and feel, which are the same on each implemented platform but on the other hand not comparable with the native ones. Therefore we decided to create our GUI using SWT.

⁴Java ARchive

⁵Multipurpose Internet Mail Extensions

⁶Abstract Windowing Toolkit

⁷Java Foundation Classes

A disadvantage of this choice is that for every supported platform a SWT library package has to be added as resource for our release. Fortunately, Web Start can determine the target operating system during the launch of the application and only the corresponding library is added as prerequisite resource needed to run the application.

2.3.4 User interaction

The user is able to interact with the application in several ways. The interactions that directly or indirectly involve with the transformation or visualization of the consensus tree are discussed in this subsection. The displayed tree can be transformed by changing the clustering, pruning or zooming levels. Using the mouse pointer some nodes of the tree can be highlighted or selected.

The clustering level, also referred to as clustering threshold, varies in the range from 0 to 1. This range is represented in the user interface by a slider, with which the user can select the desired value. The clustering threshold indicates the minimum value that must separate a leaf from a cluster root before it is merged to the cluster. At the minimum clustering level of 0 only the leaves mapped on the same node inside the SOM are clustered together. At the maximum clustering level of 1 all leaves are contained in a single cluster. The tree is then displayed as a full circle, where all leaves are at the border of the circle and the root node in the middle of it.

Like the clustering level, the pruning level also varies in the range from 0 to 1. This range is represented the same as with the clustering level, using a slider with equal length. The pruning threshold indicates the minimum branch length that must separate an internal node from its parent node in order to be displayed in the tree. If the distance between an internal node and its parent node is smaller than this pruning threshold, the leaves contained by the node will be pruned away by adding it to the parents node. Note that only internal nodes can be pruned away.

The difference compared with clustering is that the branch length of the traversed leaf is increased with the branch length of the disappeared node, constructing a more circular represented tree. At the minimum pruning level of 0 none of the internal nodes are pruned away. At the maximum pruning level of 1 all internal nodes are pruned away, creating a circular tree with wedges according to the current clustering level.

When a consensus tree contains a large number of nodes, the visualization can be quite complex due the fact that the nodes will be displayed close

together. Therefore a zoom level is introduced, which is represented in the user interface by a scale. The zoom factor varies in the range from 1 to 10. At the minimum zoom factor of 1 the whole tree is displayed in the boundaries of the canvas. Zoom factors above 1 will virtually enlarge the spanning of the tree so that the canvas only displays a part of the tree. If the zoom factor increases the nodes will only be displayed further away from each other. The nodes itself would not be enlarged.

If the user moves the mouse pointer inside the display of the tree, its location is being tracked. If the mouse pointer is for a short while above a node, this node will be highlighted. If the underlying node is a leaf, the highlight will consists of slightly enlarging this leaf and showing a tooltip⁸, containing the name of the leaf. The highlight of an internal node not only consists of slightly enlarging the node and showing a tooltip containing the number of leaves this node incorporates; it also highlights all incorporated nodes by painting them in a different color.

The user is also able to select a node by double clicking on it. If the selected node is a leaf, a new dialog will pop-up showing detailed information about this leaf. If the selected node is an internal node, the new dialog will show the clusters, including the contained leaves, according to the represented clustering level by this node.

If the clustering or pruning levels are changed, a new tree will be constructed using these new settings from the source root node. After construction, the new tree will be painted inside the boundaries of its canvas. If the zoom factor is changed or a node is selected, the tree will only be repainted accordingly.

2.4 Putting it together: VisualTreeSOM

In Figure 2.4 the main window frame of VisualTreeSOM, executed on the Windows platform, is showed. Each opened cluster tree gets its own tab inside this main window, wherein the upper part the tree is displayed and in the lower part the settings panel is located. This settings panel consists of several groups.

With the “Layout” radio button group the user can select its preferred tree visualization. The default selected tree layout shows both the internal nodes as the leaves of a tree. If the cluster layout is selected, only the cluster roots are painted. The family layout is the same as the default tree layout,

⁸a small textbox displayed on top of the current window contents

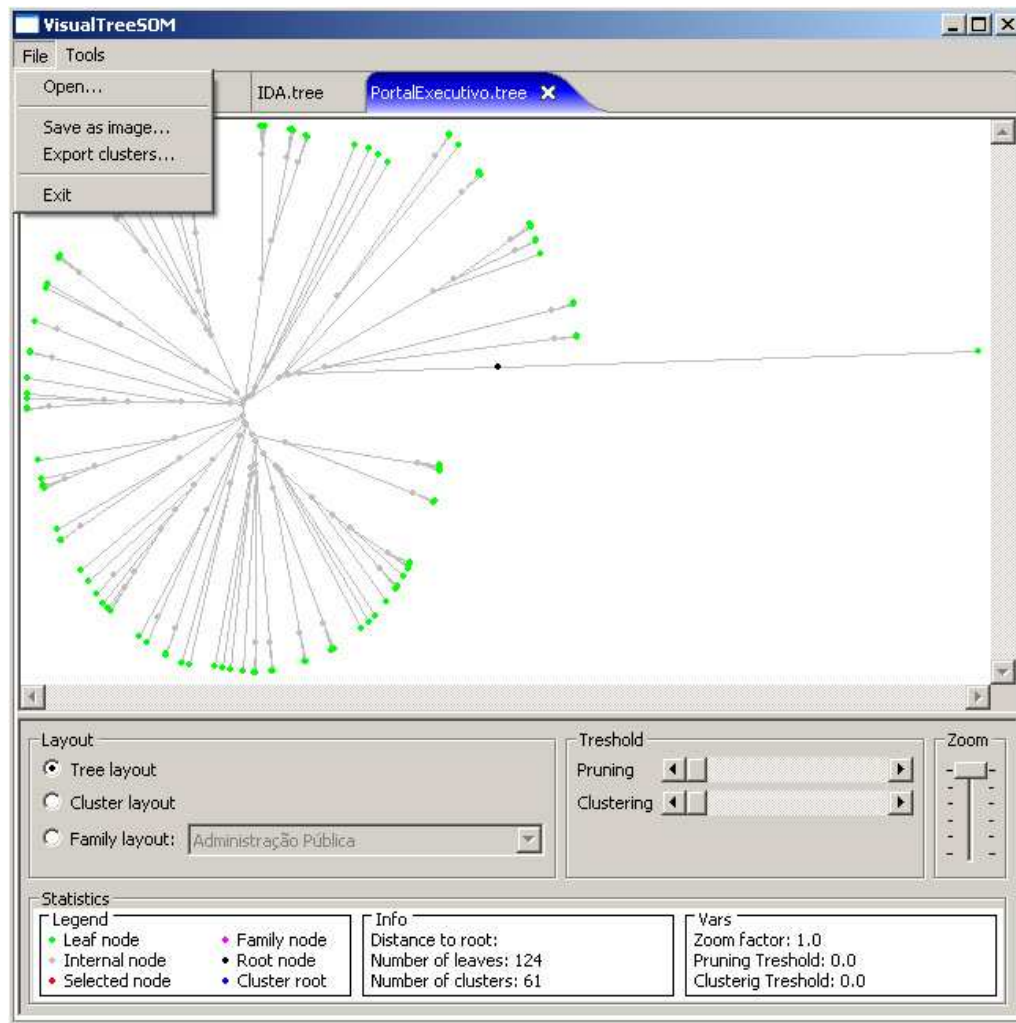


Figure 2.4: Main window frame of VisualTreeSOM on Windows platform

expect that the leaves of the family selected in the family combo box are painted with another color.

In the “Threshold” group the pruning and clustering thresholds are represented by sliders. Each slider ranges between the values of 0 and 1 in steps of 0.01. Both values are by default set to zero, indicating no clustering or pruning.

The “Zoom” group consists of a single scalar, representing the zoom factor. By default this scalar is set to 1, indicating no zoom, and the maximum zoom factor is 10. If the zoom factor is changed, the tree will be repainted accordingly and with the use of the horizontal and vertical bars the zoomed part of the tree can be observed in detail.

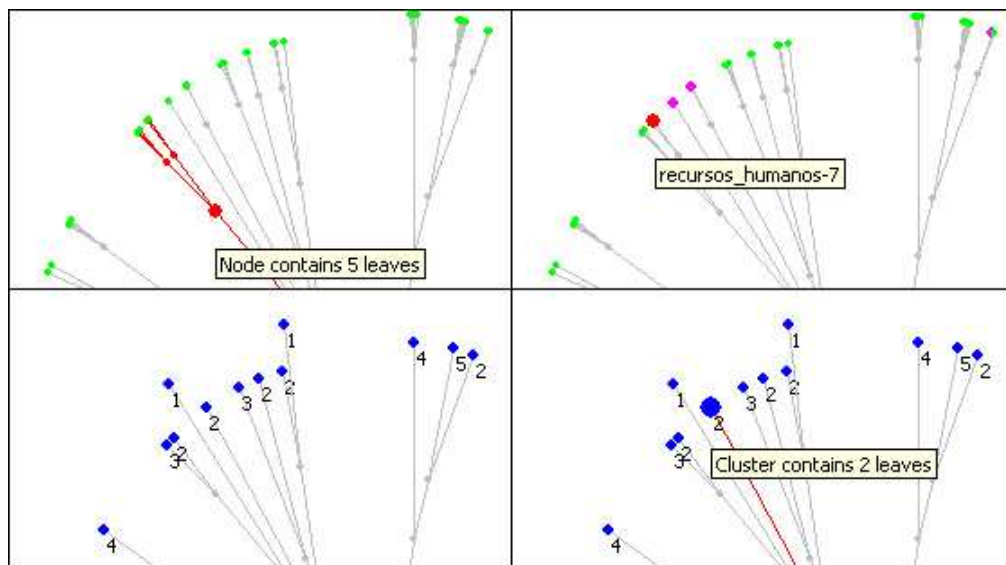


Figure 2.5: Four instances visualized with different layouts

The last group in this settings panel is the “Statistics” group, containing three subgroups. The left group is a legend, explaining the different used node colors with their corresponding meaning. The middle group shows some information about the currently displayed tree. It shows the number of leaves and clusters contained by this tree. If a node in the tree is selected, the distance from this node to the root is calculated and printed. The right subgroup displays the current values for the pruning and clustering thresholds, as well as the zoom factor. All displayed information will be updated if the tree is transformed to user interaction.

If a cluster tree is opened it is automatically checked for a corresponding information file in XML format. This file contains additional information for each leaf, such as the family name. If this file can not be found and the user would not load it, some features of the application will not be available. These features include the family layout and a stripped version of the export of the current clustering level. The user can create a snapshot of the current clustering by exporting the displayed tree to a XML file, where for each cluster the contained leaves with possible additional information is printed. Also the user has the option to save the currently displayed tree as an image.

The displayed tree can be visualized using different layouts. In Figure 2.5 three different layouts displays the same part of a tree. In the upper left corner the default tree layout is used and an internal node is highlighted. If a

2.4 Putting it together: VisualTreeSOM

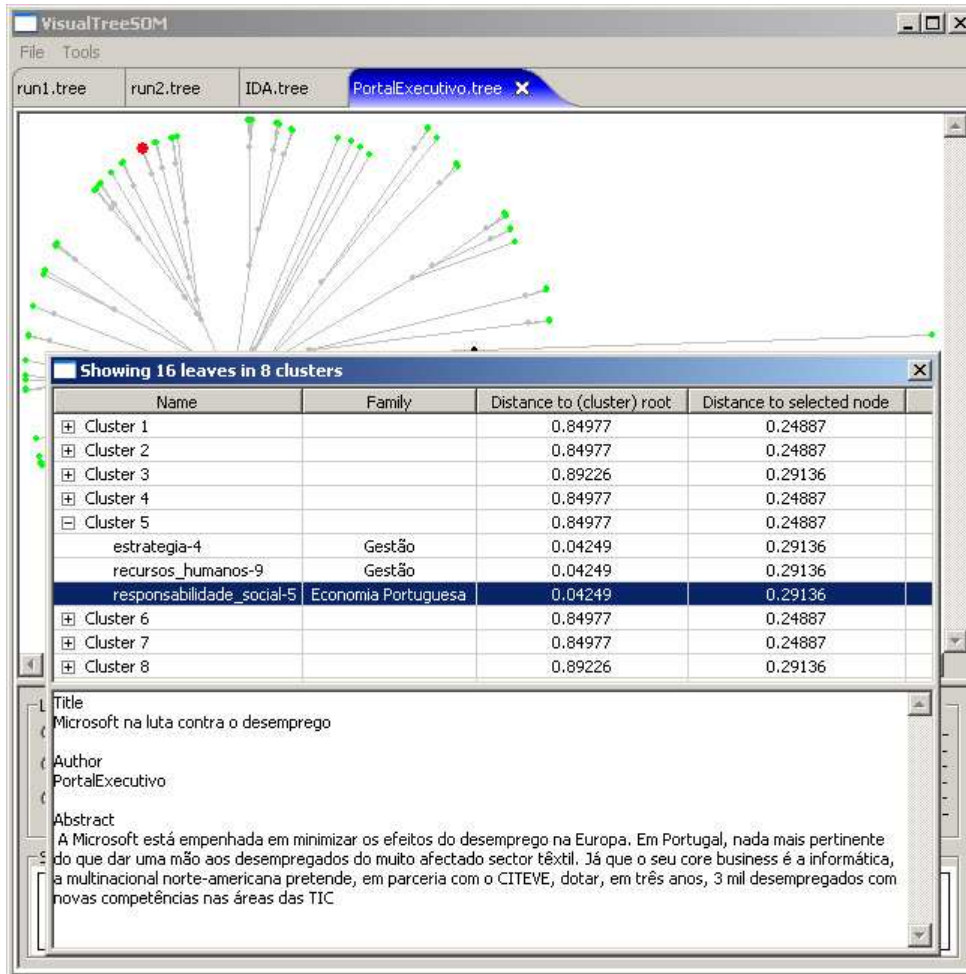


Figure 2.6: Popup window showing additional information

node is selected all internal nodes and edges contained directly or indirectly by this node is painted red. Also a tooltip shows the number of contained leaves, in this example it has 5 leaves.

In the upper right corner of Figure 2.5 the family layout is used. This layout paints all leaves corresponding to the same selected family in purple. Both in the tree and family layout the user can select a single leaf, where a tooltip shows the name of the highlighted leaf.

In the bottom of Figure 2.5 the cluster layout is used, where only internal nodes and cluster roots are showed. For each cluster root the number of leaves contained in the cluster is painted alongside the node. Also a cluster root can be selected as showed in the bottom right instance.

Not only can the user select a node of the tree, as seen in the previous figures, he/she can also click on them. By double clicking on a node, a new dialog will pop-up showing detailed information about this node. Such a new dialog is showed in Figure 2.6. In this new window all 8 clusters contained in the selected node are listed inside a tree table. The leaves inside the clusters are listed as subitems of their corresponding cluster roots. The table has four columns, displaying the following information:

- Name of node
- Family name of node (if available)
- Distance to cluster root
- Distance to selected node

For cluster roots the family name is not used and in the third column the distance to the root node is displayed in stead of the distance to the cluster root.

If a row inside the table is selected, the corresponding node inside the tree will be highlighted. In Figure 2.6 this is the case for the leaf ‘responsabilidade_social-5’. Also the additional information of this node is displayed in the lower part of the dialog. Note that for Figure 2.6 an internal node is pressed and that this tree is loaded with an additional information file. Also note that if a single leaf is selected inside the tree, only the single leaf is displayed in the dialog.

2.5 Summary

In this chapter an application is introduced that extends the TreeSOM toolset and which is based on an existing approach. In comparison with this existing approach, our application is made more flexible and robust, such that any generic data set can be used. The stand alone application has a native user interface and can be easily launched remotely by any browser.

The application itself displays in an interactive manner a consensus tree constructed from several cluster analysis files, calculated by the TreeSOM toolset. The user is able to interact with the application by changing some values, such as clustering or pruning levels, by which the visualization of the consensus tree transforms accordingly. Also the user can select a single node from the visualization, for displaying more detailed information about this selected node.

After the user performed some clustering analysis, the constructed clusters can be exported (along with the additional information) to a XML file. The corresponding displayed tree can be saved as image.

As future work, the import of cluster confidence can be a valuable addition. Such information reveals how homogeneous the cluster is, i.e., how similar the contained data elements inside the cluster are. This information can be added to the information dialog and it can be visualized inside the tree, by painting the internal nodes with greyscales based on the confidence.

Leaves belonging to the same family can be painted purple for each individual selected family. It could be an addition to assign each family its own color inside the tree. In the clustering layout the cluster roots can then be displayed with wedges, according to the number and type of families contained in the cluster.

Chapter 3

Results of VisualTreeSOM

In this chapter we will present some results acquired by VisualTreeSOM and in particular we mention how we achieved these results. This chapter starts with describing the data set used in the experiments and how this data is used for cluster analysis. The next section introduces a tool, which is part of the VisualTreeSOM application; it can transform the data, depending on user specified parameters, into a feasible input file for the TreeSOM toolset. The successive section shows how the TreeSOM toolset is scaled to a standalone application, running on a computer grid¹. In Section 3.4 the results from two experiments are shown as individual case studies. This chapter ends with a short summary and conclusion about the experiments.

3.1 Data set

In order to demonstrate the VisualTreeSOM application we use a text clustering example. Our goal is to cluster publications, based on their abstract, with pre-selected keywords. The keywords can be selected by the user in an interactive manner, as shown in the next section. The publications are provided by *Sociedade Portal Executivo*² in cooperation with the *Artificial Intelligence and Data Analysis Group (NIAAD)*³ from the University of Porto. Portal Executivo is a so-called Web portal⁴ and contains a huge set of articles, publications, papers and other information sources in electronic format, translated into Portuguese. These information sources are collected from respected magazines, such as *The Economist* and *Wired*; papers from big companies, such

¹architecture of multiple connected computers for performing large scale computational problems

²<http://www.PortalExecutivo.com>

³<http://www.niaad.liacc.up.pt>

⁴website containing a broad range of information for a specific group of users

as *Microsoft*, *Hewlett-Packard*, *Accenture* and *McKinsey Quarterly*; and published articles from universities. These information sources are categorized in a broad range of categories, such as technology, economics and tourism. The portal provides these information sources well-organized on their site to its paying customers.

The data set we will use in the experiments are taken from the *Management and Economy* category and consists of 124 publications. Each publication contains a title, author, abstract and a family. The family element indicates the subcategory wherein the publication is placed inside the main category by Portal Executive. Each publication belongs to one of following 13 subcategories: Editorship, Accounting, E-Business, E-Strategy, Ethics, Finances, Management, Innovation, Marketing, Operations, Human Resources, Social Responsibility and Information Systems. In this data set each subcategory is represented by ten publications, except Editorship, which is represented only four times, making 124 publications in total.

3.2 Keyword Analyzer and Export Tool

The clustering result of the experiments is merely dependent on the selected keywords used during the cluster analysis. Therefore VisualTreeSOM contains a tool to select keywords from the input data in a simple and clear interactive manner. Depending on several parameters the keywords are selected from the abstracts of all input publications. Note that only words inside these input abstracts can be selected as keywords.

When the tool is started the user has to specify the publications input file, which contains among other things the relevant abstracts. After this XML-based file is parsed and analyzed the main dialog is started. In Figure 3.1 the dialog of this tool, executed on the Linux platform, is shown. In this figure already some keywords are filtered and listed, complying with the minimum and maximum values of the three parameters. The dialog is horizontally divided into two panels. The left panel is responsible for informing the user about some statistics of the input data and manipulating the parameters to filter keywords from this data. The right panel shows these filtered keywords and can save or export the listed keywords.

The “Statistics” group in the left panel contains some information about the input data, which can help the user in choosing the different values for the parameters. As first item it shows the number of abstracts, which is equal

3.2 Keyword Analyzer and Export Tool

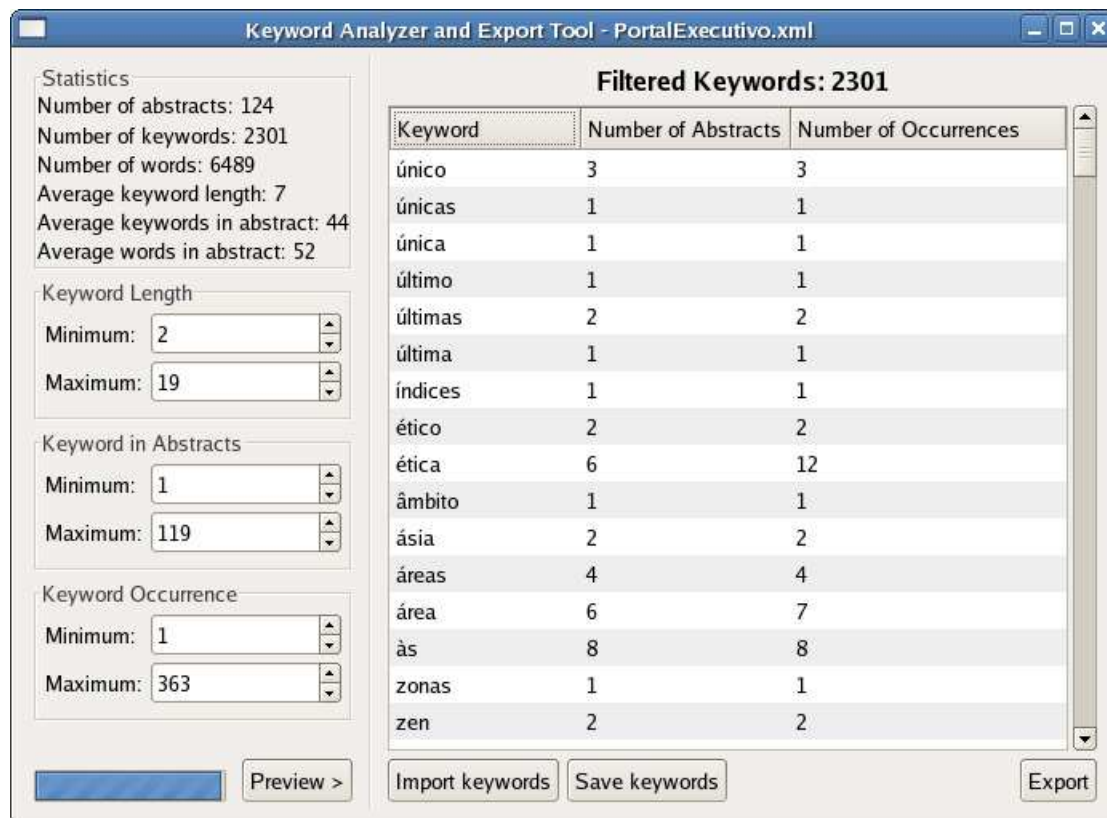


Figure 3.1: Keyword Analyzer and Export Tool

to the number of publications as each publication has only a single abstract. Secondly it shows the number of keywords in all abstracts, where a keyword is defined as a unique word. The number of words indicates the total number of words contained in all abstracts. This group is concluded with the average keyword length, average number of keywords in a single abstract and the average number of words in an abstract.

The groups “Keyword Length”, “Keyword in Abstracts” and “Keyword Occurrence” represent the three different parameters that the user can change for selecting keywords from the abstracts. Each parameter has a minimum and maximum value, where the maximum values are pre-calculated and comply with the input data. The first parameter indicates the keyword length, the the selected keyword has to satisfy. With the second parameter the user can specify the minimum and maximum number of abstracts the selected keyword must appear in. This parameter can be used to eliminate keywords that occur almost in every abstract. The third parameter specifies the minimum and maximum number of occurrences in all abstracts the keyword must

appear in. With this parameter the user can eliminate keywords that are frequently used throughout all abstracts, such as “the”, “in”, “and”, etcetera.

On the bottom of the left panel a progress bar and preview button is located. If the preview button is pushed, the input data is analyzed and the keywords complying with all parameters are selected. The progress bar shows the percentage of the already analyzed abstracts and when it reaches the end, the analysis is completed and all filtered keywords are listed in the table of the right panel.

The right panel consists of a table containing all filtered keywords and several buttons. Each selected keyword is listed in a single row inside the table, including the number of abstracts and the total number of occurrences the keyword has in the input data. The user is able to delete manually keywords from the table to get the user preferred set of keywords. This set of keywords can be saved in a XML-based file.

Not only the set of filtered keywords can be calculated from the parameters, it can also be loaded from a previously saved set of keywords. For each keyword loaded from the import file, the number of abstracts and number of occurrences are re-calculated. Such an import feature is useful for refining a previously saved set of keywords or to analyze different input data sets with the same set of keywords.

As the final set of keywords is chosen, they can be exported to a TreeSOM input-file with the following format:

```
4
0 1 1 2 item-1
2 2 1 0 item-2
0 1 1 0 item-3
```

The first line in the format states the dimension of the data vectors, which in the above example is 4. In our experiments this cardinality equals the number of selected keywords used to cluster the publications. Each subsequent line in the format represents one data vector, where the last element contains a unique label for the vector. In the above example three data vectors are shown; in our experiments this will be 124 vectors. Each publication is represented as a data vector, where each element indicates the number of occurrences the corresponding keyword has in the abstract of the particular publication. So if we use the above example in our experiments, we cluster 3 publications based on their abstract using 4 keywords. In the first publication the first keyword does not appear in the abstract. However the second and

third keyword appears both one time and the fourth keyword appears two times in the abstract.

3.3 GridSOM: SOM on a grid

After the set of selected keywords is exported to a TreeSOM input file, this file can be used for training self-organizing maps. To determine the “true” clustering of the publications, a large number of SOMs have to be trained. Such training can be a time consuming job, depending on the map-size, size of the data vectors and the number of iterations during training. Therefore we decided to scale the SOM training part of TreeSOM to a standalone application, which can be executed easily on a grid.

All tools from the TreeSOM toolset are offered as open source C++ implementations, so the SOM training could be reused in the new application. By stripping all superfluous features from the original source, a compact SOM training application could be constructed. This application, called *GridSOM*, needs as inputs the data vectors and a configuration file with several parameters used for training. It outputs a single cluster tree matching the successive cluster maps, constructed from the self-organizing map.

Each node in the grid can independently execute the GridSOM application, producing a single cluster tree file. So with the help of a grid, we can produce in a relative small amount of time a large number of cluster trees.

3.4 Results

In this section two case studies are described to show the results of the VisualTreeSOM application. For both case studies we use the data set introduced in Section 3.1. In Figure 3.1 at page 35 the statistics of this data set are displayed. We will use the GridSOM application to construct 100 cluster trees for each case. From these cluster trees a consensus tree is constructed, according to the majority rule as explained in the previous chapter, with the **consense** tool of the PHYLIP package [6]. This consensus tree is used to select the most representative cluster tree from the set of constructed trees with the **clustertree** tool of the TreeSOM package. The selected tree will be displayed in several figures at various clustering levels. The configuration parameters, including the number of selected keywords and map size will be described in detail in each case.

3.4.1 Experiment 1

In this first case study our goal is to cluster the publications based on their subcategory, where each subcategory is mapped on a single node of the SOM. In the used data set there are 13 subcategories where each subcategory is represented with 10 publications, except a subcategory with only 4 publications. Therefore we decided to use a 4×3 map with 12 nodes.

For this experiment 22 keywords were selected which complied to the following parameters. The keywords were at least 4 characters long, appeared at least in 11 and at most in 60 abstracts, and the total number of occurrences were below 180.

Training of the SOM existed of two phases: the first training phase consisted of 1,000 iterations and the second of 10,000 iterations. The differences between the phases are the used values of the learning rate and radius distance of the neighborhood function. In the first phase the learning rate linearly decreases from 0.2 to 0, where in the second phase this will be from 0.02 to 0. The radius distance linearly decreases from 3 to 1 in the first phase and from 2 to 1 in the second phase. Note that the 1 indicates that only the winning node is adjusted to the input data accordingly.

The average distance from the 100 cluster trees to the constructed consensus tree is 0.60593 as calculated with the distance measure introduced by Algorithm 3 at page 16. The most representative cluster tree has a distance to the consensus of 0.29630. This cluster tree is depicted for several different cluster thresholds in Appendix A.

The first figure, Figure A.1, shows the initial 12 cluster roots using the cluster layout. The second figure, Figure A.2, shows the initial clustering using the family layout, where the *Information Systems* subcategory is emphasized. The following figures show the cluster formation, using the same family layout, with cluster thresholds of 0.21, 0.28 and 0.34 respectively. At a cluster threshold of 0.7 the last two clusters are merged together.

3.4.2 Experiment 2

In comparison with the first case study we enlarge the map size and the number of keywords in the second case study. Our goal in this second case study is to map initially each publication on a single node of the SOM. Therefore we use a 13×10 map with 130 nodes, which is slightly more than the 124 publications in the input data.

The number of keywords is enlarged to 126, which complies with the fol-

lowing parameters. The keywords were at least 4 characters long, appeared at least in 5 and at most in 20 abstracts, and the total number of occurrences were below 60. The same training parameters were used as in the previous case study, expect a different radius values. As a larger map size were used for this case, the radius distance decreases in the first training phase from 6 to 1 and in the second phase from 3 to 1.

The average distance from the 100 cluster trees to the consensus tree is 0.85505 and the most representative cluster tree has a distance to the consensus of 0.74748. In Appendix B the cluster tree is depicted for several different clusters and pruning thresholds. All figures shows the tree using the family layout, where the *Operations* subcategory is emphasized, except Figure B.3 where the cluster layout is used.

In Figure B.1 the initial clustering is shown with 59 clusters. In Figure B.2 and Figure B.3 the tree with a clustering threshold of 0.27 and a pruning threshold of 0.04 is shown. At this clustering level there exist 32 clusters. If Figure B.1 and Figure B.2 are compared, the influence of the pruning operation becomes visible: the number of internal nodes in the middle of the tree is less, so the tree itself is more stretched and a better overview of the different clusters is gained. In Figure B.4 the clustering and pruning thresholds are respectively 0.33 and 0.04, where 22 clusters are depicted. Figure B.5 shows 10 clusters at a clustering threshold of 0.4 and a pruning threshold of 0.12. At a cluster threshold of 0.9 the last two clusters are merged together.

3.5 Summary

In this chapter the process of acquiring results with the VisualTreeSOM application is described; it concludes with showing two experiments as individual case studies. In these experiments we use a real-world data set of 124 publications located on a web portal. The publications are divided into 13 subcategories and our goal is to cluster the publications, based on their abstract, with a set of selected keywords contained in the abstracts.

VisualTreeSOM incorporates a tool for selecting keywords from the publications using a set of parameters. In an interactive manner the user can filter keywords from the input abstracts by changing the desired keyword length, the number of abstracts the keyword must appear in and the total number of keyword occurrences.

After a set of keywords is selected, the self-organizing maps are trained on

a computer grid using a stripped version of the TreeSOM toolset. Although the grid implementation is in this case not highly contributive, due the fact of the small map size and small number of data vectors, it could be a valuable addition in future research.

The results of the two experiments are not very satisfying. In the first experiment the publications were not truly mapped on nodes, each representing a subcategory. In the second experiment the initial clustering existed of 59 clusters, where 124 clusters were strived for. Some factors for these failures could be possibly the small amount of keywords used during training and the small size of each abstract. Also the publications were taken from the same field, where large amounts of keywords are overlapping.

Part II

Longevity

Chapter 4

An Introduction to Genetic Analysis of Longevity

In this chapter a study is introduced, which involves the genetic analysis of longevity. As part of this study a complex medical analysis has to be performed, which is outlined in the problem description. The following chapter describes a method to tackle the stated problem. Therefore, this chapter can be considered as a generic introduction for the following chapter, presenting some biological terms and familiarizing the reader with the subject.

4.1 Introduction

At the *Leiden University Medical Centre (LUMC)*¹ a research group of the department of Medical Statistics and Bioinformatics is performing research in the fields of ageing and longevity. The research aims at the identification of mechanisms, which play a central role in the ageing process. In this research the functional variations in genes of the general population are investigated. The search for interesting genes is two folded: on the one hand the search of genes that contribute to mortality and on the other hand the genes that might be responsible for becoming extremely long-lived. In the first case the emphasis lies in understanding the differences between humans in their risk to develop diseases, such as osteoarthritis and cardiovascular diseases. In the second case, which is the reverse of the first case, the emphasis lies in the possibility of subjects to survive to very old ages.

In our study the second case is applicable: we are interested to identify genes involved in longevity. To explore and locate these yet unknown genes,

¹ <http://www.lumc.nl>

we will use data collected by LUMC from the so-called *Leiden Longevity Study* [10]. This study includes sibships² consisting of *at least* two long-living siblings (men aged 89 years or above; women aged 91 years or above). In comparison with studying only long-living singletons, we can expect that in our case an enrichment of genetic factors contributes to longevity³. From these participating families data were collected, including a venous blood sample for isolation of DNA. This data was not only collected from the long-living subjects, but also from their offspring, and the partners of their offspring.

In our analysis we will only concentrate on the data of the long-lived sibships and the partners of their offspring. In this analysis the long-lived sibships will be the cases. Their offspring are assumed to have a higher susceptibility to become long-lived in respect to the general population, as they have a life-long mortality advantage of approximately 30%. Therefore the partners of the long-lived sibships' offspring will be the control group, representing the general population. The advantage of using partners as the control group is the fact that they roughly share the same socio-economic environment, and have the same geographical background.

4.2 Problem description

As mentioned in the introduction, the goal of this analysis is to identify genes involved in longevity. In order to identify these genes, a genome wide scan is made for each subject from the collected blood samples. This genome scan includes the measurement of 500,000 genetic variants. With such a complete scan of the genetic information of a subject, possible patterns for the genetic make-up for long-living individuals can be recognized.

The human genome is composed of 22 different chromosomes plus the sex-determining X and Y chromosomes, with in total almost 3.5 billion DNA base pairs⁴. A segment of DNA that contains information on hereditary characteristics is called a *gene* and the human genome harbours an estimated number of 25,000 genes. Note that not all strands of DNA consist of this hereditary information, i.e., other areas of DNA have other functions. These genes are

²A sibship includes all children born to a set of the same two parents.

³All relating studies showed that genetic factors plays an important role in human longevity. The studies claim that gene variation determines the lifespan of humans up to approximately 25%.

⁴There exist four distinct bases, also referred to as nucleotides: adenine (A), which forms a base pair with thymine (T); as does guanine (G) with cytosine (C).

Chromosome	Genes	Base pairs
1	2,968	245,203,898
2	2,288	243,315,028
3	2,032	199,411,731
4	1,297	191,610,523
5	1,643	180,967,295
6	1,963	170,740,541
7	1,443	158,431,299
8	1,127	145,908,738
9	1,299	134,505,819
10	1,440	135,480,874
11	2,093	134,978,784
12	1,652	133,464,434
13	748	114,151,656
14	1,098	105,311,216
15	1,122	100,114,055
16	1,098	89,995,999
17	1,576	81,691,216
18	766	77,753,510
19	1,454	63,790,860
20	927	63,644,868
21	303	46,976,537
22	288	49,476,972
X	1,184	152,634,166
Y	231	50,961,097

Table 4.1: Statistics of the human genome, as stated in [11]

unevenly distributed across the chromosomes. Table 4.1 gives an overview of the chromosomes and the estimated number of genes and bases they contain.

As commonly known, each individual has a unique DNA sequence. However, overallly our DNA is commonly shared for approximately 99%⁵. The unique sequence is, for approximately 90%, the result of a large amount of single point variations in the total DNA sequence. Such a single point variation in DNA, where a single nucleotide replaces one of the other three nucleotides, is called a *Single Nucleotide Polymorphism (SNP)*. For example, below are two DNA sequences located at the same segment:

⁵A recent study showed that one person's DNA can be as much as 10% different from another's, decreasing the total shared genetic information to 90%.

...AGGTCTTT...
...ACGTCTTT...

In this example a SNP is located at the second base, as there is a difference at this particular nucleotide between the two sequences. In this case there exist two possibilities, in biological terms referred to as *alleles*, for this particular SNP: G and C.

An individual possesses a copy of each chromosome from both parents. Assume that the two sequences in the above example belong to the same individual and each sequence represents a copy of one of the parent's chromosomes. If this is the case, the individual possesses for the second base the following pair of alleles: GC. As the selection of which chromosome is inherited for each parent occurs by chance, the individual could have possess the following pairs of alleles: GG, GC or CC. In biological terms these unordered pairs of alleles are called *genotypes*. If both alleles are the same, e.g., AA for the first base in the above example, it is called *homozygote*; if the alleles are different, it is called *heterozygote*. Note that for heterozygote alleles the ordering is irrelevant, so GC is equivalent to CG.

Not all single variations are considered to be a SNP, mostly only these variations that occur for at least 1% in the population. Otherwise there will be an unmanageable amount of SNPs with relatively small significance. Using this percentage, there is on average a SNP at every 100 to 300 bases along the human genome, including the SNPs with a frequency below 1%. The significant variations in the DNA sequence are not considered to be responsible for a disease state. Instead, multiple SNPs will help to map and identify a disease on the human genome, as the particular SNPs are located near the genes associated with the certain disease. Also the fact that SNPs are inherited, where they do not change much from generation to generation, makes SNPs very suitable for mapping disease and determining diseases susceptible.

In the beginning of this section the goal of the analysis is stated as “to identify genes involved in longevity”. In fact this is the overall goal of the *Leiden Longevity Study* and our analysis will be a small part of the entire study at LUMC. The collected data will be analyzed with several different methods, where our method is one of the more complex ones. From all these different analyses, the results will be combined to find a list of all SNPs possibly associated with longevity. The selected SNPs will be measured in the second half of the study, but this time with the DNA sample of the offspring of the long-living subjects. With this last analysis the false positive SNPs,

which are inherited from the partners of the long-living subjects, can be filtered out. In such a way a list of truly important SNPs will be left.

In our analysis the real challenge lies in the quantity and process of the SNPs. The data consists of 895 subjects, where 424 individuals are long-lived and 471 individuals form the control group, representing the general population. For each subject the genome wide scan measured 500,000 SNPs, divided along the chromosomes. In total this analysis comprises a set of roughly 450 million data points. In order to process this vast quantity of data points for localising and identifying the interesting SNPs, efficient and scalable methods are required.

Chapter 5

Association Analysis using Haplotypes

In this chapter an innovative method is presented, which tries to achieve the stated goal of the previous chapter. The method introduced in this chapter is based on haplotypes and is in particular aimed at finding patterns of haplotypes for localizing susceptible genes. In the first section the solution basis explains in detail how the method tackles the stated problem and describes the various packages used in order to achieve this result. The following section presents the results obtained by this method. Section 5.3 introduces a tool for visualizing the generated analysis files. With this tool the interesting SNPs can be easily found in a user-friendly manner. Finally, this chapter concludes with a short summary, including discussion and some suggestions for future research.

5.1 Solution basis

In this section the process that forms the solution of the stated problem is described thoroughly in several subsections. The basic outline of this solution is based on the use of haplotypes. This biological term will be introduced and explained in the first subsection. In Subsection 5.1.2 the collected data, as released from LUMC, is described and in particular the transformation of this data into the correct input is presented. The last two subsections describe existing software packages, which are used in succession to analyse the data. For both packages the basic outline and background are described briefly and the output is analyzed.

5.1.1 Haplotypes

In the previous chapter it is explained that multiple SNPs can map and identify a disease on the human genome, as these SNPs are located near the genes associated with the certain disease. Such a set of nearby SNPs, located on the same chromosome and inherited from the same parent together, is called a *haplotype*. For example, below are two strings of alleles along a single chromosome:

```
...ACATACTACAATAAGTACAAATGAT...  
...AAATACTACCATAACTACAAGGAT...
```

In this example there is a SNP exactly at the 2^{nd} , 10^{th} , 15^{th} , and 21^{th} base. Such a base, where a SNP is located, is called a *marker*. Assume that in this example the first string of alleles is inherited from the father and the second string is inherited from the mother. If this is the case, the haplotype of the father would be: $H_{father} = (C, A, G, T)$; and the haplotype of the mother would be: $H_{mother} = (A, C, C, G)$. If this is not the case and the strings of alleles are not ordered according to the parental origin, we would have the list of genotypes $G = (\{A, C\}, \{A, C\}, \{C, G\}, \{G, T\})$ for the above markers. Possible haplotype configurations¹ for this list of genotypes G could be:

$$\begin{pmatrix} AACG \\ CCGT \end{pmatrix}, \begin{pmatrix} ACGG \\ CACT \end{pmatrix}, \text{ or } \begin{pmatrix} CCGT \\ AACG \end{pmatrix}$$

In comparison with SNPs, haplotypes are much more informative as SNPs alone are relatively uninformative. Unfortunately, haplotypes can not be directly obtained from DNA samples as current laboratory techniques only produce genotypes. Therefore the construction of haplotypes from measured genotypes is a crucial step in the analysis process and will be explained in the following two subsections.

5.1.2 Data set

The data collected by LUMC includes measurements at the same marker positions inside the DNA sequence for each subject. At each single marker the genotype for the individual is measured. As explained in the previous chapter, a genotype consists of an unordered pair of alleles, where from both parents a single allele is inherited. In our analysis the type of alleles is unimportant as it makes the analysis more complicated. Therefore it is replaced by an

¹For a genotype G with k heterozygous markers, there are 2^{k-1} different haplotype configurations.

encoding. The interest in a single measurement lies in the construction of the found genotype: is the genotype constructed from both frequent, both infrequent or a combination of frequent and infrequent alleles.

The used encoding replaces the alleles with the following digits: 0, 1, or 2. Zero (0) is used if the measurement of the allele failed or is invalid. As both alleles are necessary to obtain the genotype, it is not possible to retrieve the genotype at this marker position. Therefore the other allele of this pair has to be encoded with a zero. One (1) is used for the most frequent allele for this genotype version and two (2) for the most infrequent allele. The frequencies are calculated from all measurements across all subjects for each marker position.

This encoding is elaborated in the following example. Assume we have measured the genotype GC, thus at a particular marker position there exists a variation of the bases G and C. As explained in the previous chapter, the possible genotypes for this marker position will be GG, GC, or CC. If for this particular marker position the majority of the subjects have the GG or GC genotype, the allele G gets the encoding 1 as this allele is the most frequent. Allele C will get the encoding 2 as it is the most infrequent allele for this marker position. In this case GG will be encoded as 11, GC as 12, and CC as 22.

The collected data of the overall study is stored inside a database at LUMC. From this database an export is made for the data needed in our analysis. This export function delivered a flat data file for each single chromosome. Note that in our applied method the chromosomes are analyzed independently. This is mainly chosen because of the independent nature between chromosomes, but it also introduces some parallelization. The exported file contains the data in a tabular way, with the following columns:

- CaseControl — Integer indicating if the record is a case (1) or a control (0). The individuals marked as a case are long-lived. The controls are the individuals who represent the general population
- Family — Integer for identifying a particular individual. Each of the 895 individuals has a unique number in the range [1 – 10,097]
- Code — *Perlegen Sciences*² internal SNP identifier. This company provided the genome wide scan from the blood samples
- Chromosome — Chromosome number on which the SNP marker is measured. X and Y are used for the sex-determining chromosomes
- Contig Position — Nucleotide position of the SNP marker inside the DNA sequence

²<http://www.perlegen.com>

- a1 — The encoding of the nucleotide base of the first allele measured at the marker position
- a2 — The encoding of the nucleotide base of the second allele measured at the marker position

It is clear that this exported data contains duplicated and redundant information. Also the format of this data has to be changed in order to be a valid input for the upcoming software packages. With the use of several Perl³ scripts the data was transferred into the desired format. An example of this format is as follows:

```
Id Status M1 M2 M3 M4
1 a 1 1 0 2
1 a 2 1 0 2
2 c 1 1 2 2
2 c 2 2 2 1
```

The first line of the format is the header and contains the following items:

- Id — Integer for identifying a particular individual. This identifier is successive, thus counting from 1, for the first subject, till 895 for the last subject
- Status — Character indicating if the subject is a case, also referred to as *affected* (a) or a control (c). The subjects have to be ordered in such a way, that first all affected subjects are listed and thereafter all control subjects
- Markers — The markers are ordered according to their position on the chromosome

The rest of the file contains the encoded genotype data. Each genotype is divided into two lines in the input file, with the first field denoting the subject identifier, the second field the subject status, and the rest of the fields denoting alleles at each marker. Thus, the unordered allele pair at each marker is divided into two lines for a single subject.

In the above example the population consists of 2 subjects, where both case and control are represented by one individual. For both individuals four genotypes are measured, where the measurement of the third genotype for the first individual was invalid.

During the transferring process some data is masked in order to comply with the format. This is the case for the subject identifications and marker

³Practical Extraction and Report Language

positions. This problem is tackled by constructing two mapping files, generated during the transferring process. These mapping files preserve the linkage between the subject identifications and marker positions used in our data files and the data stored at LUMC. The mapping file for preserving the marker positions stores at each line the contig position and Perlegen's internal SNP identifier. In this case, each line of the mapping file corresponds with the marker position (this is the integer after the character 'M'), used in our data file. The second mapping file preserves the linkage between the subject identifications. In this file, where at each line the family identifier used by LUMC is placed, corresponds to the line number with the subject identifier used in our data.

5.1.3 Haplotype Reconstruction

After the genetic data is transformed into the right format, it has to be haplotyped. For the construction of haplotypes we use an existing software package, called *HaploRec* [12]. This approach reconstructs haplotypes using a Markov chain⁴ approach, especially aimed at long marker maps as in our case.

As described in the previous subsection the genetic data contains the measured alleles for a large number of SNPs. However, the alleles for each marker were arbitrarily divided between the two lines of the subject. The goal is to order these two alleles at each marker position according to the parental origin. This ordering is called *haplotyping* and is executed with the *HaploRec* software package. After the data is haplotyped, each line will denote a single haplotype and each subject contains two haplotypes representing both parents.

The first step in the haplotyping process is to split the data vertically, creating so-called *windows*. Each window contains a predefined number of markers for all subjects located in the data. After each window is haplotyped, they are glued back together into a single file, containing the reconstructed haplotyped data. This reconstructed output file has the exact same format as the input file. Each window contains some overlapping and extra markers in order to improve accuracy at the borders of the windows. A schematic outline of three windows is depicted below:

⁴A Markov chain is a discrete-time stochastic process, with a finite number of states, in which the probability of occurrence of a future state is conditional only upon the current state.

mmmmmmmmmmooooee
mmmmmmmmmmooooee
mmmmmmmmmmooooee

In this example each window has a maximum size of 16 markers, divided between business, overlapping, and extra markers. The business markers, denoted with 'm', are the markers that have to be haplotyped in the current window. The overlapping markers, denoted with 'o', are used as the already haplotyped prefix in the next window. The extra markers, denoted with 'e', are haplotyped after the overlap and are discarded after the current haplotyping window. The overhead between windows in this example would be $\frac{3+3}{16} = 37.5\%$, which is the pay-off to improve accuracy.

Each window is haplotyped in a statistical approach, using a Markov chain to model the haplotype distribution. This model estimates the most likely haplotype pair for each genotype, by capturing statistical dependencies between neighbouring alleles. The outline of this model is that neighbouring alleles can tell a lot about the next allele, due the linkage disequilibrium⁵ between alleles of nearby markers. Although the linkage is the strongest between neighbours, represented by a first order Markov chain, a neighbourhood of several markers is more informative and can show a stronger linkage disequilibrium. Therefore the model uses a variable order Markov chain, adjusting the size of the neighbourhood for each marker and haplotyping it individually.

5.1.4 Haplotype Pattern Mining

The next step in the process is to locate interesting markers in the haplotyped data. Therefore we use another existing software package, called *Haplotype Pattern Mining (HPM)* [13]. The approach implemented by HPM is based on algorithms for finding frequent patterns from large databases and this model is extended with traditional association analysis algorithms. With this combination HPM can discover recurrent marker patterns in a computationally efficient manner. The method itself is model-free, i.e., there are no assumptions about the inheritance model of the disease. Also the statistical model is nonparametric and can handle missing and erroneous data.

⁵ Linkage disequilibrium is the association between alleles of two different genes on the same chromosome, where a greater co-occurrence of the two genetic markers exists than would be expected for independent markers.

In diseases with a reasonable genetic contribution, as in our case, affected individuals are likely to have higher frequencies of associated marker alleles near the disease susceptibility gene than control individuals. Note that in our case the “disease” can be described as the suspiciousness to become long-lived. These combinations of marker alleles, haplotype patterns, which are more frequent in disease-associated chromosomes than in control chromosomes, are searched for in the data. Also HPM introduces the option to find negative associations between marker alleles, where the control-associated marker alleles are more frequent in respect to the disease-associated ones. In both frequency sets we are interested.

The search for frequent haplotype patterns is aimed at sets consisting of nearby markers with predefined lengths. These frequent haplotype patterns found by HPM are sorted by the strength of their association to the disease and the resulting list of patterns is saved in a so-called *pattern file*. Note that only the patterns with an association strength, also referred to as the χ^2 value, above a predefined threshold are taken into account. HPM saves the following information for each pattern:

- CHI — The χ^2 value of the pattern
- MA (Matching Affected) — Number of affected subjects that contain the particular pattern
- NA (Non-matching Affected) — Number of affected subjects that do not contain the particular pattern
- MC (Matching Control) — Number of control subjects that contain the particular pattern
- NC (Non-matching Control) — Number of control subjects that do not contain the particular pattern
- F(A) — The frequency of the pattern among all affected subjects
- F(C) — The frequency of the pattern among all control subjects

HPM also produces a so-called *marker file*, where for each marker in the haplotyped data its score and p -value⁶ is stored. The marker score represents the number of frequent patterns in which the particular marker is part of. The local p -value indicates the statistical significance of the marker and is estimated with permutation testing. This marker file is exceptionally suited for localizing the interesting markers in the data.

⁶The probability that the results of an experiment have occurred by chance.

5.2 Results

The previous section shows the general process of how each chromosome can be analyzed, describing each step in detail. In this section the results acquired by each step in the process are described. This is elaborated by analyzing one of the largest chromosome in the genome and showing the obtained results. For each other chromosome the process is exactly the same, but the expected analyzing time will be lower, depending on the chromosome's size.

The analysis process of the first chromosome is executed on a dedicated machine, including two 64-bits Pentium 4 CPU's at 3.20 GHz, 2 GB RAM, and operating on a 64-bits implementation of SUSE⁷ Linux 9.2.

5.2.1 Preprocessing

The exported data of the considered chromosome, as released from LUMC, had a size of almost 7 GB. With the use of three Perl scripts this data was transformed into the desired format. The first script filtered all SNP markers from the data and generated the mapping file in order to preserve linkage. The second script inverted the data, as the desired format stores the marker positions in a column oriented fashion in contrast to the row based fashion used in databases. The third and final script reordered the individuals based on their status and changed the identifiers in a strict successive order. This last script also generated a mapping file for preserving linkage between the subject identifications.

The transferring process took in total roughly 17 minutes and constructed the desired data file, used in the next step. This data file, representing the considered chromosome, had a size of 95 MB. It contained 895 subjects, where 424 individuals are marked affected and 471 individuals marked as control. For each subject there were 27,738 SNP markers measured, where in total 5.23 % of the markers were invalid. This corresponds with 1,451 unknown genotypes per subject on average.

5.2.2 Haplotyping

For the haplotyping process another Perl script is used. It firsts cuts the header line from the data file and then it removes for each subject its identifier and status field. The remaining genotyped data is split into windows and saved as temporarily files. Each window contains a subset of markers for all

⁷ <http://www.novell.com/linux/>

subjects, thus the data is split vertically. After the windows are haplotyped independently with HaploRec, they are glued back together. The identifiers, status fields, and the header line are added back, so that the format remains the same.

The window size is set to 100 markers, including 20 overlapping and 20 extra markers. The overhead between windows is in this case $\frac{20+20}{100} = 40\%$ and results in 462 windows for the considered chromosome. These 462 windows were haplotyped in 109 hours, which corresponds with 15 minutes for each window on average.

5.2.3 Frequent pattern mining

After the data is haplotyped, it is analyzed with HPM for frequent patterns. The maximum pattern length is set to 10, including a possible wildcard. Such a wildcard can represent any character in the encoding set. HPM will calculate the χ^2 value for each possible combination of successive markers with a maximum length of 10. This χ^2 value indicates the association strength to the disease for the particular pattern. The χ^2 -threshold is set to 10, so only patterns with a value greater than this threshold will be listed. Both positive and negative associated patterns will be examined. To increase the significance of the local marker p -values, the number of permutations is set to 10,000.

With the above described parameters the considered chromosome was searched for frequent patterns. This analysis took 73 hours in total and there were 32,928 strongly associated patterns found. The pattern file stored from these patterns the top 10,000 strongest. The marker file contained the local score and p -value for each of the 27,738 markers. The 10,000 strongest patterns were constructed from 3,573 unique markers. So, the marker file contained 3,573 markers with a marker score above 0 and a p -value below 1. Unfortunately, from these 3,573 markers there included some with a p -value of 0, indicating that the number of permutations were too low to get a valid statistical significance.

In order to get statistical significant p -values, the number of permutations should be increased. However, this implies that the analyzing time increases linearly to the number of permutations, as for each marker extra calculation has to be done. This extra calculation will be useful for only a small set of the markers, as the major part of the markers are not included in any frequent pattern. Therefore an efficient strategy was constructed, that reduced the analyzing time and preserved the accuracy of the results.

This strategy first analyzes the full chromosome, i.e., all markers, using a low number of permutations. Afterwards, the produced marker file is inspected and searched for markers with a p -value of 0. If such a marker is found, extra analysis is needed with a higher number of permutations. This extra analysis is only done on the neighborhood of the found marker. Therefore, a subset of markers is cut from the original haplotyped data and analyzed further. The number of permutations is iteratively increased with a factor of 10, until all markers in the subset have a statistical significant p -value greater than 0. The results of the extra analysis are merged into the original marker file, replacing the incomplete values. Note that the pattern file does not need any update, as it is complete from the original analysis on the full chromosome.

The above strategy was implemented with a Perl script and used to search for frequent patterns in the considered chromosome. The same parameters were used as in the first experiment, with the exception that the initial number of permutations was set to 1,000. In total this analysis took 14 hours and 27 subsets had to be analyzed further. From these 27 subsets, 24 had valid p -values after 10,000 permutations; 3 subsets needed 100,000 permutations to get statistical significant p -values.

Both output files produced by HPM are simple text files, containing a great amount of information. In order to make the analysis of the output more easily, a visualization tool is constructed. This tool can show both files independently in a user-friendly way and inside a graphical environment. In the following section this tool will be described in detail.

5.3 VisualSNP

For the graphical representation of the produced output files a Java application, called *VisualSNP*, is constructed. Like *VisualTreeSOM*, this application can be easily launched using Java Web Start and has a native look and feel, as the GUI is created with SWT. With this application both pattern and marker files can be visualized. Each type of visualization will be discussed in the following two subsections.

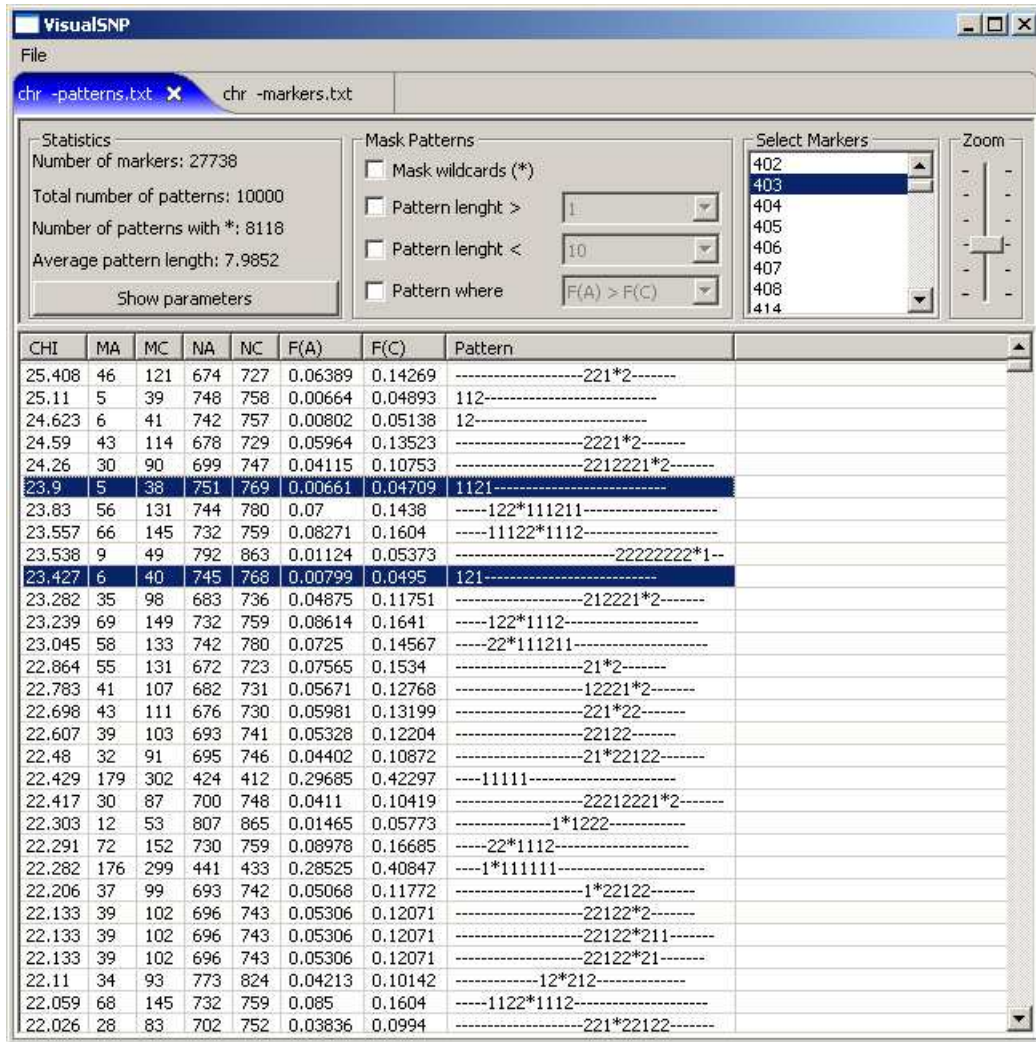


Figure 5.1: VisualSNP visualizing a pattern file

5.3.1 Visualizing pattern files

In Figure 5.1 the main window frame of VisualSNP, executed on the Windows platform, is shown. Each opened pattern file gets its own tab inside this window, where in the upper part the settings panel is located and in the lower part the frequent pattern table is displayed.

This settings panel consists of several groups, mainly used for manipulating the data listed in the underlying table. This table lists all frequent patterns, complying with the above specified parameters.

The “Statistics” group contains some general information about the pat-

tern file. At first it shows the total number of SNP markers inside the analyzed data. Secondly it shows the number of frequent patterns inside the pattern file and the number of these patterns with a wildcard. Finally it shows the average pattern length. If the parameters button is pushed, a dialog will pop-up containing the parameters used by HPM to calculate the frequent patterns.

The group “Mask Patterns” shows a number of options to mask certain patterns listed in the table. It can mask patterns with a wildcard, patterns with a defined minimum or maximum length, and patterns based on their frequency inside the status group.

The “Select Markers” group shows all markers, which are represented by at least one single frequent pattern. If a marker is selected from the list, all patterns which incorporate this marker will be selected in the pattern table. Multiple markers can be selected from the list.

The “Zoom” group consist of a single scalar, representing the zoom factor on the chromosome. By default this scalar is set to the maximum, so that only the pattern itself is displayed in the last column of the table. If the scalar is set to the minimum, all markers on the chromosome are displayed for the pattern. With the use of this zoom factor it is possible to easily check the position of each pattern inside the chromosome.

Each record in the table corresponds with a frequent pattern. The data listed in the table can be ordered according to each column, by pressing the corresponding headers. The last column will order the pattern according to the position inside the chromosome. A tooltip shows for each frequent pattern the exact position of the first marker of this particular pattern.

5.3.2 Visualizing marker files

In Figure 5.2 the visualization of a marker file with VisualSNP, running on the Windows platform, is shown. Each opened marker file gets its own tab inside the main window frame, where in the upper part the marker graph is displayed and where in the lower part the settings panel is located. This settings panel consists of several groups, mainly used for manipulating the graph and showing additional information.

The “Statistics” groups contains two buttons, each opening a new dialog with additional information. If the parameters button is pushed, a dialog will pop-up containing the parameters used by HPM to calculate the marker scores and p -values. If the peaks button is pushed, a dialog will pop-up containing a table, which includes all markers that are part of at least a single frequent pattern. For each marker inside this table, the same information is

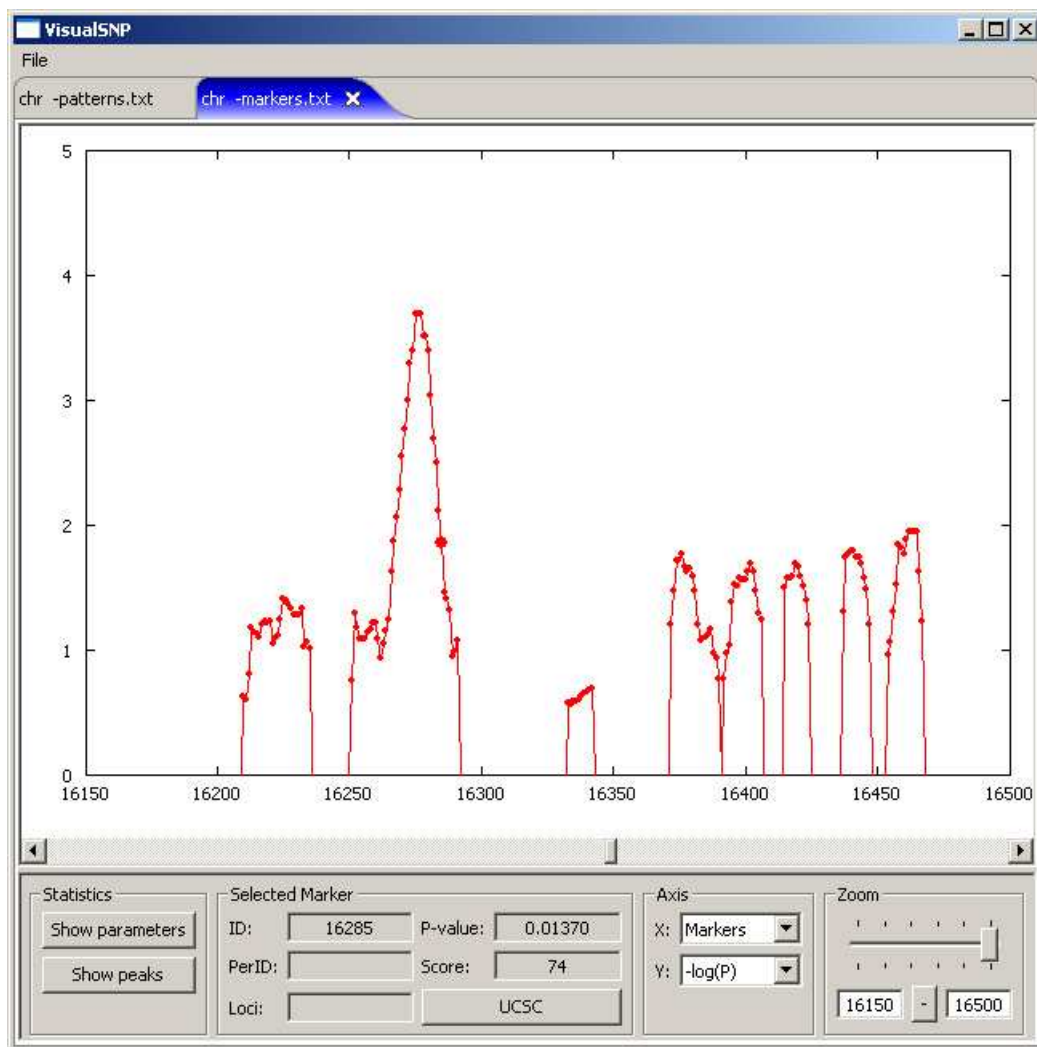


Figure 5.2: VisualSNP visualizing a marker file

listed as shown in the “Selected Marker” group, only in a tabular fashion.

The “Selected Marker” group shows all available information about the selected marker. A marker inside the graph can be selected by clicking on the dot, representing the marker. If a marker is successfully selected, the dot will be slightly bigger painted inside the graph. The corresponding marker identifier, marker score, and p -value will be listed. If the associated marker mapping file is specified during opening of the displayed marker file, additional information will be shown. This marker mapping file contains for each marker the contig position on the chromosome and *Perlegen Sciences* internal SNP identifier. Also the UCSC button will then be enabled. If this button is

pushed, it will automatically open a new web browser and shows the selected marker in the *UCSC Genome Browser*⁸.

The group “Axis” shows all available variables for representing the axis of the displayed graph. For the *X*-axis there is only a single option, namely the marker identifiers. For the *Y*-axis there are three options: the marker score, the *p*-value, and the negative logarithm of the *p*-value. This last option is added to stress the markers with a *p*-value near 0, as these markers are more interesting than the others.

The “Zoom” group contains a scalar, representing the zoom factor on the *X*-axis. By default this scalar is set to the minimum, so that all markers on the chromosome are visible. The user is enabled to zoom in at a particular part of the chromosome, by specifying the desired range in the corresponding text fields. In Figure 5.2 there is zoomed in at the range of markers with identifiers between 16,150 and 16,500. This range can also be specified with the mouse, by selecting the desired range inside the graph.

5.4 Summary

This chapter introduced a method, based on haplotypes, for localizing interesting markers in genetic data. The described method consists of multiple steps and uses two existing software packages. The genetic data is first transformed into the desired format, where it can be haplotyped. After the data is haplotyped, where the measured alleles for each marker are ordered according to the parental origin, it is analyzed for frequent patterns. This analysis includes the calculation of the local *p*-value for each marker in the haplotyped data. This value indicates the statistical significance of the marker and is dependent on the number of occurrences that the particular marker is part of a found frequent pattern. A visualization tool presents both the patterns as markers in a graphical way. With the help of this tool, interesting markers (which are located near susceptible genes) can be localized in an easy and user-friendly way. The method is elaborated by showing the entire process on the basis of one of the largest chromosome in the genome.

In this method each chromosome is analyzed independently. This is mainly chosen because of the independent nature between chromosomes, but it also introduces parallelization. In this case each chromosome can be analyzed in detail for explicit SNP markers located on the particular chromosome. However, unknown relations between markers across chromosomes can not be detected. Another drawback of the method, which is the direct consequence of using haplotypes, is the implicit assumption that affected individuals are

⁸ <http://genome.ucsc.edu>

distantly related and share genomic regions from a common ancestor.

As future work, the software package *Tree Disequilibrium Test (TreeDT)* can be added to the existing method. This package extracts in a tree-structured model, information about historical haplotype recombinations in the population. This information is used to locate and map fragments, potentially inherited from a common diseased founder, to the disease gene.

Overall, it could be useful to integrate the method into a single package, so that the entire process can be controlled and changed using a simple tool. This will give the biologists full control over the entire method, including parameters and direct feedback. In this way the analysis will become more transparent and the analyzing time could be decreased.

Part III
Conclusion

Conclusion

In this thesis we showed how two existing data mining algorithms have been extended successfully for handling larger or more generic data sets. Both initial algorithmic approaches are described in detail, including the fundamental concepts and principles used by these methods. After for each method the problem description is outlined, a structured solution basis explains how the introduced problem can be tackled. Combining all described concepts in the solution basis, a working implementation is constructed. At last it is showed how the working implementations acquire their results. Both results are presented with a visualization tool that can be launched easily from a web browser. This tool has a native user interface and shows the results in a dynamic manner, where the user is able to interact with.

In the first part of this thesis we showed how an existing data mining approach, which consisted out a set of tools for clustering analysis in Self-Organizing Maps, was made more flexible and robust, such that any generic data set can be used. The tool responsible for training the SOMs was scaled to a computer grid. An interactive application was constructed that could display the results in a dynamic manner.

Two experiments were carried of with a real-world data set, containing publications of the same field, located on a web portal. The publications were clustered, based on their abstract, with a set of selected keywords contained in the abstracts. The results of the experiments were not very satisfying, but factors for these failures could be the small amount of keywords used during training and the small size of each abstract.

As future research, experiments should be carried out with a data set including publications from different fields and with abstracts of reasonable size. This will show the true potential of the used clustering algorithm. The visualization application can be extended by some valuable additions, such as importing cluster confidence and introducing colors for representing the contents of each cluster.

In the second part of this thesis we introduced a method that can mine a very large genetic data set, using existing software packages. This introduced method tries to identify genes involved in longevity, using association analysis with haplotypes. The genetic data, consisting of 500,000 genetic variations in 900 subjects, was split based on the chromosome number and haplotyped. The haplotyped data was searched for frequent patterns, where the found results were displayed in a visualization tool. This tool can be used easily by biologists in identifying susceptible genes.

The analyzed chromosome, containing 27,738 SNP markers, was haplotyped in 109 hours. Frequent pattern mining took 14 hours, where 27 interesting marker subsets were found containing markers with a p -value below 0.001. From these 27 marker subsets, 3 subsets contained markers with a p -value below 0.0001.

As future research, the rest of the chromosomes can be analyzed with the introduced method for identifying susceptible genes. As the considered chromosome is one of the largest in the genome, it is expected that for the other chromosomes the analyzing time will be lower.

Acknowledgements

First and most important to thank are my loving parents, without their financial and unconditional support this master thesis could not be manufactured. Secondly, gratitude goes out to my supervisors J. Kok and W. Kusters. They introduced me to both subjects, provided valuable comments and suggestions, and supported me during the entire project. Without their guidance, understanding, and patience, this thesis would not have its current quality.

I would like to thank the following persons individually: E. Samsonova for her help in explaining and clarifying the TreeSOM toolset; N. Mellegård for the initial idea and source code of the visualization applet for TreeSOM; C. Soares for his valuable feedback of VisualTreeSOM and the providing of the used dataset; B. Heijmans and M. Beekman for their help in introducing and explaining all biological terms; and H. Toivonen for his valuable advices and suggestions about the use of the HaploRec and HPM software packages.

Apart from the people who helped me directly I would also like to thank my fellow colleague students that started studying Computer Science alongside me in 2001. We had some great time during lectures and even more during the breaks. These fond memories I will never forget and made studying so enjoyable. Last, but certainly not least, I want to thank my friends, family and relatives for their encouraging, support and inspiration during all those years.

Bibliography

- [1] E.V. Samsonova, T. Bäck, J.N. Kok, and A.P. IJzerman, *Reliable Hierarchical Clustering with the Self-Organizing map*. In A. F. Famili, J. N. Kok, and J. M. Peña, editors, *Advances in Intelligent Data Analysis VI: 6th International Symposium on Intelligent Data Analysis*, volume 3646 of *Lecture Notes in Computer Science*, pages 385-396. Springer Verlag, 2005.
- [2] T. Kohonen, *Self-Organizing Maps*, vol. 30 of *Springer Series in Information Sciences*. Springer, second ed., 1997.
- [3] E.V. Samsonova, *TreeSOM User Manual*, University of Leiden, September 2005.
- [4] N. Mellegård, *Utilizing the TreeSOM Algorithm to Cluster and Visualize Free Text data*, LIACS Competence Project — Published Results, University of Leiden, January 2006.
- [5] J. Felsenstein, *The Newick Tree Format*, University of Washington, <http://evolution.genetics.washington.edu/phylip/newicktree.html>.
- [6] PHYLIP, *PHYLogeny Inference Package*, University of Washington, <http://evolution.genetics.washington.edu/phylip.html>.
- [7] C. Bachmaier, U. Brandes and B. Schlieper, *Drawing Phylogenetic Trees*, Department of Computer & Information Science, University of Konstanz, 2005.
- [8] *Java Web Start Overview*, White Paper, Sun Microsystems, May 2005.
- [9] S. Northover, *SWT: The Standard Widget Toolkit*, White Paper, IBM OTI Labs, March 2001.
- [10] M. Schoenmaker, A.J.M. de Craen, P.H.E.M. de Meijer, M. Beekman, G.J. Blauw, P.E. Slagboom, and R.G.J. Westendorp, *Evidence*

- of genetic enrichment for exceptional survival using a family approach: the Leiden Longevity Study.* In *European Journal of Human Genetics* (2006) volume 14, pages 7984, October 2005.
- [11] Wikipedia, *Chromosome* — *Wikipedia, The Free Encyclopedia*, September 2006, <http://en.wikipedia.org/wiki/Chromosome>.
- [12] L. Eronen, F. Geerts, and H. Toivonen, *A Markov Chain Approach to Reconstruction of Long Haplotypes.* In *Proceedings of the 9th Pacific Symposium on Biocomputing*, pages 104-115. World Scientific, January 2004.
- [13] H. Toivonen, P. Onkamo, K. Vasko, V. Ollikainen, P. Sevon, H. Manila, and J. Kere, *Gene mapping by haplotype pattern mining.* In *IEEE International Symposium on Bio-Informatics and Biomedical Engineering*, pages 99-108. IEEE, November 2000.

Part IV
Appendices

Appendix A

Screenshots experiment 1

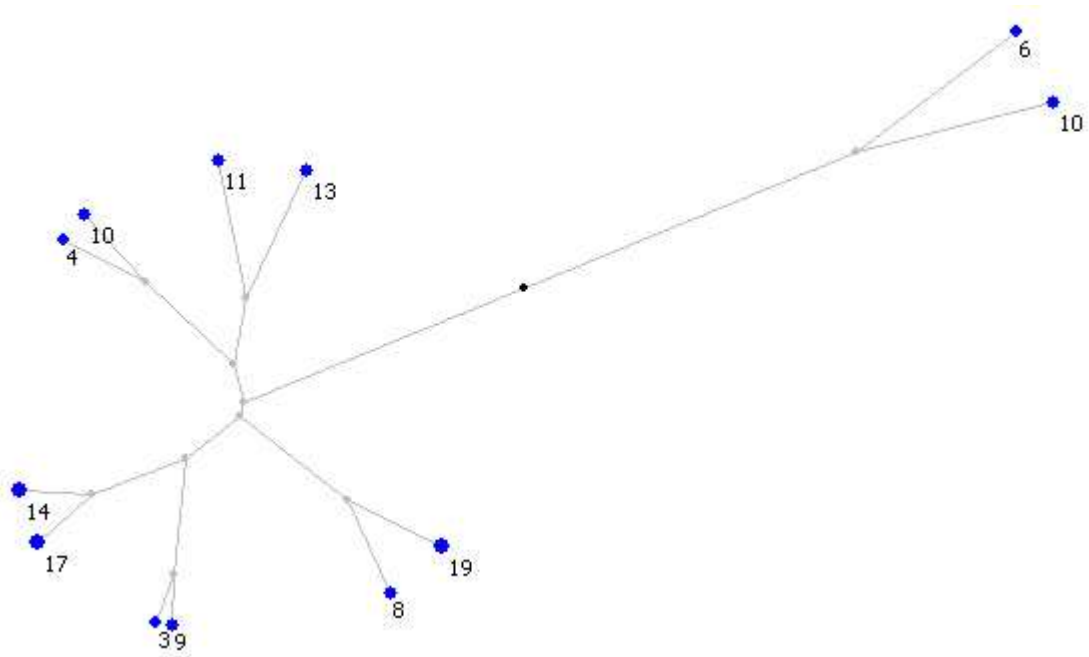


Figure A.1: 12 Clusters at initial clustering level with cluster layout

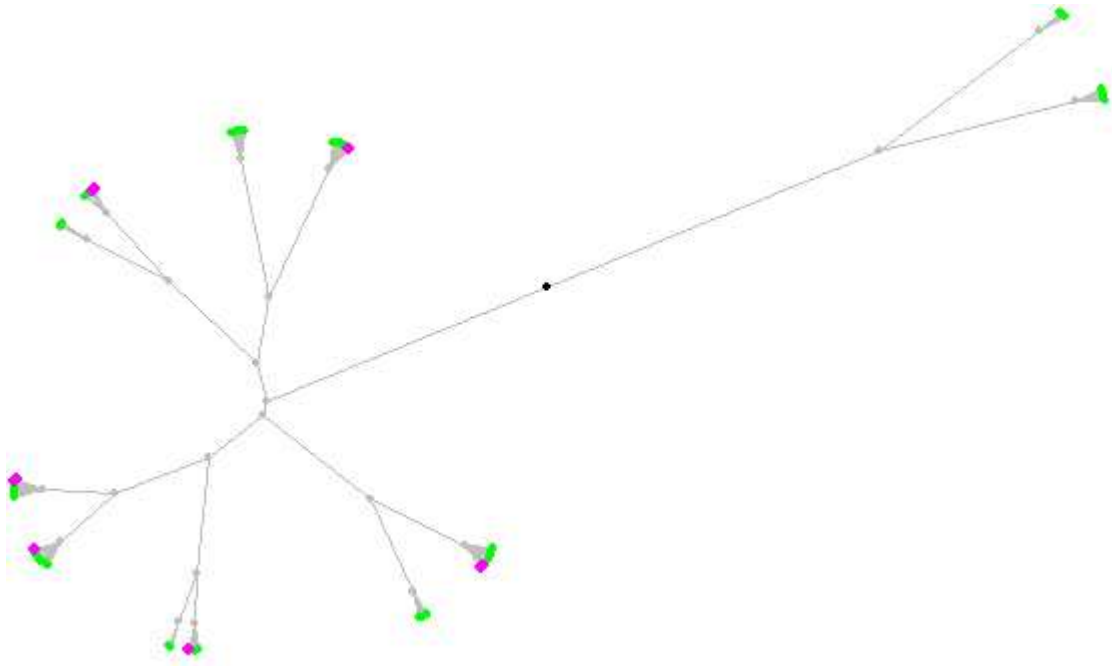


Figure A.2: 12 Clusters at initial clustering level with family layout

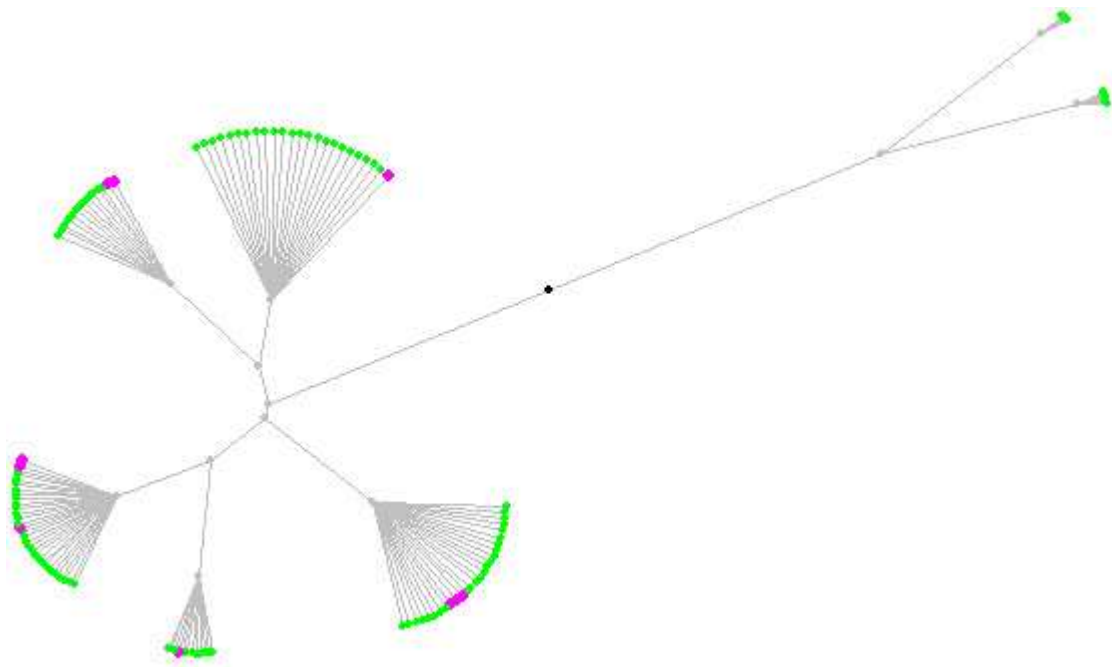


Figure A.3: 7 Clusters at clustering threshold of 0.21 with family layout

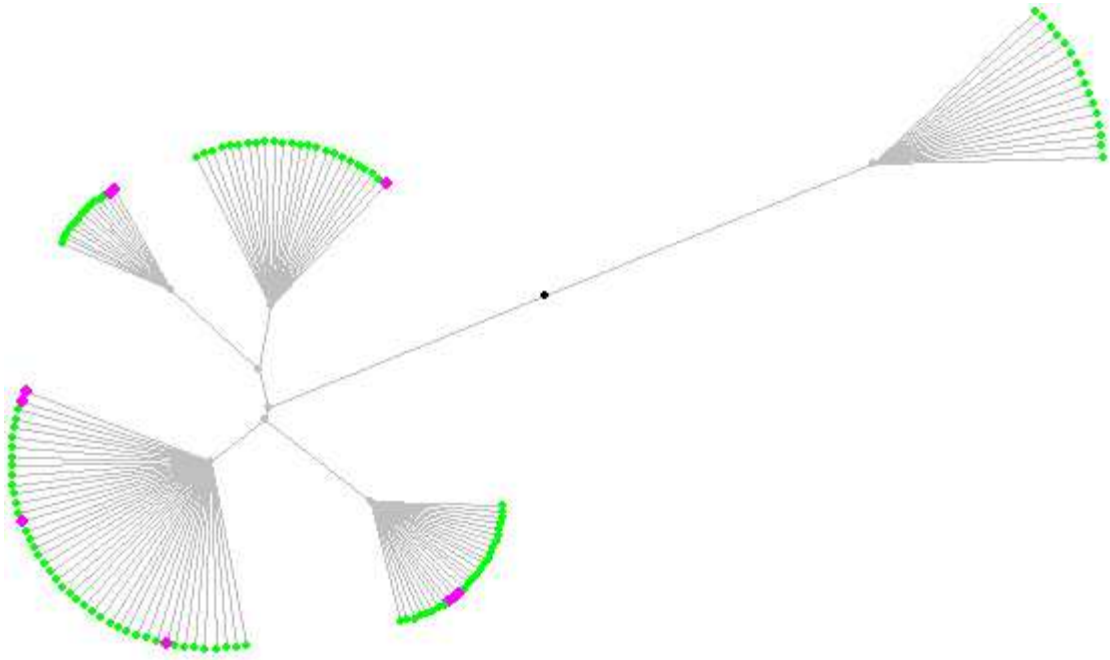


Figure A.4: 5 Clusters at clustering threshold of 0.21 with family layout

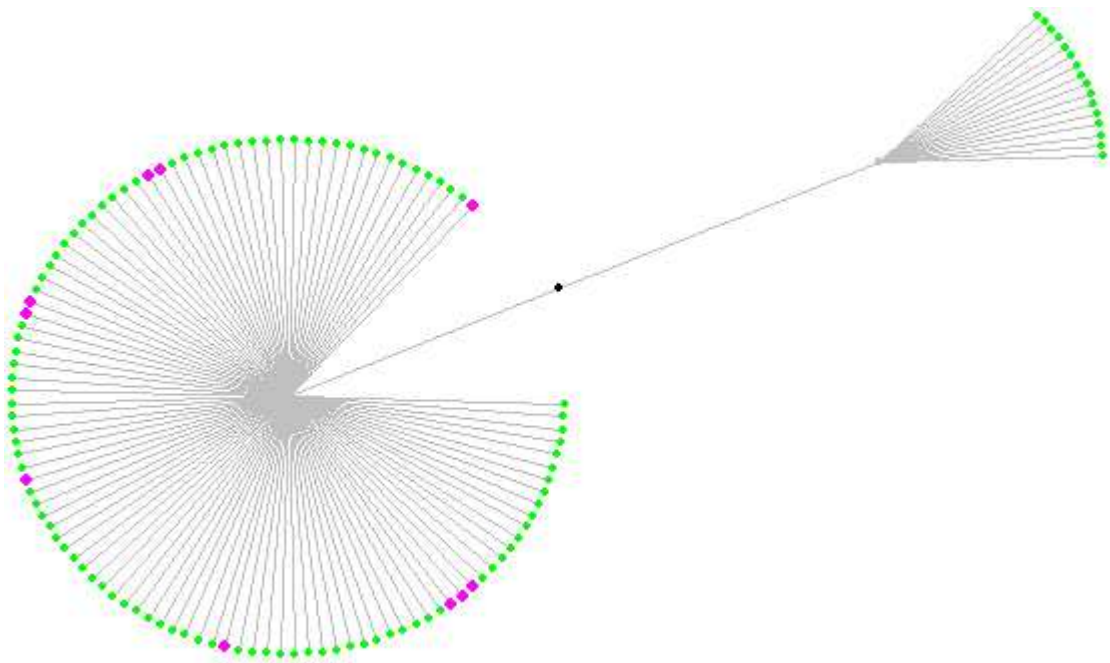


Figure A.5: 2 Clusters at clustering threshold of 0.21 with family layout

Appendix B

Screenshots experiment 2

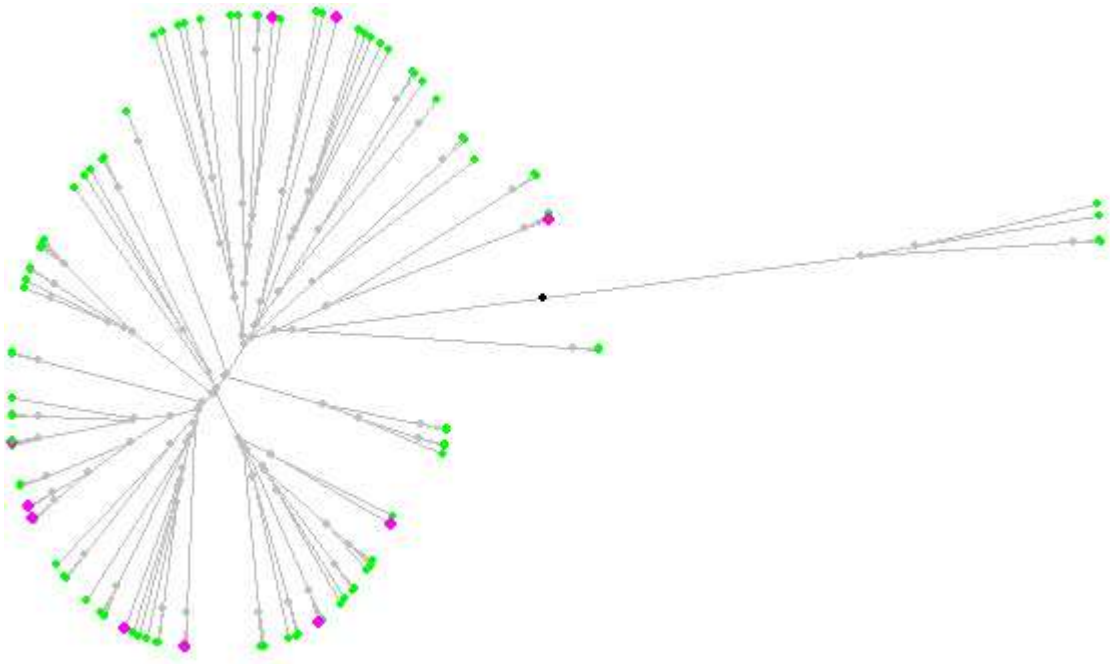


Figure B.1: 59 Clusters at initial clustering level with family layout

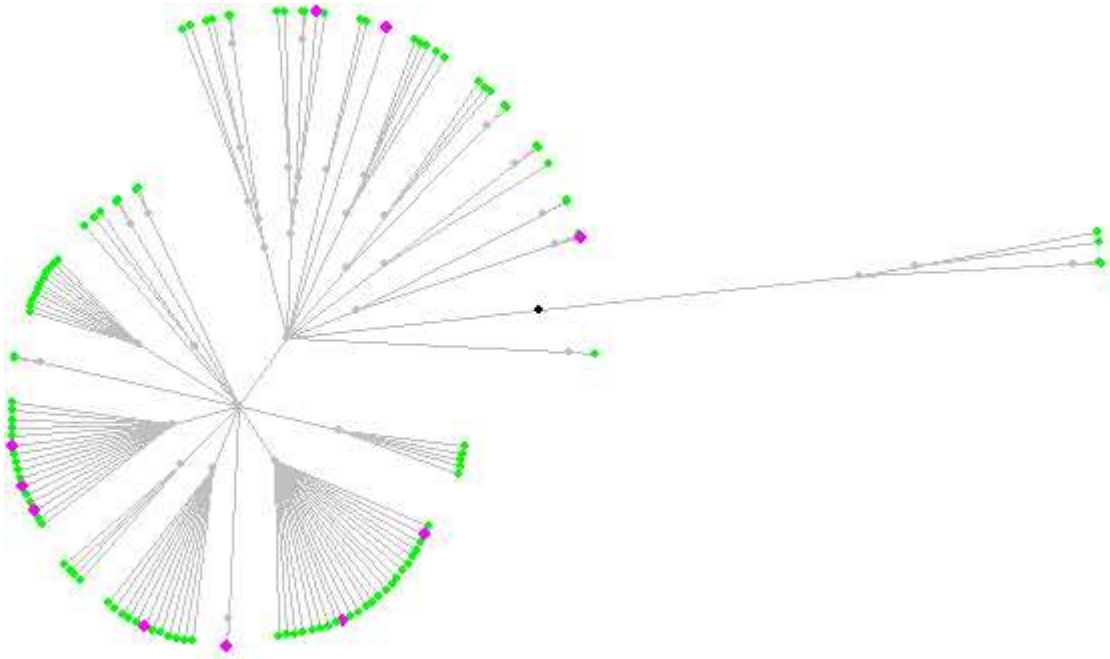


Figure B.2: 32 Clusters at clustering threshold of 0.27 and pruning threshold of 0.04 with family layout

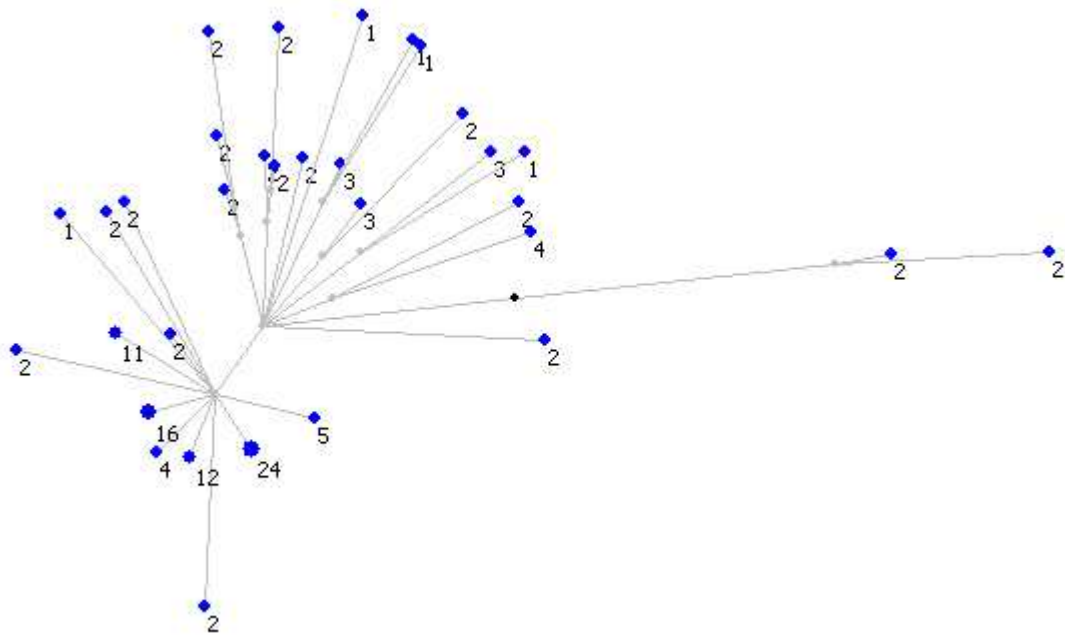


Figure B.3: 32 Clusters at clustering threshold of 0.27 and pruning threshold of 0.04 with cluster layout

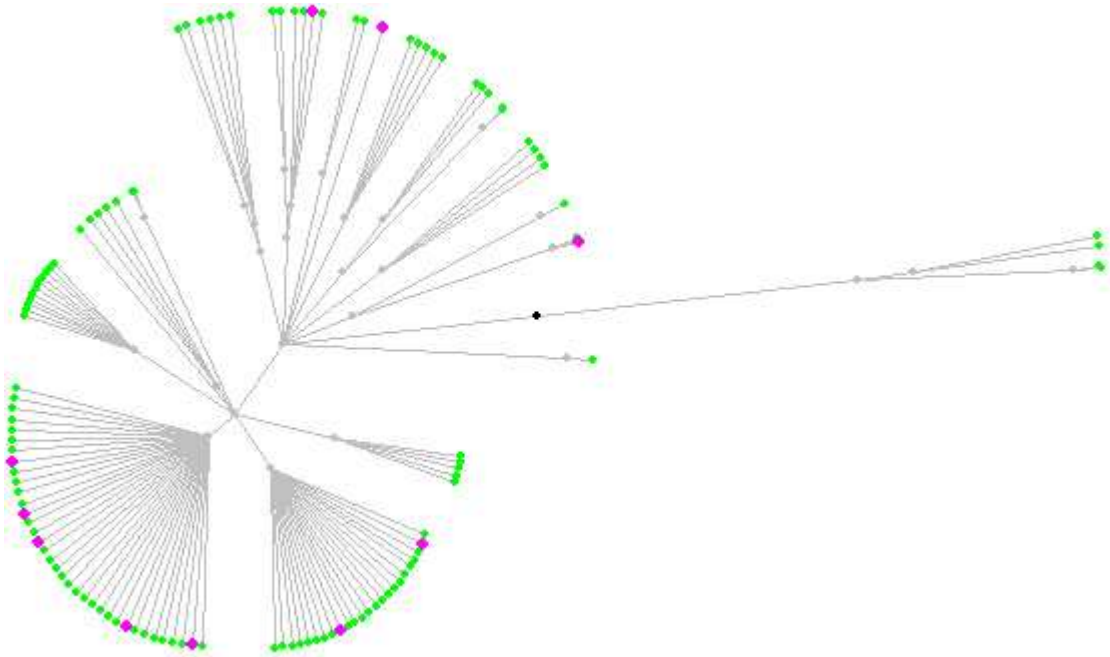


Figure B.4: 22 Clusters at clustering threshold of 0.33 and pruning threshold of 0.04 with family layout

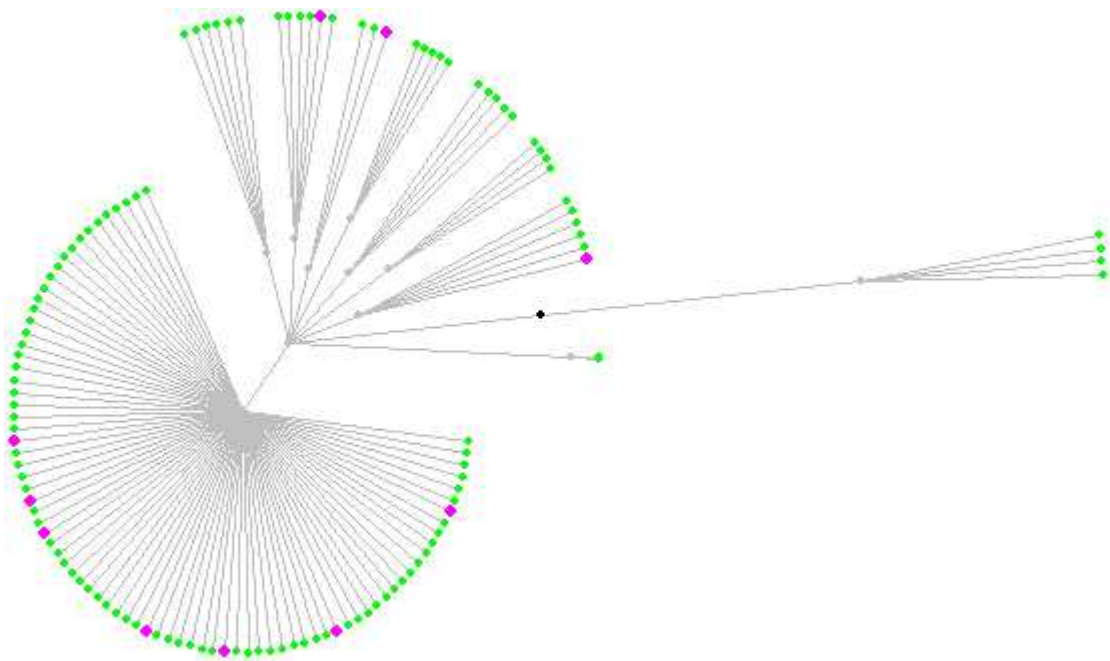


Figure B.5: 10 Clusters at clustering threshold of 0.33 and pruning threshold of 0.12 with family layout