CARINTHIA UNIVERSITY OF APPLIED SCIENCES
School of Engineering and IT
Department of Geoinformation & Environmental Technologies
Graduate Program of Spatial Information Management

MASTER THESIS

SOMatic Viewer: Implementation of an Interactive Self-Organizing Map Visualization Toolset in Processing and Java

Submitted in partial fulfilment of the requirements of the academic degree

Master of Science in Engineering

Author: Manuel RAINER

Registration No.: 1110362012

Supervisor: Dr.-Ing. Karl-Heinrich Anders
Department of Geoinformation & Environmental Technologies
Carinthia University of Applied Sciences, Villach, Austria

Second Supervisor: Dr. André Skupin
Department of Geography
San Diego State University, San Diego, California, USA

San Diego, August 2013

STATUTORY DECLARATION

I hereby declare that

- the Master Thesis has been written by myself without any external unauthorised help and that it has not been submitted to any institution to achieve an academic grading.
- I have not used sources or means without citing them in the text; any thoughts from others or literal quotations are clearly marked.
- the enclosed Master Thesis is the same version as the version evaluated by the supervisors.
- one copy of the Master Thesis is deposited and made available in the CUAS library (§ 8 Austrian Copyright Law [UrhG]).

I fully understand that I am responsible myself for the application for a patent, trademark or ornamental design and that I have to prosecute any resultant claims myself.

Villach, 11 Sept 2013 

Place, date Signature

"Everything is related to everything else, but closer things are more related than distant things."
First Law of Geography (Tobler 1970)

ACKNOWLEDGMENTS

I would like to thank my supervisor at the San Diego State University, Dr. André Skupin, for giving me the chance to finish my Master studies at the SDSU in beautiful Southern California. His support was excellent. We had endless hours of valuable discussions and improvement sessions. His ability to motivate and excite people for his work, giving insights to his research activities, and showing pieces of creative scientific stuff which has not been done by anyone before had a positive effect on the results of my work.

Moreover, I want to show my gratitude to Michael Spöcklberger, who was a great roommate, research fellow, travelling buddy and friend during the whole time in San Diego. We complemented each other very well and our mutual input saved a lot of time and brain drains. Cheers man!

My appreciation also refers to Dr.-Ing. Karl-Heinrich Andres, supervisor at the Carinthia University of Applied Sciences. His feedback and support, together with Dr. Gernot Paulus, encouraged me to head into the right direction for groundwork.

Another important role played the scholarship granted by the Austrian Marshall Plan Foundation, without their financial support I would not have been able to come to San Diego State University. I am indebted for this funding and hope that parts of this thesis can be used for future research activities and encourage people to work with Self-Organizing Maps. This scholarship provides such a great chance for students to go abroad, exchange experiences and expand their horizon.

Finally, and not less important, I have to thank my family who gave me the wings for independence in early years and they still support everything I do. My parents provided the chance to receive a perfect education which is now the basis for my personal development and career.

ABSTRACT

The Self-Organizing Map (SOM) is an artificial neural network, used to determine similarities in high-dimensional datasets through simplified abstractions. This thesis deals with the development of an interactive and integrable SOM visualization tool. The aim of this work is to support students and researchers to accelerate their understanding how to analyze large datasets using the SOM approach and to provide software that is open for modifications. The implemented SOMatic Viewer is a toolset which consist of a *Processing* 2.0 library for SOM visualization and a standalone Java application. The difference to other SOM software is its ability to be used within a larger knowledge discovery workflow. It provides seven popular SOM visualization techniques, from component planes and the U-Matrix to input vectors projected onto the SOM. Both similarity- and topology-based SOM coloring as well as k-Means clustering are integrated. SOM visualizations can be linked to the geographic space to find spatial relationships and clusters. Therefore, loading of referenced Shapefiles is supported. SOMatic Viewer uses an enhanced version of the well-known SOM_PAK file format. Application settings can be saved and restored with project files. The core of the software is a flexible SOM grid and the interactive selection highlighting between all visualizations and data tables. The software can be seen as first release and is intended to be improved by students or by the Processing community in the future. As practical example, SOMatic Viewer is applied to a real-world dataset to analyze the census records of municipalities in the region of Carinthia, Austria. The preprocessing and SOM training procedure together with results and conclusions are given. Another outcome of this thesis is the classification of SOM visualization techniques. A classification matrix which contains 23 visualizations, logically ordered into four main groups, is created. Through the vast collection of SOM visualization methods and latest research activities described throughout this thesis, it provides an interesting overview about the current state of the art in the field of SOM visualization research.

KEYWORDS

Self-Organizing Map, Processing, Java, visual data mining, knowledge discovery, data visualization, clustering, high-dimensional data, U-Matrix, component planes, SOM coloring, interactive design, dimensionality reduction, artificial neural network.

Table of Contents

1. Introduction

This chapter gives an introduction to the applied research work presented in this thesis. It describes the motivation for pursuing this work, investigates the background by discussing used terms and methodologies and lays out the problem statement. Then, the underlying goals and research questions are explained in detail. Further, it gives a summary about the methodology as well as the expected results. An overview about the structure of this thesis is presented at the end.

1.1 Motivation

Dealing with high-dimensional data is demanding work where correct information extraction and interpretation requires specific domain knowledge. In contrast to conventional data mining techniques, a Self-Organizing Map (SOM) provides visually understandable results by showing clusters and dependencies between elements through ordering similar objects close to each other. A variety of visualization methods, from component planes to distance matrices and creative trajectory maps, can be applied and represent multi-facetted information from input records. A SOM is an artificial neural network (ANN) which breaks down high-dimensional data into simplified abstractions and enables preserving topological and metric relationships (Kohonen 1998). When speaking of a high-dimensional data space, it describes the amount of attributes which define a certain object. SOMs help to find complex correlations and allow retrieving valuable information from those data spaces. Visualizations of a SOM are objective and scalable, this makes it easier to define and handle the results. In the manner of how attributes are depicted in geographic space, SOMs can even serve as a serious alternative to conventional maps (Skupin and Esperbé 2011).
This thesis work aims to support students and researchers to accelerate their understanding how to analyze large datasets and to provide new perspectives from various angles. The implemented SOM visualization library has programming interfaces, which allows integrating additional visualization methods and new functionalities, such as clustering algorithms, or other distance measures. Since the emergence of SOM in year 1982 and its first software package, called SOM_PAK (Kohonen et al. 1995), a bunch of software applications and toolboxes have been implemented for SOM training and visualization. Most of them are still updated and have a varied range of functionalities, but they are not designed for specific integration purposes into other workflows and usually not open for modifications too. Seven different software products and add-ins are described in section 2.1.

Data mining tools are complex. It requires some time to get used to the software, whereas there is often the need to get quick results. The present thesis work is intended to give support to that demanding task. The extendibility and portability of this library is a criterion to make it attractive for future use. Therefore, the chosen programming language *Processing* (2013) with its simplified syntax enables extending the library, even in case of limited programming knowledge. As there is no current SOM visualization library available in *Processing,* this is a challenge and great motivation to create a modular SOM toolset and contribute it as an official library to the community. Further, a SOM Visualization tool which integrates the library is developed and used to analyze the sample dataset of Carinthia census data as a practical example of the implementation.
Another major interest of this work is the combination and interaction of all three types of SOM visualizations. In the end, not only visualizations of the SOM itself are provided, there is also a connection of the attribute space with Geographic Information Science (GIS) and visualizations in geographic maps. Working in the rather small research field of Self-Organizing Maps offers the need of open and extendible toolsets. An entire knowledge discovery workflow, for instance, becomes long and tedious and requires integrable and modifiable libraries. The SOM visualization library for *Processing* can be used as a part of an automated computation process with a combination of different data analysis techniques such as dimensionality reduction, spatial clustering and GIS functions. According to the statement of Skupin and Fabrikant in 2003, "the potential of SOMs seems inexhaustible, because they can be used for any kind of attribute data and they provide the possibility of mapping almost everything".

1.2 Background

This chapter provides explanations about the principle of a Self-Organizing Map, the field of visual data mining and knowledge discovery, as well as it discusses the term geovisualization and how it is connected to this research work. As the major part of the implementation is based on the programming language *Processing*, it is also described here.

1.2.1 Self-Organizing Maps

The Self-Organizing Map, also called Kohonen Map, or Self-Organizing Feature Map (SOFM), was developed by the Finish professor Teuvo Kohonen and is a type of artificial neural network (Kohonen 1998). The term 'self-organizing' describes the fact that there is no supervision used. SOMs learn on their own through unsupervised competitive learning methods. The SOM consists of neurons, also called units or nodes, organized on a regular, mostly two-dimensional, grid. The SOM terminology uses a couple of different words for the same terms. A complete

list of equivalent expressions can be found in Table 1. The usual arrangement, so called topology, of these nodes is rectangular or hexagonal, as depicted in Figure 1.

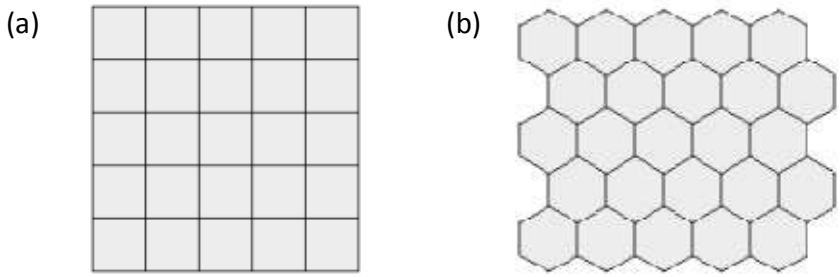


Figure 1: Regular two-dimensional SOM topologies, using (a) rectangular or (b) hexagonal arrangement.

The used topology affects the connection of neighbors. In a rectangular topology a neuron has a maximum of four neighbors, where in a hexagonal arrangement six neighbors are connected with each other. A hexagonal grid is more frequently used and provides smoother transition in visualizations. There are other approaches such as a spherical topology, which is not subject of this work, but shortly described in the literature review, subsection 3.3.2. The SOM uses three different layers (see Figure 2).

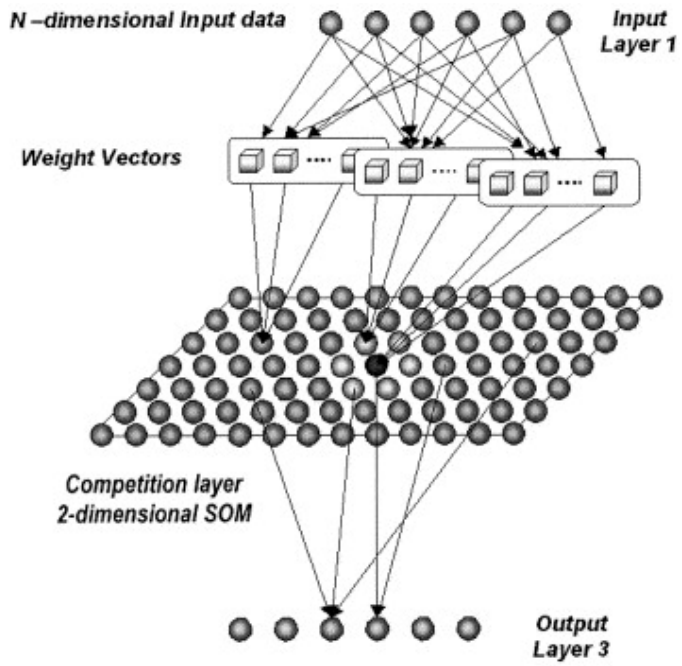


Figure 2: Structure of a SOM. First, initial values are given to the weight vectors from the input layer. During training phase the BMU is determined on the competition layer. A visual representation of the results is done on the output layer (image source: Mongini and Italiano 2001).

There is the SOM competitive layer with its neurons and the input layer containing the high-dimensional input vectors. The resulting output layer contains the topology preserved map units. Each input vector is associated with a set of weights in the same dimension. To understand how a SOM operates, it is useful to know that the algorithm combines two tasks: training and mapping. The training step constructs the map using learning vector quantization (LVQ) as competitive process. During this process, the distance of an input vector to all weight vectors is calculated. Used distance measures include Euclidian, Cosine, or Manhattan distances. The neuron with the most similar weight values to the input vector is called best matching unit (BMU). Once the BMU is found, the input vector is assigned, so-called mapped, to this single winning neuron. The weights of the BMU including surrounded nodes are then adjusted towards the input vector. An example of this process, using a 3x3 SOM arranged in rectangular topology and trained with four input vectors, can be seen in Figure 3.

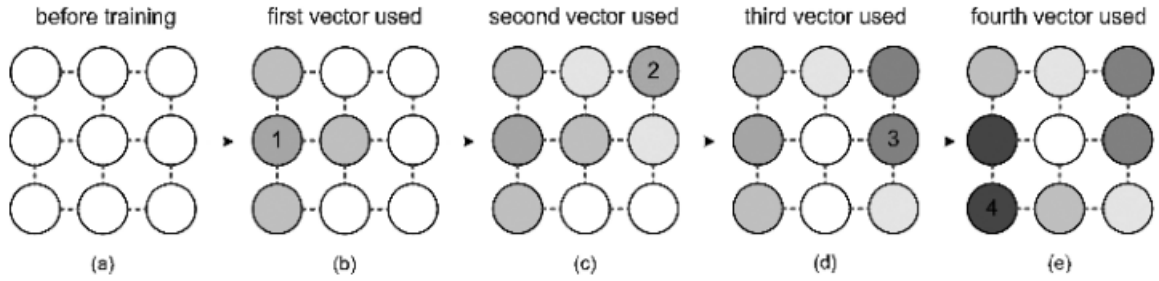


Figure 3: The competitive learning process from initialization of the neurons (a) to the adjusted weights (e) from four input vectors after best matching unit search (Skupin and Agarwal 2008).

This process is repeated for each input vector for a usually large number of training iterations. The magnitude of change, so called training rate, as well as the distance measure is reduced over time. This also reduces the number of weights that get updated per iteration. Figure 4 illustrates the non-linear projection during training in the high-dimensional input space. The projection is restricted to the map topology. This topology-preserving mapping shows that the more similar two data samples are in the input space, the closer they will appear together on the final map. In other words, it preserves the relative distance between the points. The visualization of a SOM allows cluster identification and pattern recognition. Therefore, SOMs operate both as a kind of visual similarity graph and clustering diagram. It comprehensively visualizes natural groupings and relationships and has been successfully applied in a large spectrum of research areas ranging from speech recognition to biomedical analysis. Due to the fact that SOM enables low-dimensional views of high-dimensional data, it is restricted to its grid projection.

Enhanced algorithms, which tackle this limitation, are discussed in section 3.3. In recent years, research brought up some alternatives and extensions to the basic SOM technique, such as the growing (hierarchical) SOM or the time adaptive Self-Organizing Map to name only two of them. These methods are not described and dealt within this thesis work.

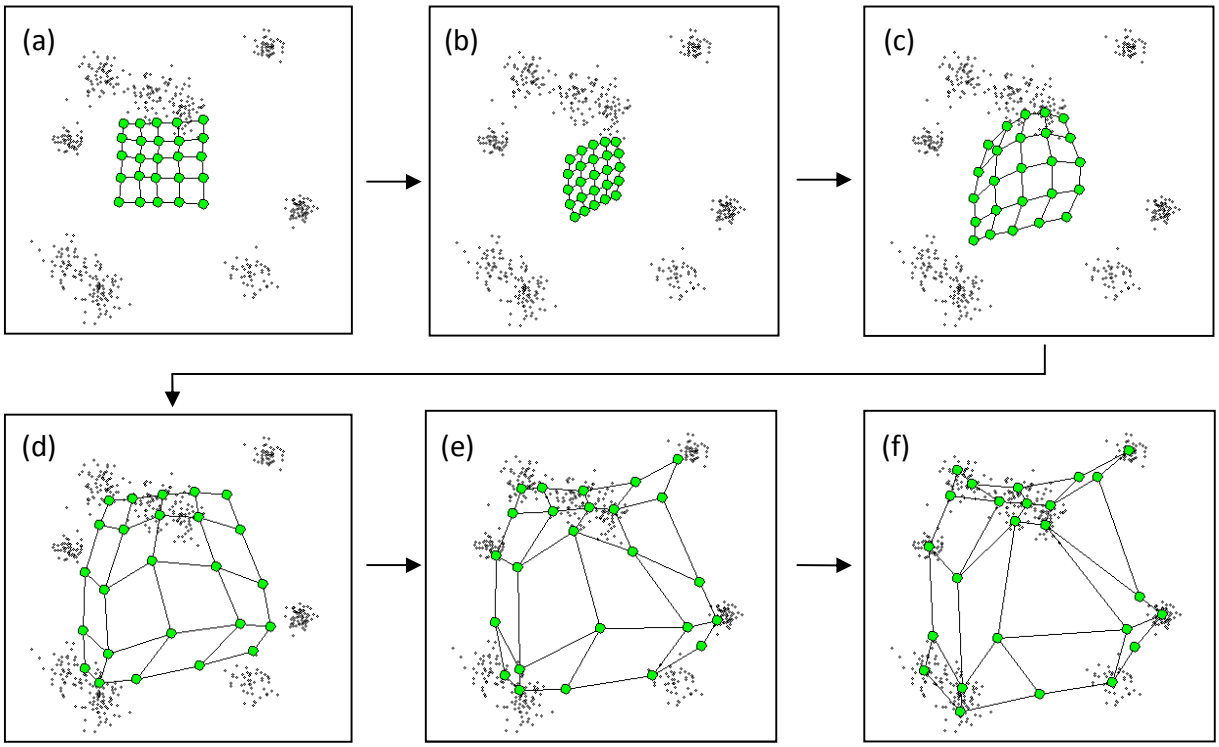


Figure 4: Non-linear projection of a 5x5 SOM, where the nodes are iteratively moved towards their best matching units in multi-dimensional input vector space (image source: http://www.peltarion.com/doc/index.php?title=Self-organizing_map).

SOMs are applied in any fields of science where high-dimensional data needs to be visualized and relationships need to be found. Examples for such application areas are multispectral remote sensing imagery, biomedicine, robotics, socio-economics, finance and trading, or climatology.

SOM Terminology			
	equivalent to…		
Grid	Lattice	Mesh	
Neuron	Unit	Node	
Attributes	Components	Weights	
Codebook vector	Prototype vector	Model vector	Feature vector
Input vector	Training vector		
SOM space	Output space		
Original space	Input space		
Attribute space	Data space		

Table 1: SOM Terminology.

1.2.2 Visual Data Mining and Knowledge Discovery

As SOM is a technique used for visual data mining and knowledge discovery, these two terms require some further explanation. Traditional methods for extracting knowledge from data involved manual analysis and interpretation. Since in all fields of science, society, and industry massive amounts of data are collected, there was the need of computational theories and tools to assist humans in retrieving useful information from the rapidly growing volumes of data (Fayyad et al. 1996). For this purpose, knowledge discovery in databases (KDD) emerged. KDD deals with the development of techniques to make sense of data, by providing a more compact, abstract, and more useful view on voluminous datasets. To handle the problem of data overload, data mining, as advanced analysis step in the KDD process, applies specific methods for pattern recognition and extraction. Machine learning, artificial intelligence and statistics provide the technical basis for data mining. Visual data mining aims at integrating the human's perceptual abilities, presenting the data in some visual form, getting insight to the data, drawing conclusions, and directly interacting with the data (Keim 2002). To find hidden information in the data, visual data mining as a technique for knowledge visualization uses representations in multiple dimensions and hierarchies. Varied display technologies and methods for interaction, such as zooming, filtering, or projection enhance this interdisciplinary research field. Because of limitations in human visual and cognitive processing restricted by large volumes of numbers and objects in a two dimensional map (Koaua 2003), a SOM visualization framework provides powerful functions to

analyze geospatial data which meet exactly these requirements. Figure 5 shows such a framework for exploratory data analysis using Self-organizing Maps, where knowledge discovery is performed using a computational data mining process followed by visual interpretation, and applying the results to the user domain tasks.

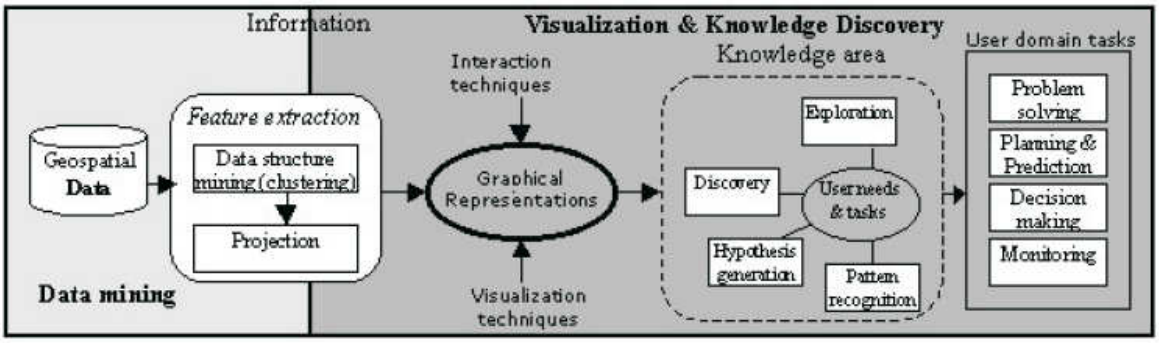


Figure 5: Data exploration and knowledge discovery using a SOM data mining and visualization framework (Koaua 2003).

The highest benefit of a visual data mining is the simple way to deal with highly inhomogeneous and noisy data and its intuitive process, which requires no understanding of complex mathematical or statistical algorithms or parameters (Keim 2002).

1.2.3 Geovisualization

When visualizing SOM data in geographic and attribute space, the present work consequently aims towards the fields of geovisualization. The comprehensive definition of geovisualization unites various visualization methods, such as exploratory visualization and information visualization, scientific visualization, cartography, image analysis and GIS (Dykes et al. 2005). A SOM can be applied for and combined with all these representation methods. Using a SOM as part of a knowledge discovery process, analyzing AAG (Association of American Geographers) paper abstracts from the last 20 years for instance, the results are in the domain of information visualization. If the color coded neurons are linked to a geographic map, then one is speaking of a cartographic visualization. More complex combinations of visualization methods with GIS, such as the so-called spatialization, done by Skupin and Fabrikant (2007), are handled in section 3.1. Spatialization uses geographic metaphors to visualize non-geographic data and makes use of human's perceptual capabilities. It depends on the application area, the used datasets and the combination of methodologies to be able to speak of a specific visualization. Therefore, the SOM visualizations mentioned in this thesis all depend somehow to the area of geovisualization. According to Dykes et al. (2005) geovisualization can be described as a loosely bounded domain that addresses the visual exploration, analysis, synthesis and presentation of geospatial data by integrating approaches

from cartography with those from other information representation and analysis disciplines. SOM combined with geovisualization methods provides intuitive and experimental representation methods which enhance the understanding and knowledge retrieving of increasingly large and complex geospatial datasets (Koua 2003). When dealing with large datasets, the users tend to need interaction and dynamic visualizations. After researchers found out that visualizations had not taken advantage of exploiting the full potential of geospatial data, a commission within in the International Cartographic Association (ICA) was created. This Commission on Visualization and Virtual Environments then came up with theories, practices and tools for geospatial data exploration, analysis and knowledge retrieval (MacEachren and Kraak 2001). Today, it has become the Commission on GeoVisualization and continues the work of the former commission which has been establishing the emergent discipline of geovisualization since 1995 (ICA 2013).

1.2.4 *Processing* 2.0

The programming language and integrated development environment (IDE) *Processing* is used for the implementation of the SOM visualization library. It is open-source and purely based on the Java programming language. *Processing* is popular for creating images, animations and interactions (*Processing* 2013). It has a simplified syntax and allows fast development of visualizations, so called sketches. Since the community is rapidly increasing and hundreds of modules and thousands of code examples have been published (OpenProcessing 2013), it became a serious environment for the development of professional work in the fields of electronic arts, visual design, and recently in scientific research areas. Besides drawing in two dimensions, *Processing* provides accelerated 3D using OpenGL. The simplicity compared to other programming languages makes it easier for people who want to do follow up enhancements of the SOM visualization library, even though they may not have extensive programming skills. The *Processing* sketchbook can be exported into JavaScript for website integration, or embedded into a Java application. These features make it a universally integrable visualization toolkit. For the present thesis work, the Eclipse IDE was used for developing the SOM visualization library which extends the *Processing* PApplet class. The SOM visualization library was then embedded into a Java SWING application. As part of the preliminary work, some performance testing with *Processing* sketches, running as JavaScript, Java Applet and integrated into a standalone Java application, was done (see section 4.1).

1.3 Problem Statement

First of all, the SOM visualization library needs to be extendable and reusable. It is necessary to give an exact documentation about the programming interfaces to ensure that subsequent developers have a source to understand the architecture and to extend the software. Also the integration of functionalities has to underlie

these principles. As already mentioned, there is no available SOM visualization library in *Processing* right now, which requires a lot of groundwork. The structuring and creation of classes has to be done based on the guidelines for contribution, so that it is working properly within the *Processing* Development Environment (PDE).
Another challenge will be to design and implement a tool which fulfills not only the technical requirements, but also meets the interaction and performance needs. Because of usability reasons, a simple and understandable user interface is taken into account. This user interface needs to be decoupled from the visualization itself. Certain visualization methods and clustering techniques need to be adjusted with the use of parameters, for example to set the distance measurement between objects or to define the minimum number of objects within a cluster. However, the integration of the SOM visualization library into the Java application must be done without any direct references from the visualization library to the user interface. Moreover, what if the user wants to explore the data using two different visualizations? A solution for displaying multiple frames and interactive selection has to be investigated. An issue is the granularity of the library modules, because a novice user might want to have a component planes class which can be initialized with a few parameters. Another one might only want to use the core methods for creating another customized SOM visualization, so the whole toolkit has to be more a white-box rather than a black-box.
Then, the SOM input file format is an important factor. What are common data formats used in SOM software tools? In addition, there has to be an effective way to process these mostly large amounts of data, which leads back to the performance of the system. An inspection of available SOM tools will help to find out more about data formats as well as to get some impression of how to design a proper user interface. The results of this tool testing can be found in section 2.1.
The intention is to integrate two-dimensional visualization representations of the SOM. The topology of a neural network can be rectangular or hexagonal, which has to be considered when visualizing the data. Basic geometric algorithms need to be implemented to draw the shapes and automatically adjust the map according to frame size. The SOM should be visualized with at least one technique out of three main visualization groups, described in section 2.2. Further, clustering is an aspect that corresponds to the visualization. Therefore, besides representations of the SOM neurons and the projection of input vectors, the integration of a clustering algorithm applied on the data and visualized in the SOM is considered. The integration of SOM visualization in geographic space is another essential task within the present thesis work. The way how to integrate SOM data into geographic map needs to be investigated. As *Processing* is a rather young programming language, the capabilities for geographic data mapping might be limited. Thus, this task requires research on available libraries which provide necessary functionalities. An animation of training steps should give the user a visual representation of the evolution of a Self-Organizing Map. The same can be done with clustering iterations where one can interactively see how clusters are formed. There is the question if the animation

can be done in real-time which would need an additional communication interface with another SOM training tool, or by reading SOM data with training iteration steps from previously saved files.
The present thesis work also tries to outline the various known SOM visualization methods. There is the intention to provide a compact classification of all found techniques. As visualizations vary in their representation style, algorithms and information content this requires a method to categorize each based on common characteristics.

1.4 Research Questions

The three research questions and their detailed description derived from the problem statement are as follows:

1. **How can an extendible SOM visualization library be developed for *Processing*?**

What are the requirements for successful library integration into the PDE and the contribution to the community? How to cope with the PApplet inheritance in a large project where multiple sketches are running simultaneously? Under the extendibility aspect, which interfaces have to be defined? To which granularity needs the modularization be done to guarantee useful method access to the user? Then, what are the limitations of a library, written in mostly pure Java, which references the *Processing* core and other libraries? How can the data be read and stored? In general, what should the input data look like? Also, what is an effective way to run multiple *Processing* SOM sketches simultaneously and keep them all interactively selectable?

2. **How can a SOM visualization tool with integrated *Processing* library be implemented?**

The first thing to clarify here is if the software should be developed for server or client side application. What are the criteria for that decision? Are there any issues when embedding the *Processing* SOM visualization library into a Java application? What does an appropriate user interface look like? What is an effective way to control the sketches from the user interface and enable simultaneous changes in all windows? What kind of further data exploration functions can be added?

3. **What are methods for SOM visualization classification?**

What kind of SOM visualization techniques have been developed and published? In which representation format should the classification result? What are suitable classification criteria? In general, is it possible to get a complete and comparable list or overview of SOM visualizations?

These research questions will be dealt throughout chapters 4 and 5, respectively the applied methodologies and thesis results, and are going to be explicitly answered in the conclusions, section 7.

1.5 Methodology

The SOM visualization library is implemented using the open-source programming language *Processing 2.0*, which is mostly simplified Java code. The software design is based on object oriented programming, using packaged class modules and communication interfaces. The toolset uses an enhanced version of the SOM_PAK file format (Kohonen et al. 1995) for data input. This file format was specifically modified to support the implemented visualization methods and is backward compatible to SOM_PAK. The SOM is drawn according to the data read from a codebook file, containing the description of the SOM itself. Data from input vector files is projected onto the SOM. Seven different visualizations are implemented. Component planes, hit diagram with labeling and marker symbols, hit histogram, the U-Matrix, as well as k-Means are visualizing and clustering the SOM space. For the representation of the SOM in geographic space, the *Processing* library MapThing (Reades 2013), which is base on the Java GeoTools GIS library (OSGeo Project 2013), is used display and modify Shapefiles. Geographic features are color-coded according to the applied visualization in the linked SOM. The SOM coloring classes are using a commonly defined interface. Visualizations are interactively connected using a global variables class. This Singleton pattern class (Vlissides et al. 1995) keeps the data and parameters in memory, and is accessed and updated from each of the visualizations. The visualization settings and file paths can be saved and loaded as project files. An animation effect during SOM clustering or SOM training can be achieved through step-wise updating of the vector attributes while running the *Processing* sketch in an infinite drawing loop. There are two different approaches for coloring the SOM space. One method simply creates two-dimensional plane of RGB (Red Green Blue) colors and stretches it over the SOM. The other one uses SOM training to create a one-dimensional color SOM from the codebook vectors, where colors are assigned to each neuron according to their BMU in the color SOM. Diverging-diverging color schemes and automatic color picking from HSB hue circle are implemented. The SOM visualization library for *Processing* is embedded into a Java application, which uses SWING user interface elements. Multiple windows allow interactive exploration and comparison of the visualized SOM and input data. The global variables class from the *Processing* library together with event listeners is used for interactive highlighting between all visualizations and data tables.
A criteria matrix is used for the classification of SOM visualizations, which allows a compact visual comparison of 23 representation techniques. The classification characteristics are elaborated from common differentiation features, for instance if

vector distance or density is visualized or cluster connections are drawn. This task requires an extensive literature review and involves the testing of SOM visualization tools for gathering a basic practical understanding of certain techniques.

1.6 Expected Results and Significance of Work

First of all, the aim of this work is not the development of highly complex visualizations or inventing new ones, rather, it is to provide a basic and working toolset of common SOM visualization techniques. The implementation results should be used as growing and evolving tools for SOM visualizations and data exploration, where users can implement new functions and representation techniques. The two primary results are, on the one hand, the technical implementation of a SOM visualization library in *Processing*, the so called SOMatic Viewer library. This library should be contributed to the *Processing* community. On the other hand, there is the development of SOM visualization software in Java which uses the library as integrated part. As sample dataset, the census data of Carinthia, a region in Southern Austria, is visualized with the implemented SOM toolset. The analysis results are discusses at the end of this thesis.

Additionally, as theoretical part of this thesis, a list of known SOM visualization techniques are collected and ordered into a classification matrix to provide a current state of the art overview for SOM visualizations, based on their main characteristics.

The significance of this work can be seen in the fact that it is the first extensive SOM visualization library implemented for the *Processing* IDE. This is also the first which analyzes census data of Carinthia using the SOM visualization approach. The data preprocessing and analysis should be an example for others to follow up with further research interests on the dataset. Then, the classification matrix for SOM visualizations is, according to the current state of literature review, the first which tries to create a comprehensive overview about the types of available SOM representations and their characteristics. It can be seen as prototype which allows getting an imagination of the varied visualization possibilities of SOMs, logically grouped and classified.

1.7 Structure of the Thesis

To provide an overview about the subsequent parts in the present thesis work, the next chapters and their associated content are given below:

Self-Organizing Map Tools and Visualizations (chapter 2)
Seven SOM software tools are tested and described with their capabilities and how they differ. After that, the range of available SOM visualizations is explained in detail.

Literature Review (chapter 3)
This chapter deals with fundamental information about research activities, common methods and improvements of the SOM. Besides the description of related work in the domain of SOM visualizations, popular clustering techniques and enhanced SOM algorithms are discussed.

Methodology (chapter 4) and **Results** (chapter 5)
The design and implementation workflow is given in the methodology chapter. This includes used methods, libraries, and algorithms. A description of the resulting SOMatic Viewer toolset as well as the final SOM visualization classification can be found in chapter 5.

Discussion (chapter 6)
The proof of concept compares the implemented visualization methods with the same ones from other software tools to show if they are correct or not. Further, the analysis of the Carinthia census dataset is discussed here.

Conclusion (chapter 7)
Conclusions and further concerns, based on the findings and outcomes of the thesis work, are presented. Moreover, the research questions are explicitly answered here.

Outlook and Future Work (chapter 8)
This last chapter gives an outlook about potential enhancements of the SOM visualization software and exemplifies possible upcoming development steps.

2. Self-Organizing Map Tools and Visualizations

Since Kohonen introduced the SOM algorithm, several software applications and tools have been developed. The first section describes seven different SOM software applications and toolboxes. The second section consists of SOM visualizations, ordered into four main categories. Among traditional methods such as component planes or the U-Matrix there are also recently developed visualizations.

2.1 SOM Tools and Software

This section presents four freely available SOM tools, one commercial software, and two software extensions for SOM training and visualization. The focus here is less on

the training aspect rather than on the given file handling, processing workflow, functionality range, and visualization capabilities. Further, the integration and visualization of geographic data was critical too. The SOM software tools are briefly described and a conclusion is given afterwards.

Standalone Software. The first software package was SOM_PAK, developed by Kohonen et al. (1995) in 1992. It was updated until 2004, but is still in use and a popular SOM component which can be integrated into newer SOM applications. SOM_PAK is written in the programming language C and comes without graphical user interface. Commands for SOM initialization, training, and visualization have to be manually typed into the console. Component planes, U-Matrix, and Sammon's mapping are the only visualizations, saved as PostScript (PS) file. Another well-known software for SOM visualization is GeoVISTA Studio (PennState 2013, Takatsuka and Gahegan 2002). It offers an environment for geospatial data analysis with various functional components, so-called JavaBeans. This approach of creating reusable models by weaving beans into a workflow is unconventional but offers a lot of opportunities. Models can be shared with others and serve as functional part of a larger workflow. The variety of SOM visualizations is limited, but includes the most common ones together with a 3D representation of the SOM. However, its extendibility and the combination with build-in analysis tools make it to an interesting alternative to traditional SOM-specific software. Especially geospatial datasets which need to link their output to a geographic map can benefit from its features. A sophisticated implementation with a vast number of visualization methods, clustering and quality measures is the Java SOMToolbox, developed at the Vienna University of Technology (2013). It contains by far the most extensive functionality range, but this has the drawback of a flood of settings and menus. The way from getting into the tool to the first SOM visualization result can be long and requires technical and subject matter understanding. The input data concept is reasoned but also includes many different files. The biggest advantage comes from their self-developed visualizations in the viewer which cannot be found in any other software so far. Compared to the Java SOMToolbox as expert tool, the next one is rather for novice users and therefore straightforward with only three different visualizations. The name of this software is SOMVis (Guo 2005, 2013), developed in Java as well. It solely accepts Shapefiles with comma separated value (CSV) files. The SOM, parallel coordinate plot (PCP), and geographic map view are linked and allow interactions. The SOM coloring methods are highly adjustable, the SOM clustering very user-friendly. The only issue of this simple tool is the restricted SOM dimension and it gives no information about the training or mapping details. All mentioned solutions are freely available, and mostly open-source. On the other hand, there are a few proprietary SOM applications too. One of those which needs a closer look is the SOMine software product (Viscovery 2013). It is a very complete product with high visualization performance and good usability. Besides its own input file formats it can also handle SOM_PAK codebook files and statistical data

from SPSS. The visualization, clustering and analysis capabilities are complex, but do not require such a long learning curve as for the Java SOMToolbox. SOMine is implemented in Visual C++. Only the trial version was tested which is limited in its functionality. The special feature of SOMine is the project workflow, which splits up the data preparation, processing and evaluation into separate logical steps. The visualizations are not as varied as in Java SOMToolbox, but provide extensive functional parameters and analysis options. Its primary focus in not on geospatial data visualization, thus it does not provide geographical maps in any form.

Software Add-Ons. Among others, there are two interesting extensions for existing software products, namely the SOM Analyst (Lacayo and Skupin 2007) for ESRI's ArcMap, and the SOM Toolbox for MATLAB (Vesanto et al. 1999). SOM Analyst was developed from a student project and serves as toolbox for data preprocessing and SOM training in ArcMap, using SOM_PAK as training core. The map can be visualized with common ArcGIS tools. This provides on one hand a great flexibility for combining geospatial data and applying geoprocessing functions, but on the other hand most popular and necessary SOM visualization techniques are missing. At that point, SOM Toolbox for MATLAB suits better, which has a lot more visualization capabilities. But, the toolbox requires basic understanding of the MATLAB environment and also of its syntax for more sophisticated training and visualization. The SOM Toolbox comes with simple GUI frames that offer common functions for data preprocessing, training, and visualization. As MATLAB is widely used environment for technical and statistical data analysis and representation, this toolbox is a popular SOM software in that domain. Visualizations cover various kinds of SOM topologies and vector projections including the most common techniques to color the SOM space. Geographic maps cannot be loaded or linked which is a remarkable difference to SOM Analyst and thus makes it less suitable for geo-referenced training data.

Conclusion. The variety of SOM tools is as large as their range of functionalities. There is no one-for-all solution which is fast to learn, easy to use and offers all functions for the analysis of spatial and non-spatial datasets. Each of the mentioned software tools has its strengths for one more particular purpose. This is exactly what drives the usability, functionality, and complexity aspects of the tools. Even though most of them have their own data formats, common SOM_PAK files are occasionally supported by newer software. One reason might be that the data formats follow a similar content structure. Interesting approaches are the component-oriented framework of composing functional modules with JavaBeans in GeoVISTA and the workflow-oriented data processing and evaluation in SOMine. Another important fact was the introduction of project files. The more complex and comprehensive a software, the more it makes sense to read and save these settings and processing statuses into separate files. This saves time and is convenient for keeping track of training and visualization parameters and such. Further, SOM

extensions for ArcMap and MATLAB are doing a good job, but require some knowledge of how to use these products before being able to run the SOM add-ons. A powerful aspect here is, that the data can be further processed in the given software environment. There are also simple tools available, but they have a very limited application area as described for the SOMVis software. But simple does not automatically mean bad; it comes with great SOM coloring methods and fast clustering results for the attribute and geographic space. Unlike SOMVis, there is usually a long learning curve to get used to the software and to understand the processing workflow. The present thesis work is intended to bridge the gap between expert and novice users. The use of a SOM tool will always require domain knowledge to a certain degree, but there are ways to simplify the file handling, SOM control and interaction for a fair amount of visualizations.

2.2 Visualization Techniques

There are basically three groups of SOM visualizations, introduced by Skupin and Agarwal (2008). Visualizing the SOM grid itself, mapping data onto the SOM and linking data from the SOM to other visualizations. Another category is encountered during this thesis research and added as fourth group, which describes projections from the SOM space to other representations. Vesanto already came up with three categories of SOM visualizations in 1999, but they are slightly different to the ones used in this thesis and do not consider linking to geographic visualizations.

2.2.1 Visualizing the SOM itself

The SOM itself allows visualizations showing component planes, clusters through applied cluster algorithms, as well as interneuron distances and density in the data space.

Component Planes. This popular visualization method slices the SOM into separate component planes to see how the values of certain attribute, also called component, vary on different locations on the map (Himberg et al. 2001). Each plane contains the values of a single variable of the input vector in each node of the SOM. When using component planes, the number of maps increases according to the selected or displayed number of variables. Component planes are perfect visualizations for correlation detection because even partial relationships or correlations can be found by visually side-by-side comparison of different planes (see figure 2). An intelligent way for correlation hunting through a rearrangement of the component planes was shown by Vesanto (1999). Based on their correlation, similar looking component planes are automatically placed near each other, which results in a more efficient comparison capability.

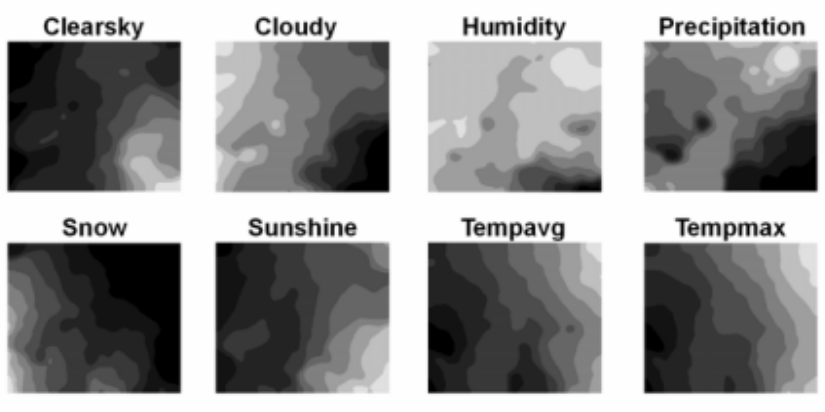


Figure 6: Component planes of a high-resolution SOM constructed from climate data (Skupin and Esperbé 2008).

Vector Fields. Two kinds of SOM cluster structure visualizations based on vector fields are developed by Pölzlhuber et al. (2006). On the one hand, there is the gradient field visualization which projects an arrow on each of the neurons that points to the center of a nearby located homogeneous area. The calculation of the length and direction of the arrow is based on the prototype vector, the map topology, and the size of the neighborhood kernel. Figure 7(a) illustrates the result, a smoothed vector field which outlines clusters in the map. On the other hand, there is the borderline visualization which shows an alternative representation of the cluster boundaries. It is derived from the gradient field, but doesn't use arrows for representation. Instead, it draws the orthogonal of each arrow as a line from both sides of the center. The length of the lines has the same purpose as the length of the arrow. It depicts the magnitude of cluster separation. When looking at the entire map in Figure 7(b), this representation forms a kind of cluster boundaries in the map.

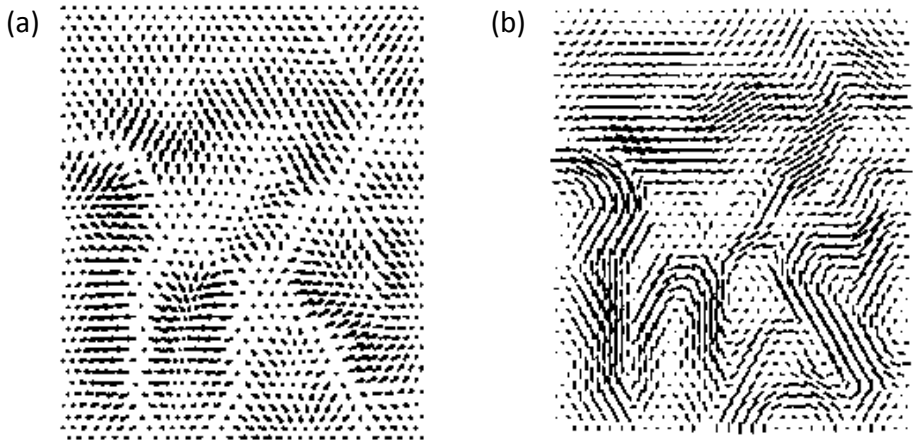


Figure 7: Vector fields. (a) Arrows are pointing to a cluster center and result in a smooth gradient field. (b) Similar method showing cluster boundary lines (Pölzlbauer et al. 2006).

Cluster Connections. This technique draws a grid of connected nodes onto the SOM with edges showing their mutual similarity (Merkl and Rauber 1997). The degree of connectivity is based on the distance between two neighboring neurons. Neurons are connected if they are similar to each other and thus belong to the same cluster. The similarity threshold can be parametrically adjusted. If they are outside or along a cluster boundary, no connection is drawn. A grey-scale coloring represents the distances. This visualization method looks similar to the distance matrix or U-Matrix. Figure 8 shows the result on a SOM with rectangular topology.

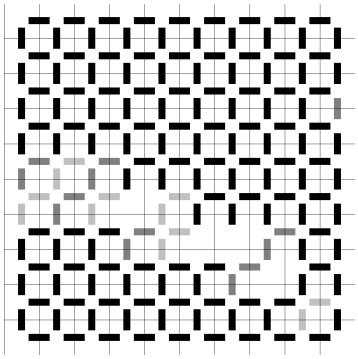


Figure 8: Cluster connection visualization where nodes from same clusters are connected. The color of an edge indicates the distance between the neurons (Merkl and Rauber 1997).

U-Matrix and other Distance Matrices. The unified distance matrix or U-Matrix calculates the Euclidian distance from each unit center to all of its neighbors. The distance to adjacent neurons is presented using a gray scale or color range representation on the map grid. It is an effective method to find clustering structures. Data clusters can be seen as valleys and borders are depicted as mountains or ridges (Himberg et al. 2001). There are different types of distance matrices, where two of them are called U-Matrix. The original U-Matrix keeps the dimensionality of the SOM grid. Another kind of U-Matrix is visualized with interpolated neurons between each pair of neurons which creates a larger grid than the original one. The third type is a distance matrix which preserves the dimensionality but draws the neurons in relation of the distance to adjacent neighbors. Examples for these three kinds of distance matrix are shown in Figure 9, where (a) is showing an original U-Matrix. Based on another dataset, (b) is an interpolated U-Matrix and (c) shows the distances using the size of the neurons.

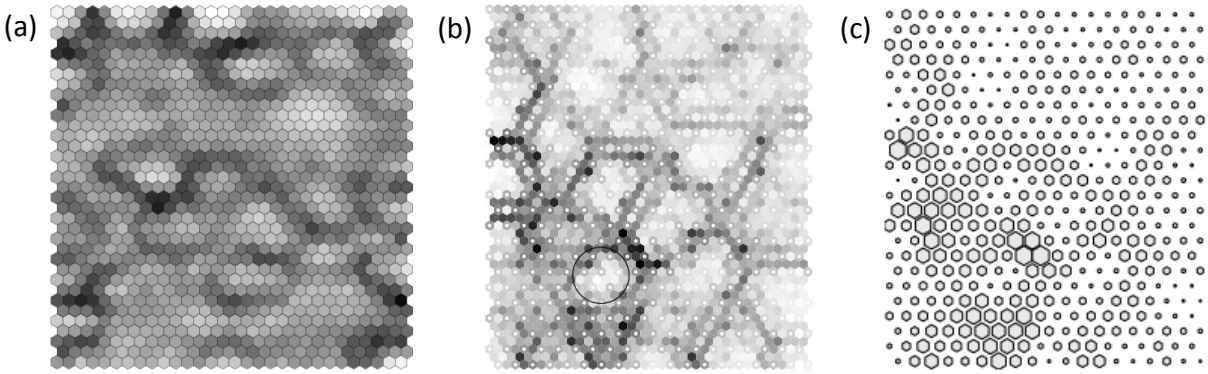


Figure 9: (a) U-Matrix without interpolated neurons, (b) U-Matrix with interpolated neurons, (c) distance matrix resizing the SOM neurons to their interneuron distances (Vesanto 1999).

P-Matrix. The P-Matrix was introduced by Ultsch (2003) as new visualization method for the ESOM tool (Databionic 2007). This visualization method measures the data's density structure using the pareto density estimation, which is a special case of the kernel density estimation with a fixed kernel bandwidth (Ultsch and Mörchen 2005). At each neuron position, a density estimation for the data space is displayed. Taking a closer view on figure 10 it seems that most but not all of the patterns of the P-Matrix are the inverse of the U-matrix (Ultsch 2003).

U*-Matrix. This method is a combination of the distance-based U-Matrix and the density- based P-Matrix described above. The U*-Matrix disregards local distances in dense regions where they do not matter inside a cluster, keeps the values in average density areas, and emphasizes sparse regions of the SOM data space (Ultsch and Mörchen 2005). This brings a much clearer outline of clusters compared to the U-Matrix. Figure 10 allows a comparison of the visualized results using the U-Matrix, P-Matrix and U*-Matrix for a given dataset.

The close relationship between the distance and the probability density of the SOM vectors is coming from a SOM characteristic. It is known as magnification factors (Pampalk et al. 2002) and expresses that areas with high density of vectors are described with more detail than sparse ones. The same principle is used for the visualization in the next paragraph.

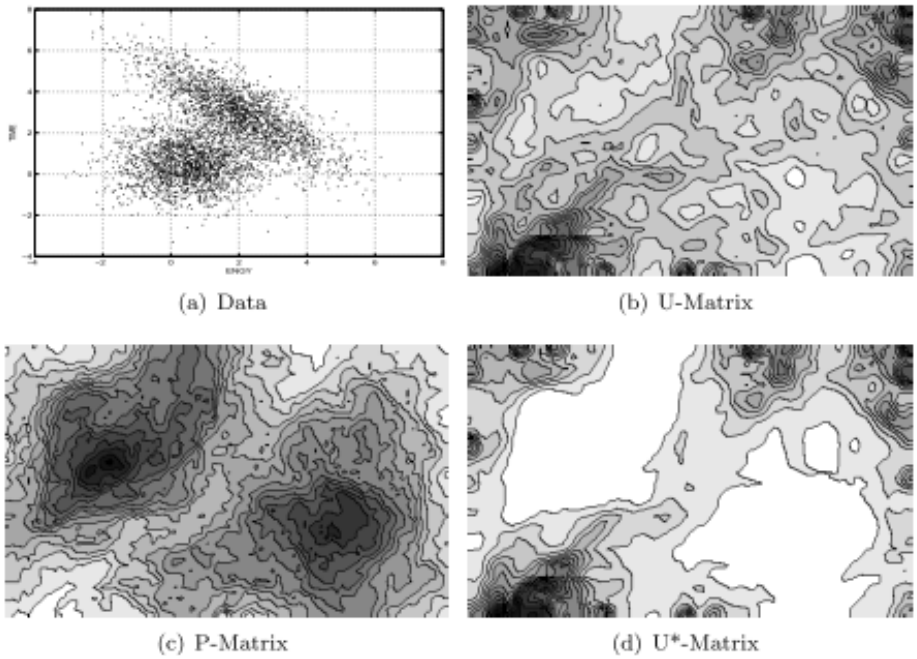


Figure 10: Comparing the visualization results from U-matrix (b), P-matrix (c) and U*-matrix (d) applied on the same dataset (a) to find cluster regions (Ultsch and Mörchen 2005).

Smoothed Data Histogram. Pampalk et al. 2002 developed this technique to parametrically visualize clusters in SOMs. The smoothed data histogram (SDH) method tries to find clusters through estimation of the probability density of the data on the map. The idea is that clusters are areas in the data space with a high density of data items. The smoothing parameter *s* allows changing the membership degree and thus affects the cluster building, visually explained in Figure 11. While a U-Matrix representation with large distances might represent lower distances and possible clusters less significant, the various cluster shapes of the SDH show a more precise hierarchical structure of the clusters in the data (Pampalk et al. 2002).

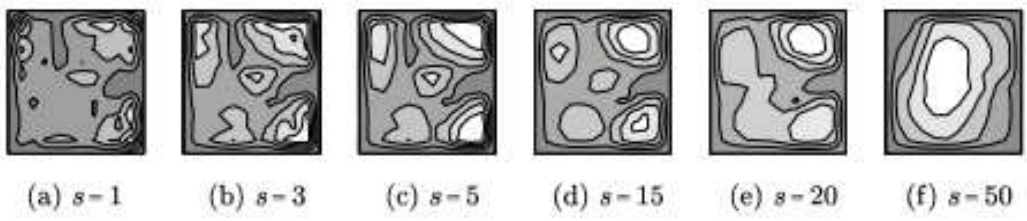


Figure 11: Effects on cluster detection by changing the value of the smoothing parameter *s* for the SDH (Pampalk et al. 2002).

2.2.2 Projections onto the SOM

The examination of new data with the map is used to see how the training data belongs to the SOM space. This is utilized to find similarities and correlations in the input data. Further, it is a popular method for novelty detection (Marsland 2003), where data samples other than the ones used for training are associated with the map. Match-accuracy is another important topic in SOM visualization. How it can be made visible on the map is described at the end of this subsection.

Data Histograms. One might want to know the distribution of the input data items on the SOM grid, respectively on the neurons. The visualization of hits per neuron can be done by simply doing a BMU search and mapping the input vectors onto the matching neurons. Euclidean distance is commonly used for measurement type. If there are multiple items on the same neuron, a data or hit histogram is obtained. The visualization can be done in several ways. Figure 12 shows three types of hit histograms on a rectangular and hexagonal grid topology. There is an interpolated heat map used in (a), (b) has proportional markers placed on the center of the neurons, and (c) colors the neurons in relation to their hit count in a unique color range. An issue with this representation, where simply the BMU is pointed out, is the fact that it shows no detail in the accuracy of the neuron match. Therefore, more enhanced visualizations, such as the sky-metaphor, were developed.

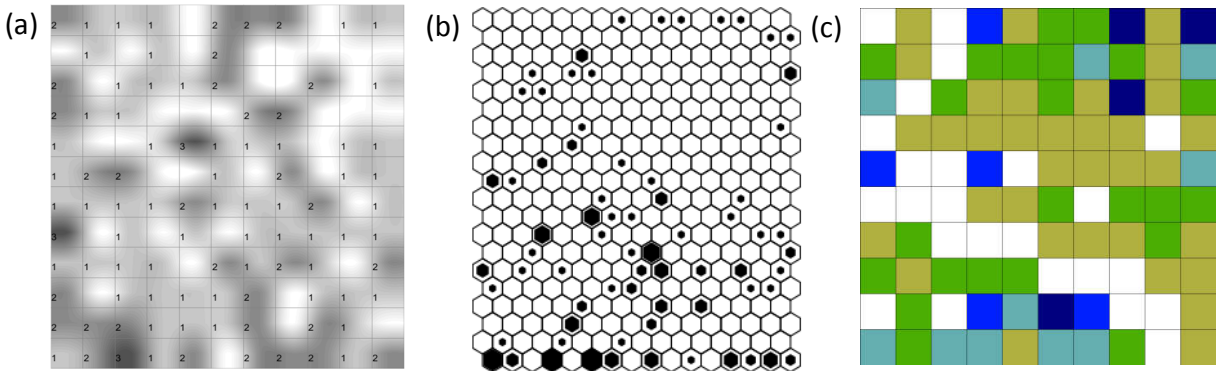


Figure 12: Types of data histograms, showing the distribution of hits per neuron with (a) interpolated density coloring, (b) markers, (c) color range (TU Vienna 2013, Vesanto 2002).

Sky-Metaphor. The sky-metaphor visualization (Latif and Mayer 2007) is similar to a hit histogram, with used techniques from the P-Matrix and the SDH (see subsection 2.2.1), but provides more detail about how the input data is distributed over the neurons, and especially how hits are scattered on a single neuron. In a hit histogram, the center of the neuron is used for placing the input vectors. The present approach shifts the input vectors towards the closest neighboring neuron,

based on their distance. This is resulting in a better visual differentiation of vectors on the same map unit as well as to in a better detection of similarities between vectors across neuron boundaries. For the creation of the night sky effect, neurons are colored in black and input vectors are mapped as white stars onto the map. Smoothed density histogram visualization at the top of the black neurons creates the galaxy effect. Further, interconnections can be automatically or manually be drawn as connected lines between the mapped input vectors. Figure 13(a) allows a closer look onto mapped vectors within the neurons, together with drawn interconnections. The whole map with four galaxies is shown in (b).

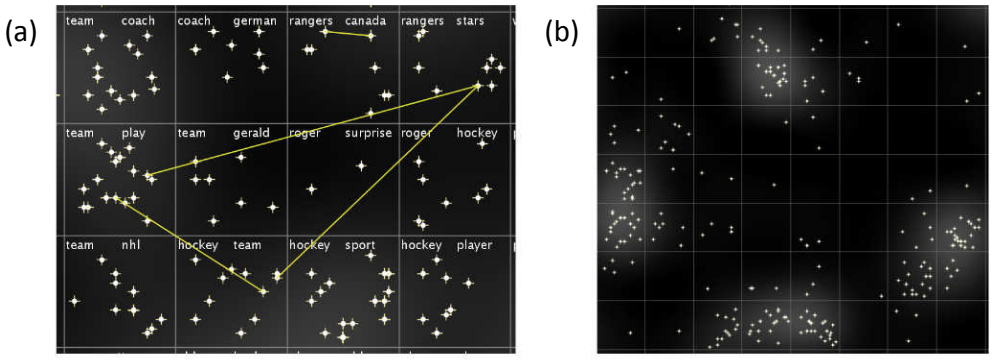


Figure 13: Sky-metaphor. (a) Detailed view of the map, with input vectors mapped as stars onto the neurons, some of them connect to trails. (b) The entire map with four galaxies (Latif and Mayer 2007).

Component Charts. A possibility of showing multiple vector components is generating charts and projecting them onto the map. Attributes from codebook vectors as well as from input vectors can be used for multivariate symbolization. This representation is limited to a small number of components. Popular chart types are pie or bar charts. Figure 14 shows both types, (a) applied on projected inputs vectors and (b) used as component bars describing SOM neurons. A more enhanced method which uses cross-symbolization for different data spaces is explained in subsection 3.1.3. Moreover, special types of glyphs are able to visualize the multi-dimensional SOM vectors, for example fan plots or Chernoff's face (Vesanto 2002).

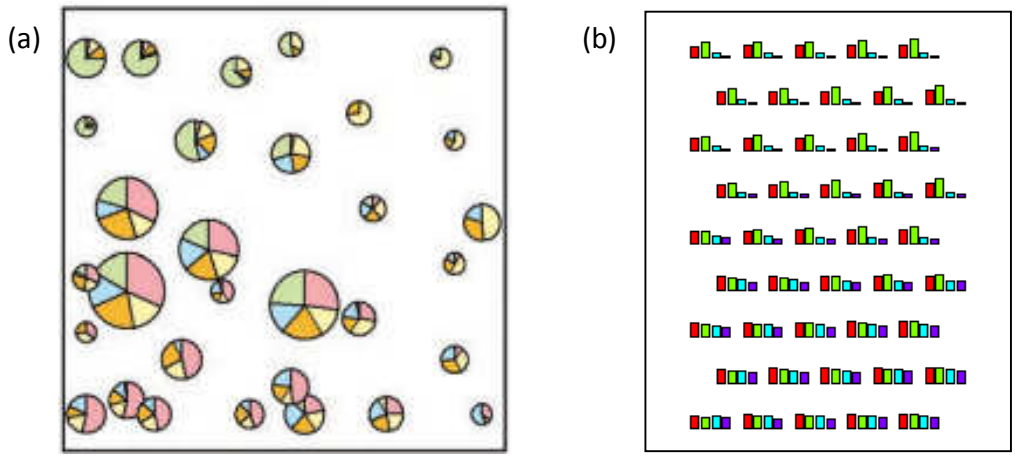


Figure 14: Component charts. Projecting (a) input vector attributes as pies charts and (b) codebook vector attributes as bar charts (Skupin and Fabrikant 2003, SOM Toolbox for MATLAB 2013).

Trajectories. If there is an interest in following a selected input vector on the map during the training process, trajectories can be created by drawing connected lines between the changing positions of the BMU. This visualization is also named projection of the multi-dimensional state space (Himberg et al. 2001). The trajectory is drawn according to the updated BMU position of the input vector during certain training iteration steps. Figure 15 shows such trajectories mapped onto a SOM, created from three input vectors. As labeled in the map, the state of each vector was recorded once every 10,000 iterations (Skupin and Agarwal 2008). The trajectory visualization approach can basically be used for many other purposes to show connections between input vectors (see sky-metaphor) or to show movement. More enhanced research work based on SOM trajectories is given in section 3.1.3.

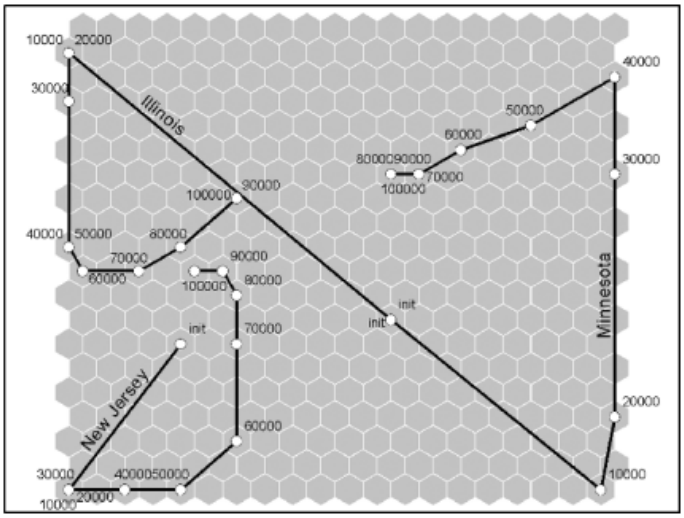


Figure 15: Visualizing the SOM training. Three input vectors are recorded every 10,000 training iterations and connected at each BMU position (Skupin and Agarwal 2008).

Metro Map. This approach utilizes the well-known metro map metaphor to project multiple component lines onto the SOM (Neumayer et al. 2007). A component line is drawn from the center of discretized areas in a component plane, depicted in Figure 16(a). Line connections are done from the lowest to the highest component value with mutable steps in between. As the main concept of metro maps, lines show a simplified, skewed representation of the underlying data. Other functions are the aggregation of highly correlated component lines together with line snapping and the representation of metro stations as markers scaled to the number of intersections. Hierarchical clustering is used for aggregation. Figure 16(b) illustrates a metro map on the SOM, showing four colored and snapped component lines. All in all, this visualization provides extensive calibrating and scaling possibilities for the chosen component planes.

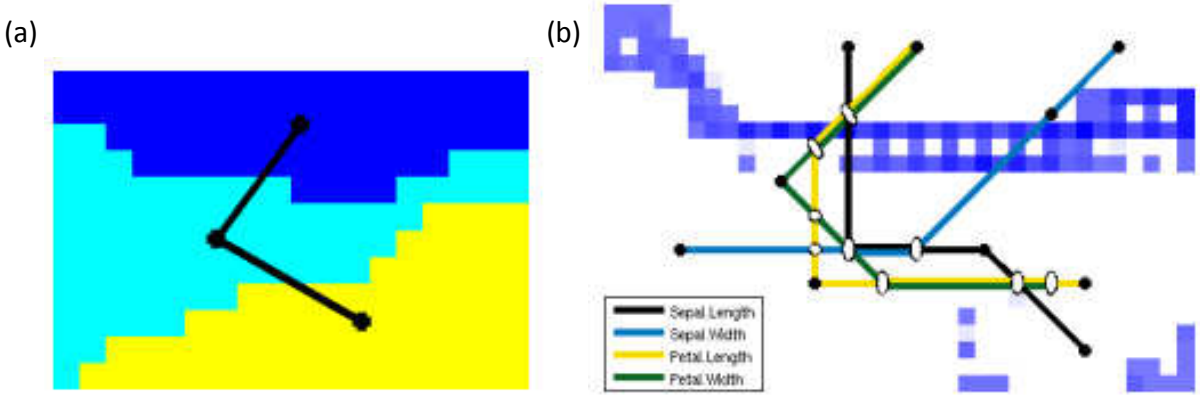


Figure 16: (a) Trajectory connecting the centroids of discretized areas of one component plane. (b) Metro map showing four component lines (Neumayer et al. 2007).

Neighborhood Graphs. Graph-based visualizations of the SOM input data can be used to determine the mapping topology and how relations are preserved after projection (Pölzlbauer et al. 2005). Clusters can be indicated which are close in input space, but moved further apart after training. Two methods are introduced. The first graph structure is generated by nearest neighbor calculations of the input data. The second one uses pair-wise distances between vectors in input space. The first method makes an efficient use of the space in large SOMs with a lower number of input vectors, whereby the second approach has its advantages in the detection of outliers and dense areas. Figure 17 shows the nearest neighborhood graph representation of input vectors projected onto a SOM with rectangular topology.

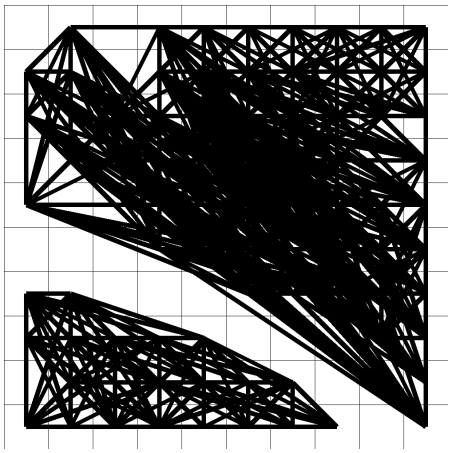


Figure 17: Neighborhood graph representation of the projected input data onto the SOM (Pölzlbauer 2005).

Minimum Spanning Tree. The minimum spanning tree (MTS) is a similarity relationship visualization method used to connect similar vectors, represented as nodes on the SOM (Mayer and Rauber 2010). Both codebook vectors as well as projected input vectors can be visualized. The edges are basically weights showing the distance between connected nodes. Figure 18 depicts a weighted MST representation for both the SOM space (a) and projected input space (b). The line thickness of the edges is related to their weight values. This approach provides visual interpretation of clusters, cluster connections and vector correlations. A related method is the neighborhood graph, described in an earlier part of this subsection.

(a) (b)

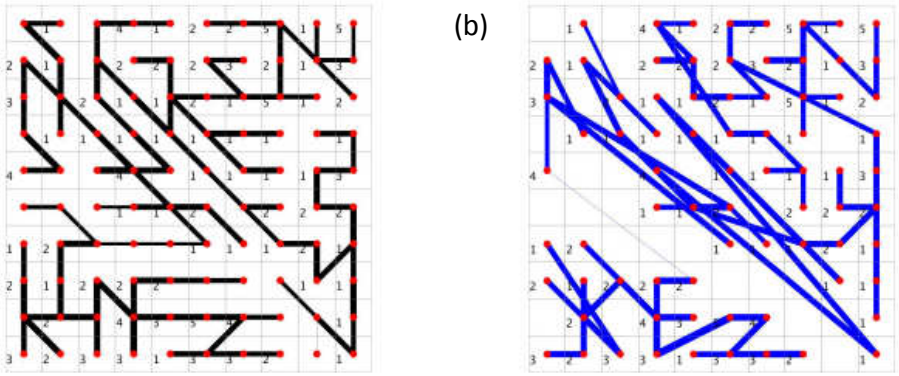


Figure 18: MST Visualization of (a) SOM codebook vectors and (b) input vectors. The edges are scaled according to their weight values (Mayer and Rauber 2010).

Class Map. If the input data is assigned to classes, this representation makes it possible to visualize clusters and similarities between distributed classes in the map. It uses a graph-based coloring where the detected regions are first subdivided into

Voronoi diagrams (Mayer et al. 2007). Then, imaginary lines are drawn, which connect regions containing same classes. Pixel-wise color-assignment along the connection lines and across borders is used to visualize classes that span over multiple regions. To create a smooth partitioning at the border transitions, the line segments are weighted at both ends. Figure 19 shows different abstraction levels of a class map, projected onto a SOM grid with rectangular topology. The granularity can be modified by setting a minimum class threshold value.

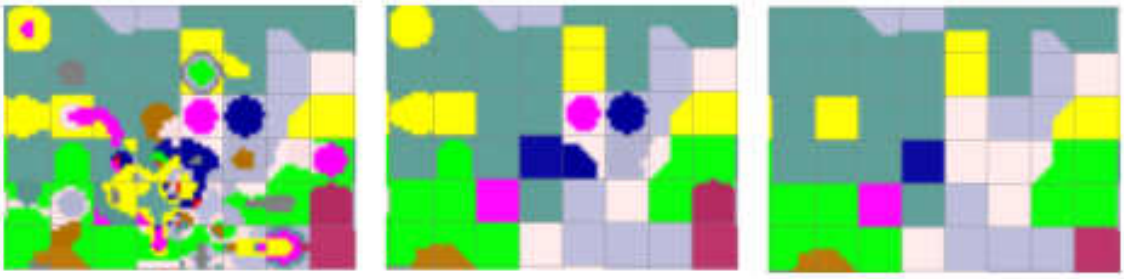


Figure 19: Levels of class granularity. The minimum class threshold was set to 0%, 50% and 100% contribution fraction (Mayer et al. 2007).

Response Surface. The BMU search has the purpose to find the best matching neuron, but usually there are other neurons which might be almost as good as the BMU. Additionally, a projection onto the BMU gives no detail about the hit accuracy. The accuracy issue was already visually tackled by the sky-metaphor method (see Figure 13). Response surfaces (Vesanto 1999) can solve both problems through highlighting potential matches on the map. This visualization shows the relative goodness of each neuron to a given input vector by coloring the neurons from black, which is the best, to white, corresponding to the worst possible match (see Figure 20). The quantization error is used as indicator for the matching goodness. The quantization error measures the distance between an input vector and its BMU.

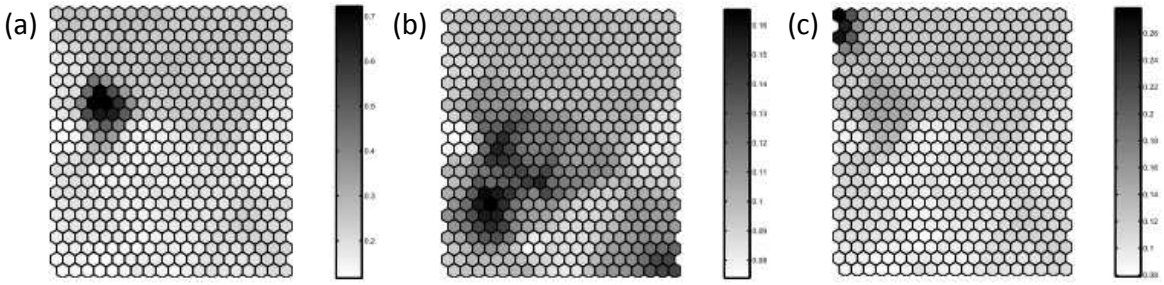


Figure 20: Response surfaces. (a) Good match, (b) poor match, (c) average match. Black color associates the best response and white signalizes the worst (Vesanto 1999).

There are two ways to determine the accuracy: First, by calculating the average quantization error, and second (used for independent data measures) by getting the average distance of each neuron to its neighbors. Finally, the quantization errors need to be scaled with the accuracy. As depicted in Figure 20, a good match can be recognized as a clear cluster, highlighted as dense black spot. An average match may be highlighted as cluster which is cut along the border of the SOM. Maps with scattered dark cluster structures indicate that there is a poor match of the input vector, because a large number other potential BMUs exist.

Position Accuracy Marker. Other than the response surface, which uses the whole map to show match accuracy for one input vector, position accuracy markers indicate match accuracy for multiple vectors by rescaling the size of each sample marker. The position of the marker shows the BMU and the size the quantization error (Vesanto 1999). Figure 21 shows an example for calculated accuracy markers. The smaller the diameter of a circle, the better is the BMU accuracy.

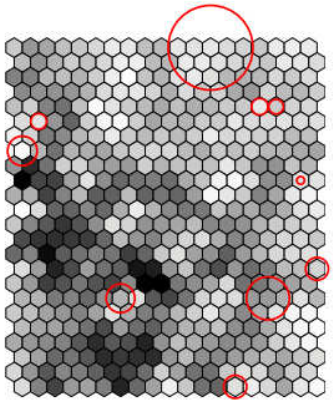


Figure 21: Position accuracy markers placed on top of a distance matrix (Vesanto 1999).

2.2.3 Projections from the SOM

Sometimes it can be useful to project the data from SOM output into another space to get a better insight into distances and correlations between neurons or to visualize the distribution of vector attributes. Among other possibilities, two common methods are presented here. The first approach projects the units from SOM space into a distance-preserving low-dimensional space. The second method generates a parallel coordinate plot from codebook vector attributes and clusters. Both types require color-coding to be able to track neurons and clusters over space.

Distance-Preserving Visualizations. This technique is commonly used when, for some interest, the contraction and expansion effects of the SOM training need to be visualized. As already mentioned, the topology-preserving SOM grid doesn't keep

interneuron distances. A distance-preserving projection of neurons using Sammon's mapping and PCA is shown in Figure 22. In (a), smooth topology-based coloring of the grid is done, the neurons are then projected into the 3D space. The distribution of neurons within clusters can be seen in (b), using a two-dimensional plane. More information about these two projection methods and related improvements of the SOM algorithm is given in subsection 3.3.1. Color-coding is the best way to recognize the nodes after projection into another space. Distance-preserving projections also allow a data space distance visualization of the SOM (Himberg 1998). An example can be seen in Figure 23, where in (a) the SOM neurons are projected using Sammon's mapping. Then, based on interneuron distances in this projection, the coloring of the SOM is adjusted in (b). Another purpose of combining a SOM with such projections is to find out correlations between pairs of vectors. An accurate comparison of values in component planes is almost impossible (Vesanto et al. 1998). Two-dimensional functional plots allow a more detailed comparison between two components. SOM can be used to reduce noise in the data, then as further step the component vectors are laid out in the function plot to depict their related distances.

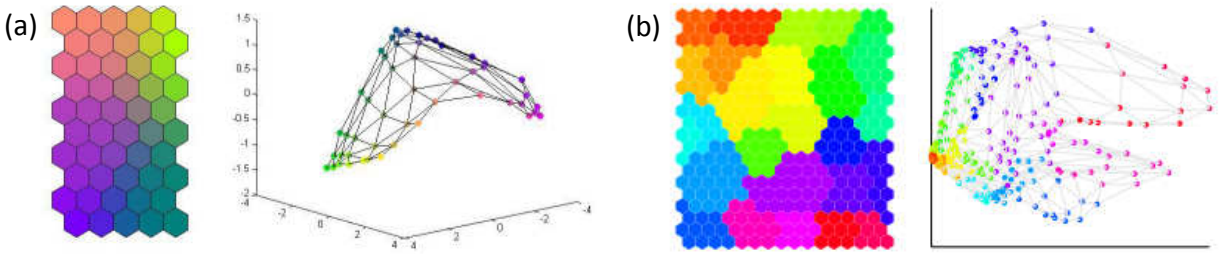


Figure 22: Distance-preserving projecting of SOM neurons (a) into three-dimensional space to see interneural distances, or (b) into two-dimensional space to see the distribution within clusters (Gorricha 2009, Vesanto 2002).

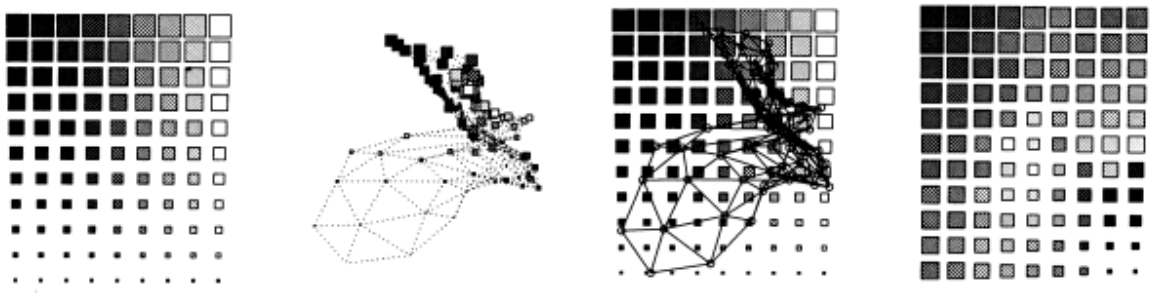


Figure 23: Data space distance visualization of SOM neurons using projection and back projection (Himberg 1998).

Parallel Coordinate Plot. The parallel coordinate plot (PCP) is a popular method for the visualization of multivariate patterns (Guo et al. 2005, 2006). In combination with SOM, this tool provides a powerful data exploration technique. SOM clusters, or data vectors, are projected into the PCP and linked with the corresponding colors in the map. The PCP uses nested-means scaling, which divides each axis into equal-length portions, where the mean value is always in the center (see Figure 24). This method makes diverse vectors and value ranges comparable. The lines can be scaled in relation to the cluster size, or as another possibility, to the variance within each cluster (Guo et al. 2005). Therefore, a PCP provides great insight into the characteristics of each cluster. An interactive highlighting at the SOM neuron or cluster level with corresponding lines in the plot can improve its usability.

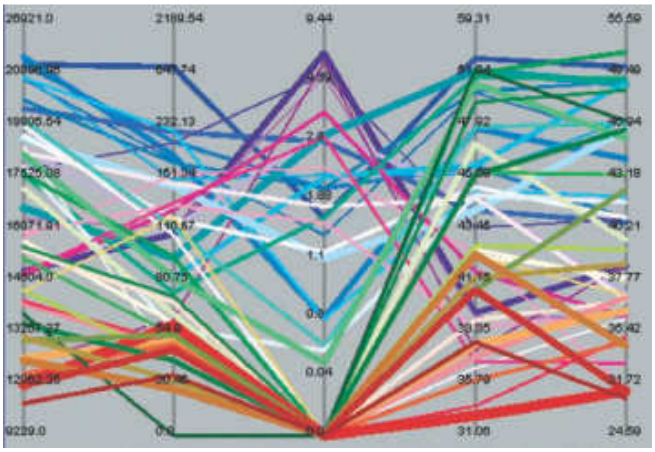


Figure 24: Parallel Coordinate Plot showing SOM clusters.
The line thickness is scaled to the cluster size (Guo et al. 2005).

Remark: The mentioned techniques for projecting data from the SOM can of course be applied to input data as well, which makes sense to explore the original input space.

2.2.4 Visualizations linked from the SOM

Linking SOM Data to Geographic Maps. A method for linking geo-referenced SOM data back into the geographic space was presented by Skupin and Argwal (2008). Gorricha and Lobo (2012) used the same methodology for referencing the geographic map features with label colors obtained from 3D SOM instead of a two-dimensional one. No matter if a 2D or 3D SOM is used, the results follow the same principle. Figure 25 shows a geographic map, colored in relation to the corresponding BMU of the neurons in a high-resolution SOM. This method of linking different data projections allows an interactive and efficient way to recognize and associate areas in the map. This combined representation provides valuable

conclusions for geospatial data analysis as the result from attribute space is transferred to an actual real-world location.

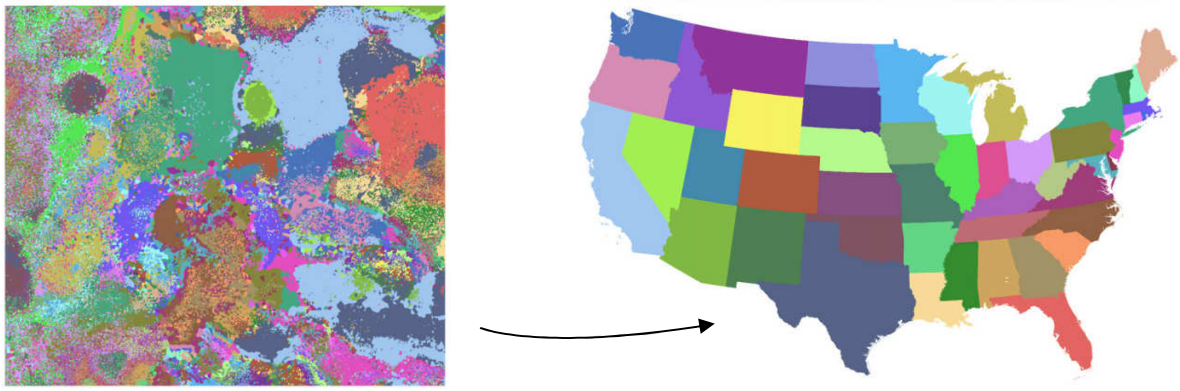


Figure 25: Linking the BMU colors from SOM space to their corresponding geographic map features (Skupin and Esperbé 2011).

3. Literature Review

There is a lot of related research done within a broader context of this thesis work, reaching from sophisticated visualizations to intelligent color-coding algorithms. GIS is heavily employed to the related research work in the area of SOM visualizations presented throughout this chapter. Further, k-Means and hierarchical clustering of SOM space is discussed. Finally, other interesting SOM algorithms are given, which have an effect on the visualization as well.

3.1 Related Work

Spanning from common representations to more enhanced methodologies, the next five subsections show impressive SOM visualizations and coloring results.

3.1.1 Spatialization

During the research on Self-Organizing Maps an expression called spatialization appeared. Spatialization is a comprehensive term which refers to spatial metaphors which are used to describe an abstract concept. These metaphors include basic geographic concepts, such as location, distance, pattern, or scale. Spatialization is

defined as the methodology for knowledge construction by applying dimensionality reduction and spatial layout on large, multi-dimensional datasets (Skupin and Fabrikant 2003). Spatio-temporal techniques developed and applied in GIS are also applicable in spatialization. The data can be geo-referenced or not, and it does not matter if it is unstructured, semi-structured or structured. In other words, it is possible to visualize and map almost every domain of interest. In case of non geo-referenced sources, spatialization works on implicit relationships derived from quantifiable notions of distance and similarity (Skupin and Fabrikant 2007). The SOM is a central element in such a spatialization procedure. But how does it work? According to Skupin and Fabrikant (2007), there is no single method to apply spatialization. Reasons therefore are the mostly very inhomogeneous data and the different disciplines for which the workflows are developed. Figure 26 shows the spatialization procedure for the visualization of abstracts presented at the annual meeting of the Association of American Geographers (AAG).

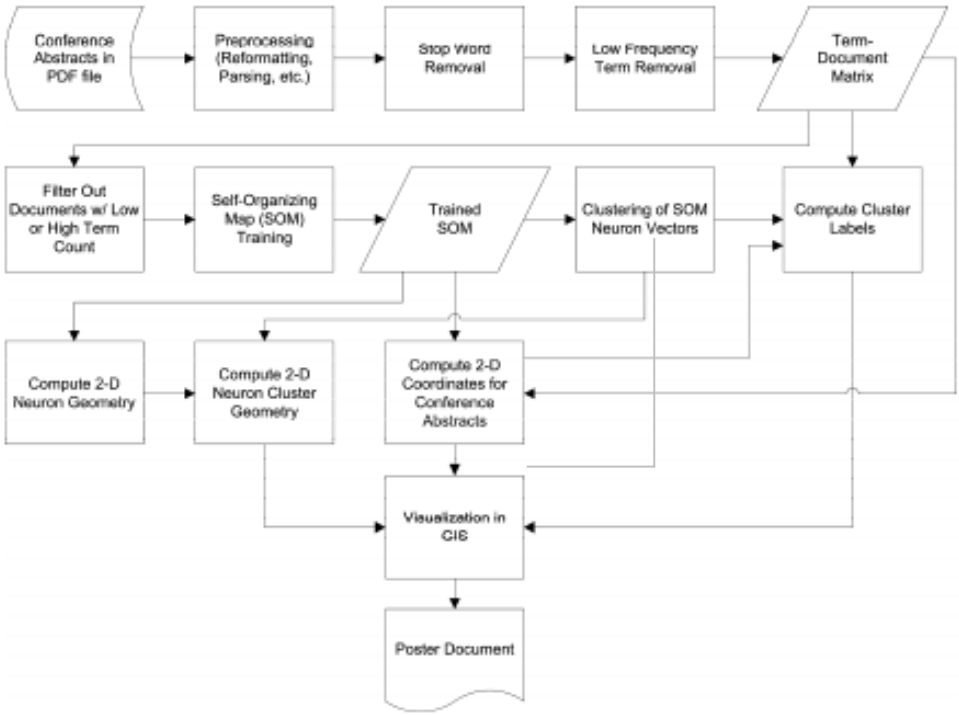


Figure 26: Spatialization process used to visualize AAG conference abstracts (Skupin and Fabrikant 2007).

Many different methods are used to create the final spatialized map. There is an extensive preprocessing task to get the abstract data into the right format. Then, the most frequent terms are extracted and a term-document matrix is created. The SOM algorithm is used for dimensionality reduction and spatial layout. Hierarchical clustering generates the boundaries. GIS software is used to transform the spatialized geometry and other elements. The scale dependence and symbolization

relies on cartographic principles. The result of this workflow is a map with a geographic look where the most occurring terms are spatially grouped and aligned according to their mutual relationship. The scaling of the labels shows their magnitude and clusters can be identified using different border styles (see Figure 27).

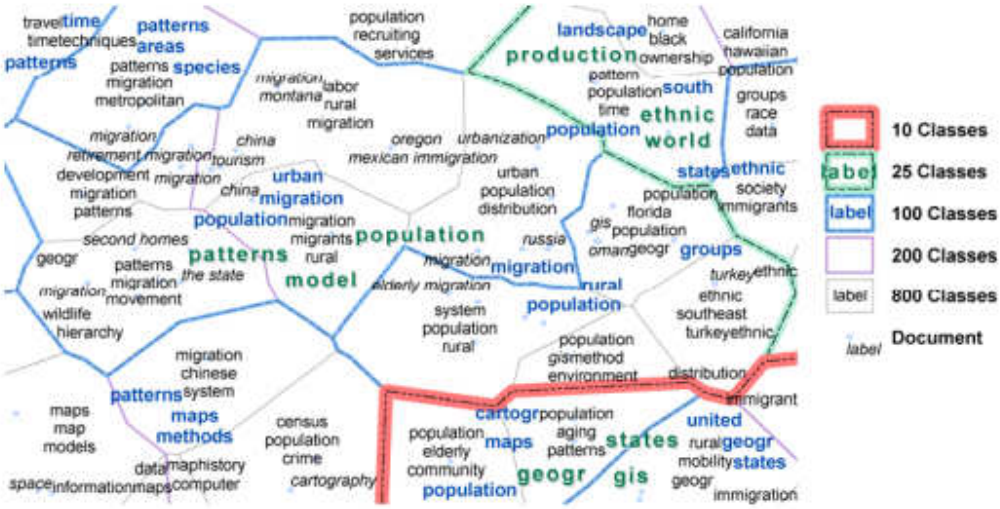


Figure 27: Extracted part of the resulting map from the AAG spatialization procedure showing five levels of hierarchical clustering (Skupin and Fabrikant 2007).

3.1.2 Novel Projections and High Resolution SOM

As already seen in the previous subsection, GIS provide powerful tools for creating sophisticated multidisciplinary SOM visualizations. Besides the described knowledge visualization approach, there are other research works which use SOM to create new map projections and visualizations by applying typical GIS techniques onto input and output data.

Skupin (2003) introduced a novel map projection where he trained the grid node of a SOM with geographic coordinates expressed in latitude and longitude instead of using high-dimensional feature vectors. Regular sampled coordinates, derived from a Shapefile, were used as input data source. This Shapefile had an equal-area projection and consisted of land mass locations on the entire earth, forming its continents with 14,500 points. After training the SOM, using Euclidean distance measure and a 2:1 neuron grid shape ratio to be able to compare the output to the initial projection, the result was then postprocessed and visualized with GIS software. The result of the SOM-based visualization in attribute space is depicted in Figure 28. In this representation, several layers such as border lines and country outlines, as well as the latitude-longitude grid, were projected onto the SOM with best matching unit search.

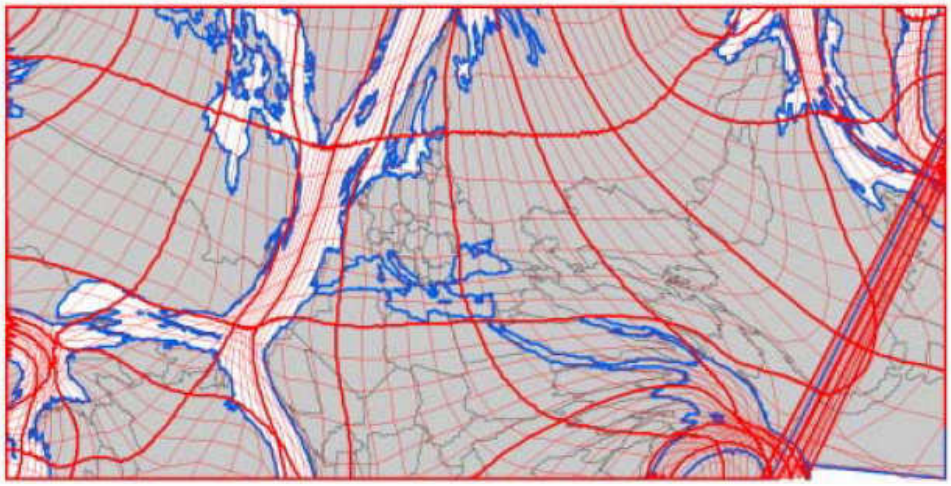


Figure 28: SOM-based visualization of the earth with projected cartographic input layers and trained with geographic coordinates using Euclidean distance measure (Skupin 2003).

Continents are filling the space of the oceans, because they were left out for training. Distortions come from the fact, that a Self-Organizing Map does not preserve relative distances. Especially on the edges, where most of the input vectors are laid out, a useful determination of recognition of areas is not possible. Also spherical distance was used for comparison, which did not lead to any meaningful results.

Another research work of Skupin and Esperbé (2011) is interesting in this context, which creates a holistic representation of the United States using 200,000 census block groups containing data from 6 different input types, such as population, climate, soil, or topography. These were visualized on a SOM consisting of 250,000 neurons. Data integration of all inhomogeneous sources was extremely difficult considering that both continuous and discrete data was used and various extents and granularities had to be unified. In the end, they had 69 attributes describing each geographic feature. The resulting high resolution SOM was then used for various visualizations besides the traditional component planes and U-Matrix representations. Through clustering the neurons with k-Means and SOM postprocessing using GIS software, an interesting map could have been realized. This map, shown in Figure 29(b), has the boundaries of 25 k-Means clusters as overlay on a density landscape, created from the neuron vectors. Blue colors can be anticipated with lower elevation and indicate low density, where brownish colors represent higher elevations and high density.

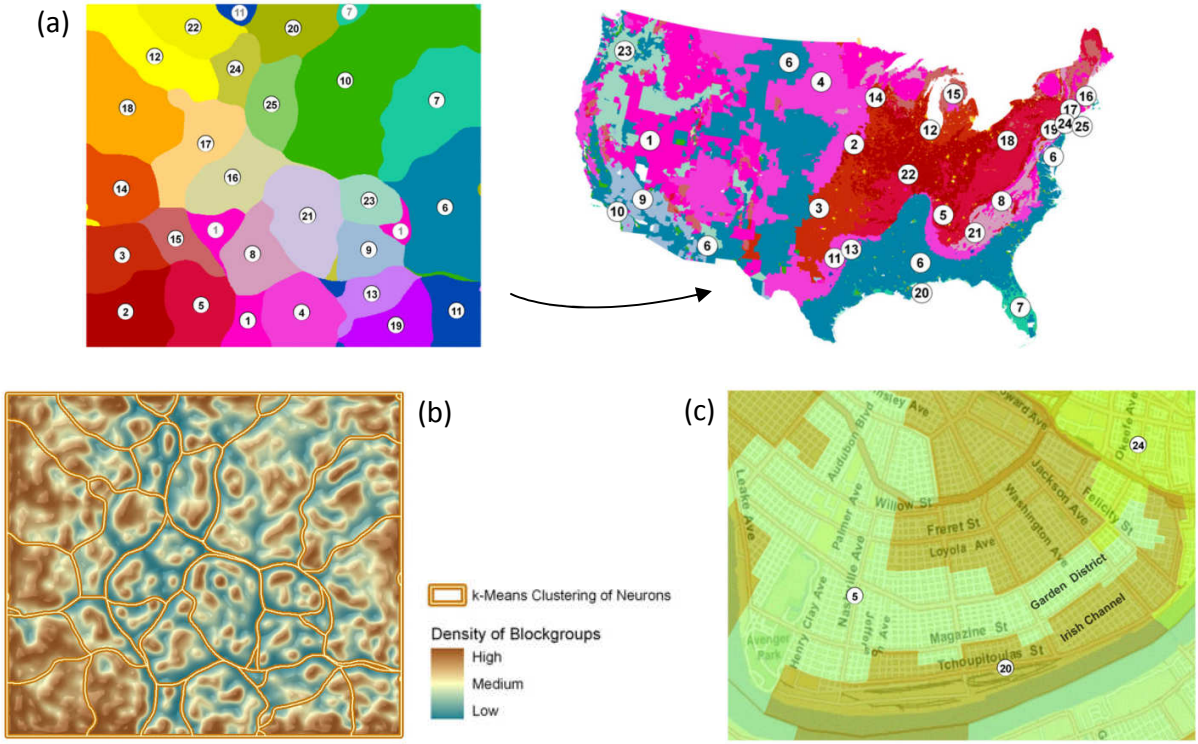


Figure 29: (a) Linked representation of the color-coded k-Means clusters in SOM and geographic space. (b) Cluster boundaries as line feature overlay on a neuron vector density landscape. (c) Cluster areas in the zoomed geographic space (Skupin and Esperbé 2011).

Another advantage of such a high-resolution SOM with multivariate attributes can be seen in Figure 29 (a) where the clustered neurons are projected from attribute into geographic space. This geographic regionalization can then be analyzed in detail. For example, if one zooms into specific regions on the map, as depicted in Figure 29 (c), block group clusters are represented in finer granularity. Such a side-by-side analysis of visualizations in different spaces and various zoom levels provides a valuable insight into multidisciplinary high-dimensional data.

3.1.3 Cross-Symbolization and Travelling in Attribute Space

Besides simply mapping input vectors as points onto the SOM, there has been the method of displaying attributes as glyphs onto the neurons (Vesanto 2002). Recently, another way to show multiple attributes describing these units using cross symbolization was introduced by Burns and Skupin 2009. Pie charts or bar charts are utilized to visualize multiple dimensions simultaneously. After determining the best matching units, the symbology is assigned to the mapped points on the SOM. The placement can be relative to one attribute space and its symbology is then derived from another attribute space. This requires having two separately trained SOMs. Figure 30 illustrates two kinds of cross-symbolized visualizations. The left one shows pie charts which are geometrically ordered by their population attributes.

The symbology shows nine utterance terms for each matching unit, the size is also derived from a chosen attribute. The SOM visualization on the right uses the same principle with bar charts, showing six utterance terms in another attribute space.

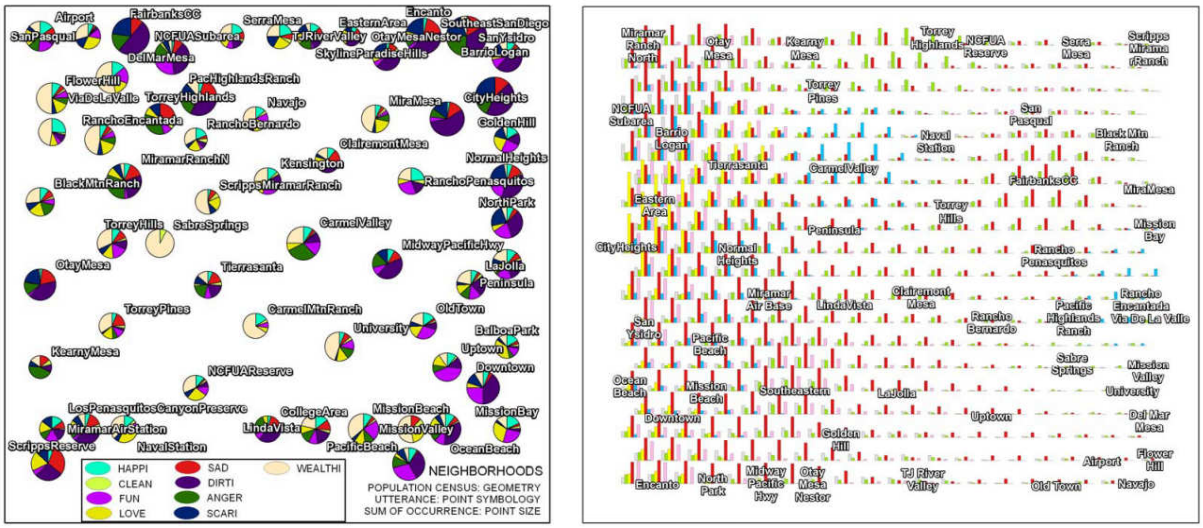


Figure 30: Cross-symbolization, showing multiple dimensions and attributes spaces simultaneously (Burns and Skupin 2009).

Travelling in attribute space is a fascinating idea of how changes in time and space can be visualized using SOM. In this approach, multi-temporal observations are linked to a spatialized representation in high-dimensional SOM space (Skupin and Hagelman 2005). It uses a neuron grid with a vast number of neurons, having enough space to project the input vectors without too much overlap. The demographic data used for training and projection is stored in a multi-year database. GIS is used for the whole visualization process, where each geometric data layer is stored as separate feature class. A trajectory is drawn as a directed and non-branching graph connecting time stamps represented as vectors in attribute space. Figure 31 shows two highlighted pairs of linked temporal vertices which indicate parallel development based on their multi-dimensional attributes.

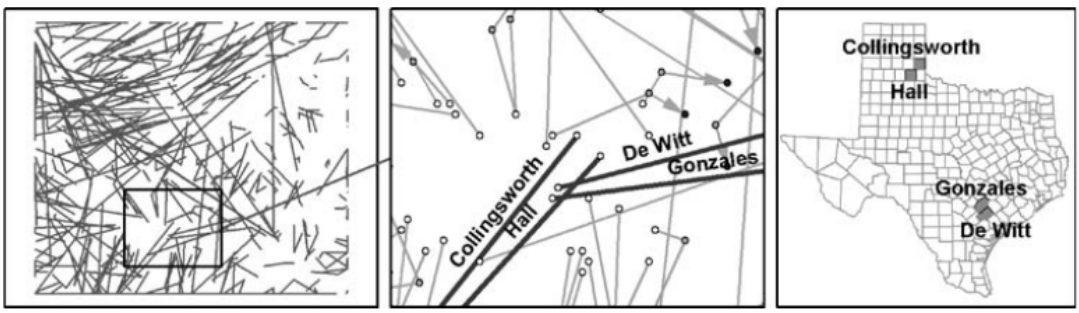


Figure 31: Multi-temporal trajectories showing parallel development of two pairs of cities in Texas (Skupin and Hagelman 2005).

An advantage of such trajectories is the capability of getting a quick and explicit visual representation of spatio-temporal changes and relationships through linking to other attributes. A criterion is the placement of the temporal vertices onto the neurons where one has to consider if the center, a random placement inside the neuron, or another method is used to get suitable results. Additionally, Skupin and Hagelman (2005) also describe a method for trajectory clustering and insertion of additional time vertices. Visualization is not limited to demographic changes; other spatio-temporal phenomena such as tornado touchdowns and hurricane paths can be mapped onto climate driven spatializations (Skupin and Esperbé 2008). The tri-space approach, lately reintroduced by Skupin (2010), uses the SOM technique with mapped data trajectories as one part of its multi-space transformation and analysis.

3.1.4 Adding a Third Dimension

The present thesis work does not apply SOM visualizations in 3D. Therefore, an excerpt to SOM representations in the third dimension is shown in this subsection. To enhance the visualization capabilities and detect clusters by observing the overall shape of the SOM, Vesanto et al. (1998) showed an example using Sammon's projection (Sammon 1969) representing the data in three dimensions.

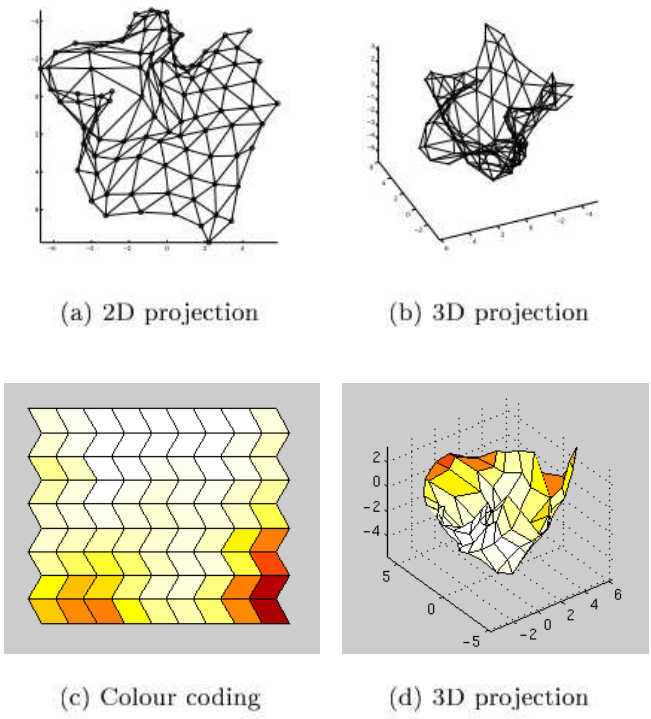


Figure 32: Mesh representation of SOM data in (a) 2D and (b) 3D, and (c) color coded component planes projected into (d) 3D (Vesanto et al. 1998).

Figure 32 shows juxtapositions where in the first approach, the neighboring units were connected with lines which resulted in a mesh presentation. The second approach was done with color-coded component planes instead of connecting the vector units. When projecting into the third dimension, the 2D projection has folded and makes a two-dimensional representation useless for correct information extraction. Where in the first approach, some effort is needed to find certain areas in the mesh, the color-coded version makes it easier to deduce from the two-dimensional SOM grid.
For the projection from two into three dimensions, typically distance or density values are taken. Takatsuka (2001) for instance used the distances from the distance matrix to visualize a SOM grid in 3D (see Figure 33). The 3D SOM visualization figure was created with GeoVista Studio software (Takatsuka and Gahegan 2002). When using the height, the distances no longer need to be normalized. Thus it is easier to find and interpret clusters in the three dimensional view. The visual recognition of clusters and boundaries can be enhanced by applying common cartographic colors to the surface. Interactive rotation functions make this to an intuitive exploration tool. From the point of visual perception and cognition, the third dimension it is the largest that human beings can easily grasp (Vesanto et al. 1998). The conclusion from SOM projections in 3D is that whenever possible and suitable, a three-dimensional projection can be used for further data exploration.

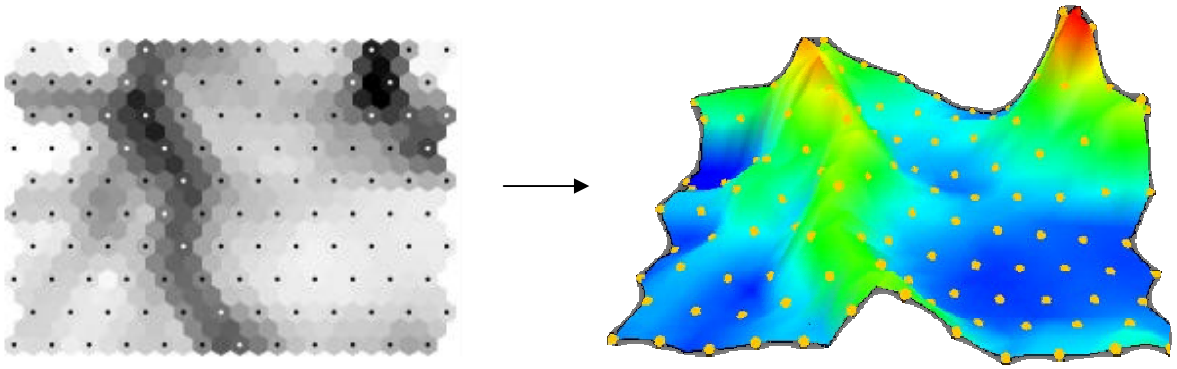


Figure 33: Creating a 3D SOM distance matrix from its equivalent 2D representation using the distances as height values (Takatsuka 2001).

Other three-dimensional SOM approaches are mentioned in the next two sections of this chapter as well.

3.1.5 Coloring the SOM Space

Indicating metrical relationships through spatial positioning is one advantage of SOMs. Coloring its neurons is the other way to achieve similarity encoding. The color-coding could be simply done by manually or randomly assigning colors or color ranges to the neurons and clusters, but this is not very intuitive. Therefore, different methods for SOM color-coding and color projections have been elaborated. Used color spaces are RGB (Red Green Blue), HSB (Hue Saturation Brightness) and CIELab. See references (Joblove and Greenberg 1978, CIE 1986, Wikipedia 2013) for further information about each color space. Basically, the intention is to get a perceptual difference in the neurons of the SOM space by approximating distances in suitably defined color spaces (Kaski et al. 2000). The coloring methods are as follows:

1) Color-coding based on the topological order of the neurons in the SOM

This type assigns colors to each SOM unit according to its position in the SOM space, without respect to relative distances between the neurons. One way is to generate a color plane and stretch it over the SOM grid. The plane can have four base colors at the edges and every transition color in between is calculated. Basically, a small number of colors results in discrete coloring. A large number of colors together with higher SOM dimensions results in smooth coloring of the grid. Himberg (2001) used the simplest form of this technique with four levels of gray for coloring equal quadrants of the SOM. The other way is to create a color plane from RGB space and let each neuron of the SOM grid pick its associated color in the plane (Himberg 1999). Figure 34 shows an example of a smoothly colored SOM grid following it topology.

Figure 34: Coloring of SOM neurons based on their topological order in the grid (Vesanto 1999).

Guo et al. (2005) have developed a very complex and well-thought technique that creates 2D color schemes from the 3D CIELab space. This method constructs a two-

dimensional array of differentiable, logically ordered colors using variations in hue and lightness. Therefore, a square grid is laid onto the CIELab color plane and lifted to the shape of a certain geometric object, like a bell or ellipsoid. The elevation of the intersection where the grid meets the surface of the geometric object defines the lightness. The coordinates of the grid on the color plane results in the associated hue (Guo et al. 2005). Figure 35 illustrates this method creating a 5x5 diverging-diverging color scheme based on an ellipsoid model. This approach is utilized in the SOMVis tool (Guo 2013).

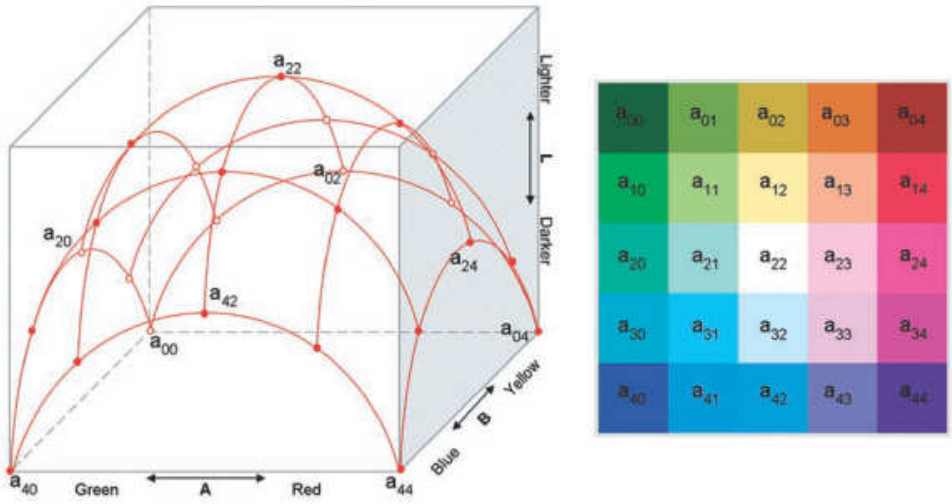


Figure 35: Diverging-diverging color scheme created from an ellipsoid model in the CIELab space.

2) Similarity-based coloring using distance-preserving projections

These methods use different non-linear projections of the codebook vectors for assigning colors to the SOM neurons based on their mutual distances. One approach uses the popular Sammon's mapping (Himberg 1999) together with the RGB color space. Another more sophisticated projection method was developed by Kaski et al. (2000) which transforms into CIELab color space. Both methods let the neurons pick their associated color based on their location in the projected color space. The CIELab colors require a calculated hue value, non-saturated colors where left out and lightness got a fixed value. Finally, the determined colors are linked back to the SOM neurons (see Figure 22 as an example). Besides RGB and CIELab space, there is the possibility to use the HSB circle and detect colors based on calculated hue values (Vesanto 2002); saturation and brightness have fixed values in this technique too. Figure 36 applies the latter of the mentioned coloring methods to the SOM neurons.

Figure 36: Similarity coloring of a SOM based on interneural distances (Vesanto 1999).

3) Similarity-based coloring through a one-dimensional color SOM

The SOM itself can be used to order the neurons in a diverging manner, as described by Vesanto (2002). First, a one-dimensional SOM is trained from the codebook vectors. Then, colors from the hue circle in HSB space are calculated for each neuron in the 1D SOM. Colors can be detected equidistant from each other or relative to the distance of neighboring vectors. As last step, each neuron in the SOM grid gets the color of its BMU in the color SOM. This method is used for this thesis implementation. Further descriptions can be found in subsection 4.5.1.

3.2 K-Means, Hierarchical, and Geo-Clustering

Clustering has been mentioned many times before this section. Visual representation of clustering can be gained from SOM visualization methods such as the U-Matrix. The next pages give some more explanation about the two most common clustering techniques applied to SOM, namely k-Means or hierarchical clustering. Vesanto (2002) provides the definition of clustering as: "Clustering algorithms divide, or partition, data into natural groups of objects. The term natural usually means that the objects in a cluster should be internally similar to each other, but differ significantly from the objects in the other clusters."
While the k-Means algorithm is often directly compared to SOM as a related technique in multi-dimensional space (Bacao et al. 2005), it can also be a useful method in combination with Self-Organizing Maps. As already described in the previous sections, k-Means is applied to SOM by showing clusters through color coding and cluster boundaries as overlay. The algorithm divides objects into k clusters where each objects depends to the clusters with the nearest mean. It starts with the initialization of the k number of means, also called seeds, which are randomly created within the data space. Then, all objects are associated with their nearest centroid by calculating the Euclidean distance from each object to each of

the k means. The partitions then represent Voronoi diagrams. After the first clustering, the centroid of each of the clusters becomes the new mean. This iterative process is then repeated until convergence. The result can be seen in Figure 37. There is an improved algorithm, called k-Means++, which uses a more sophisticated seeding of the initial k means. Arthur and Vassilvitskii (2007) describe the procedure to initialize the centroids before proceeding with the standard k-means iteration process.

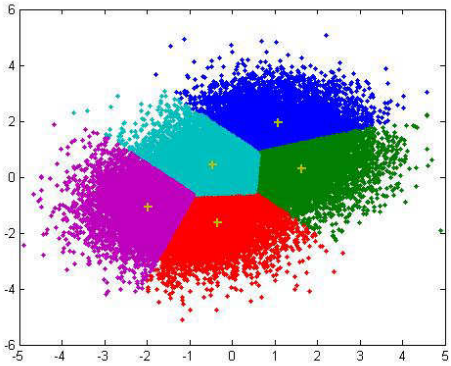


Figure 37: K-Means result, showing 5 Voronoi cluster cells (k=5) and their centroids (image source: http://www.mathworks.com/MATLABcentral/fx_files/19344/1/k_means.jpg).

Another technique besides k-Means partitioning is hierarchical clustering. Generating cluster hierarchies allows exploring the vector space from different granularities. The representation of such clustering is done in a tree diagram, a so called dendrogram. There are two types, namely the bottom-up, or agglomerative, approach as well as the top-down, or divisive, approach. In the agglomerative approach, each object has its own cluster in the beginning and those clusters are merged when moving up the hierarchy. The divisive approach starts having all objects in one cluster and recursively splitting up in smaller ones when moving down the hierarchy. Clustering algorithms are based on distance measures. The two general distance measures used are within-cluster distances and between-cluster distances (Vesanto 2002). Within-cluster distances measure the spreading in the cluster and between-cluster distances, also called linkage criteria, determine the separation between clusters. The corresponding formulas are given in Table 2.

Within-cluster distance $S(C_k)$	
average	$S_a = \frac{\sum_{i,i'} \|\mathbf{x}_i - \mathbf{x}_{i'}\|}{N_k(N_k-1)}$
nearest neighbor	$S_{nn} = \frac{\sum_i \min_{i'}\{\|\mathbf{x}_i - \mathbf{x}_{i'}\|\}}{N_k}$
centroid	$S_c = \frac{\sum_i \|\mathbf{x}_i - \mathbf{c}_k\|}{N_k}$
variance	$S_v = \sum_i \|\mathbf{x}_i - \mathbf{c}_k\|^2$
Between-clusters distance $d(C_k, C_l)$	
single linkage	$d_s = \min_{i,j}\{\|\mathbf{x}_i - \mathbf{x}_j\|\}$
complete linkage	$d_{co} = \max_{i,j}\{\|\mathbf{x}_i - \mathbf{x}_j\|\}$
average linkage	$d_a = \frac{\sum_{i,j} \|\mathbf{x}_i - \mathbf{x}_j\|}{N_k N_l}$
centroid	$d_{ce} = \|\mathbf{c}_k - \mathbf{c}_l\|$
Ward	$d_w = \frac{N_k N_l \|\mathbf{c}_k - \mathbf{c}_l\|}{N_k + N_l}$

Table 2: Clustering distances (Vesanto 2002)

The agglomerative hierarchical clustering algorithm can be explained in four main steps: 1) assign each object to its own cluster, 2) calculate the distances between all clusters based the chosen linkage criteria, and 3) merge the two clusters which are closest to each other. Finally, repeat the whole process beginning at step two until only one cluster is left. The top-down approach goes into the other direction. Agglomerative algorithms produce binary trees which have at least one extra intermediate cluster which need to be pruned out (Vesanto 2002). Figure 38 shows a hierarchical clustering result where the SOM is colored according to the calculated base cluster hierarchy from the dendrogram. The colors of superordinate-clusters are simply calculated as averages from their subordinate-clusters.

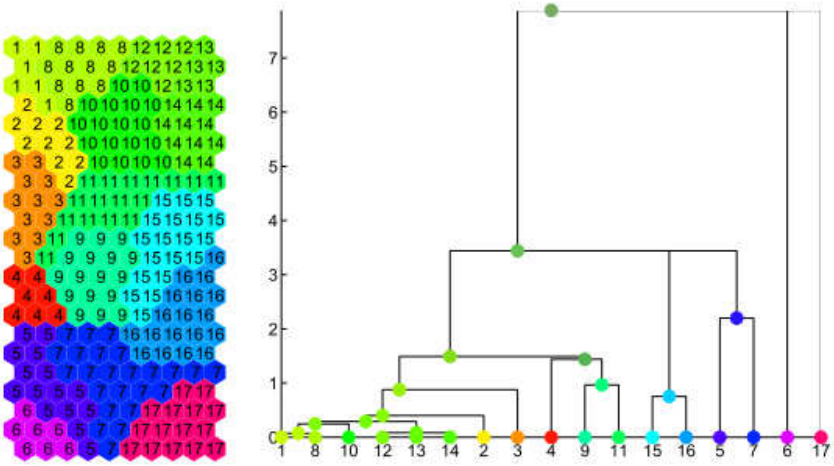


Figure 38: Hierarchical clustering of a Self-Organizing Map (Vesanto 2002).

There has also been a big effort in finding improvements for SOM clustering. Gorricha and Lobo (2012) provided interesting research work about the visualization of clusters in geo-referenced data. In their approach they use the width of the border line between geo-referenced vectors in SOM space to represent the distances from the vectors in input space to their BMUs. The clustering structure was detected by considering a cut distance and manipulating the width of the border lines. The SOM units were colored according their topological order in the SOM space. This methodology was tested with 2D and 3D SOM (Bacao et al. 2005a, 2005b) topology, whereby the 3D SOM (see subsection 3.3.2 for additional explanation) showed significantly better results (Gorricha and Lobo 2012). Finding homogeneous areas using this approach is quite efficient, because the border lines allow an identification of different zones where the color-coding from SOM space is not sufficient enough for clustering purposes. Figure 39 shows the clustering using a 3D SOM model and projecting the results to a geographic map.

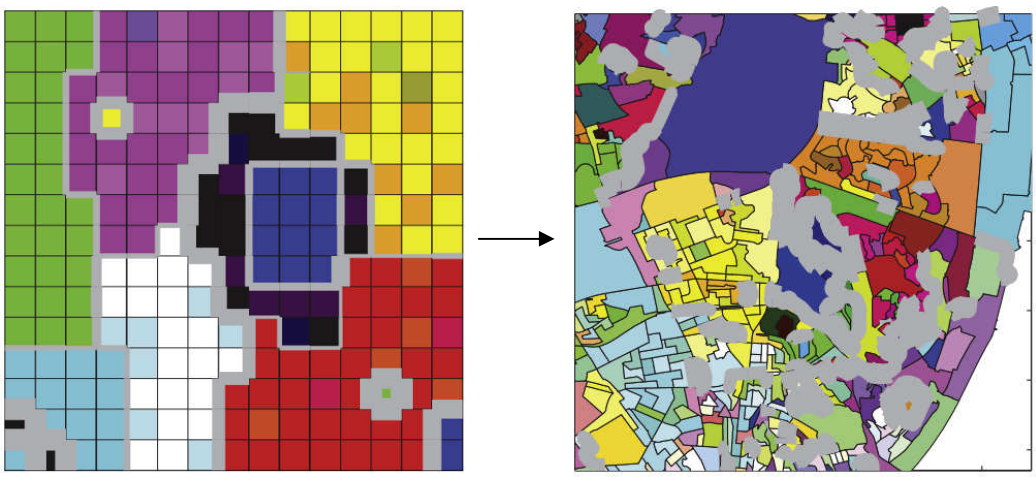


Figure 39: Cluster detection using a 3D SOM model with color-coded neurons and manipulated border line width representing the distance between geo-referenced vectors. The resulting clusters are projected into the geographic map on the right (Gorricha and Lobo 2012).

However, sometimes the membership degree in a particular SOM cluster may not be easy to judge. Therefore, fuzzy clustering algorithms (Bezdek and Pal 1992) can be applied to deal with partial membership problems.

3.3 Improved SOM Algorithms

The basic SOM algorithm is limited to factors such as not preserving the distance in attribute space, or sticking to the rectangular and hexagonal topology. The next two subsections provide an insight into the improvements done beyond traditional SOMs.

3.3.1 Dimensionality Reduction and Distance Preserving

As already mentioned in the introduction, SOM is a popular technique for non-linear dimensionality reduction. Other widely used methods are the principle component analysis (PCA) and multidimensional scaling (MDS). PCA (Johnson and Wichern 1992) is a linear data analysis method that projects data from high-dimensional space into a commonly two-dimensional principle plane. This is done by reducing data variables through eliminating minor components and finding orthogonal principle directions along the components with the largest variances. A major limitation of PCA is that it cannot find non-linear relationships defined by other than the first and seconds-order statistics (Yin 2002). In other words, it comes to a significant loss of information when dealing with data in higher dimension. Extensions to non-linear PCA exist, but are not exemplified here. MDS (Shepard 1965) is more appropriate for dimensionality reduction as it tries to preserve the distances between components from the input space when projecting to the lower dimensional output space. It needs to be mentioned that PCA and MDS are not at the same level, as PCA can be used as a projection method for MDS, which is more a class of analysis. Sammon's mapping as MDS method attempts to minimize the differences between interneuron distances in the input and output space (Sammon 1969). When speaking of input and output space, the original and projected vector space is meant. Even though Sammon's mapping shows better results than PCA, it has some issues with consecutive data input as it needs to recalculate every time. Prediction of new points is impossible. Another well-known method of MDS, and related to iterative algorithm of Sammon's mapping, is curvilinear component analysis (CCA). Demartines and Hérault (1992) introduced this algorithm, which also preserves original distances as much as possible through searching for small distances in the output space. Sammon's mapping, in contrast, focuses on small distances in the input space. All these algorithms are also not directly performing clusters, unlike SOM does. SOMs are totally different as it is based on a topology-preserving approach, making an efficient use of the available output space by accepting distortions of interneuron distances projected from input space. There is no attempt to keep the original distances after projection which creates distortions. Here provides the visualization-induced SOM (ViSOM) a solution, where the interneuron distances from input space are preserved as faithfully as possible both in the map as well as in the topology (Yin 2002). ViSOM is as simple and similar in structure as the SOM. It constrains the lateral contraction force and controls the resolution of the map by using the regularized interneuron distance as parameter. This produces a smooth and regularly graded mesh from the data points. A further improvement of ViSOM, since it does not assign any cost function, is the probabilistic regularized SOM (PRSOM). Figure 40 shows how the mentioned techniques relate to each other. The methods are horizontally ordered according to their capability of preserving the interneuron distances.

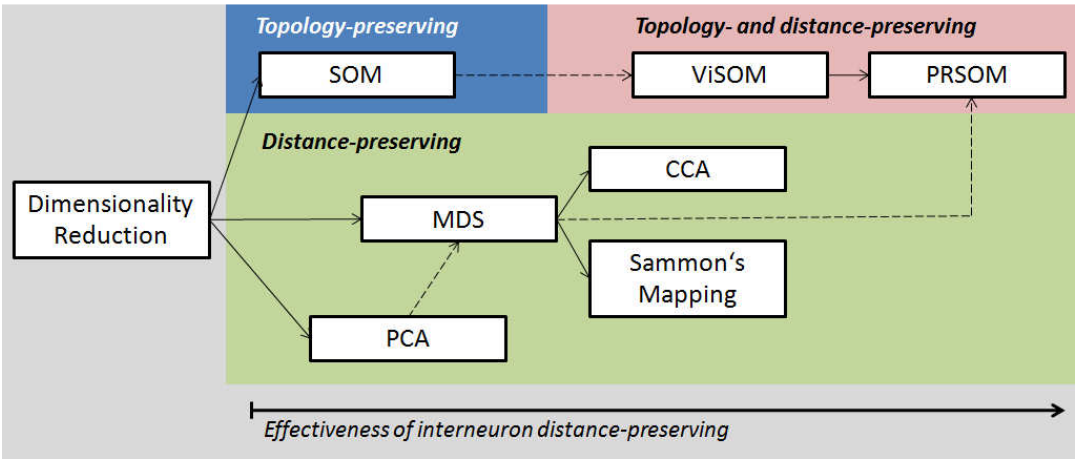


Figure 40: Diagram of dimensionality reduction methods. Ordered and aligned in relation to their capabilities of preserving topology and distances after projection from high-dimensional input space into low dimensional output space.

According to Wu and Chow (2005), PRSOM extends the sequential weight-updating rule from ViSOM with an optimization of a cost function. A color-coded area in the output space shows the accumulated probability for each neuron. PRSOM is a hybridized technique, which includes SOM and MDS into one. This makes it to an effective methodology that leads to improved visualization results compared to the other dimensionality reduction techniques SOM, ViSOM, CCV, and Sammon's mapping (see Figure 41).

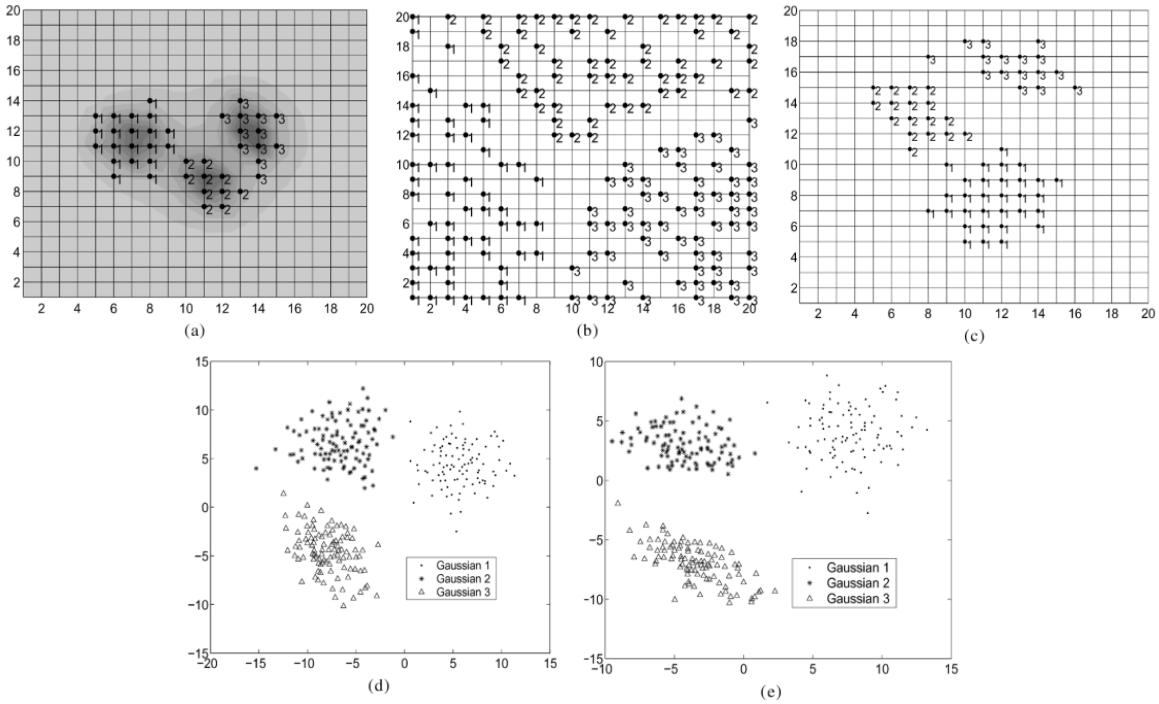


Figure 41: Visualization of a dataset with (a) PRSOM, (b) SOM, (c) ViSOM, (d) non-linear mapping by CCA, and (e) non-linear mapping by Sammon's mapping (Wu and Chow 2005).

3.3.2 About Spherical SOM and Geo-SOM

The traditional SOM topology is either hexagonal or rectangular. These topologies are coming along with a special problem, the so-called border or edge effect. Neurons at the border of the grid have fewer neighbors which results in a reduced interaction with other neurons during training and higher distortions may occur. A solution for that problem is the spherical SOM (Wu and Takatsuka 2006, Schmidt 2008). This topology does not only tackle the edge effect problem, it can also reduce the average distortion by up to two thirds (Wu and Takatsuka 2006). An illustration of a spherical SOM is given in Figure 42. The spherical SOM is shown in 3D (a), projected to a 2D plane (b), and for comparison, a conventional 2D SOM was trained and visualized with the same dataset (c). The spherical SOM shows significantly better results, especially by exploring the traditional SOM which has distortions at the neurons near the boundaries and corners of the grid, indicated by colored circles.

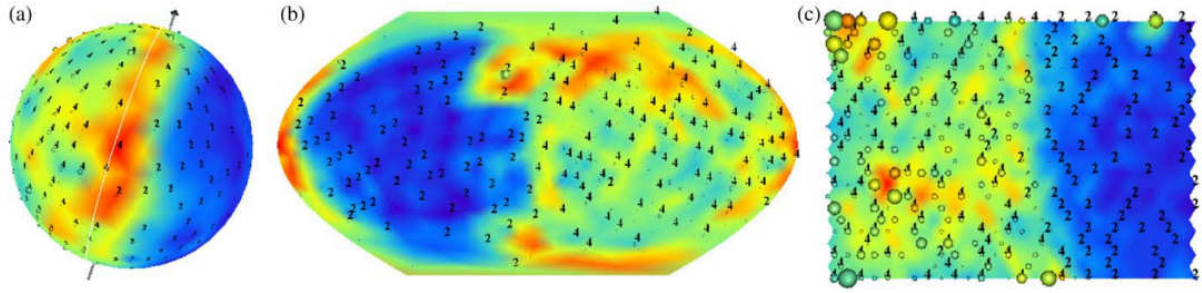


Figure 42: The trained spherical SOM (a) in 2D view, the white line indicates the cut for projection (b) into a 2D plane, and (c) a conventional SOM trained with the same dataset. The colored circles show distortions in the map (Wu and Takatsuka 2006).

Bacao et al. (2005a) developed an improved SOM algorithm that takes into account spatial dependency. The Geo-SOM architecture detects clusters in geo-referenced data which are geographically close. Homogeneous zones, as well as spatial borders are indicated. The algorithm works in a way that the BMU search consists of two steps. In the first step, the BMU search uses only the geographical coordinates of the input vectors. The neighboring units in the output space are then used for the second phase BMU search, comparing that area to all input vectors. The update procedure is the same as for the standard SOM. The neighborhood of the first BMU step can be declared with a geographical tolerance value. Increasing this tolerance creates a radius of potential BMUs in the output space. The implementation was done in MATLAB (Bacao et al. 2005b). Figure 39, illustrated earlier in this chapter, shows a variant of the Geo-SOM algorithm applied on a 3D SOM to detect clusters and cluster distances in geo-referenced data.

4. Methodology

This chapter deals with the technical implementation of the SOM visualization toolset. Basic decisions are answered at the beginning. Then, the software concept, the given sample dataset, as well as the specified data format are described in detail. All visualization methods, third-party libraries, algorithms, and system design approaches used for this work are explained throughout the following sections. At the end, used methods for the classification of SOM visualizations are given.

4.1 Fundamental Decisions

Three essential questions have to be answered before starting with the development of the software. First, what is the main purpose of this SOM visualization tool? Then, which visualizations need to be implemented? And third, based on the performance and functional support, for which technology platform should the application be developed.

Purpose of Use. This software aims to be used as a standalone SOM visualization tool, as well as for integration into extensive data mining workflows. Thus, the implementation is split into two separate parts: the library and the application. Scientists and students are major target groups that will work with the tools. The intention of continuous improvement of this work requires well-defined interfaces and explicit documentations.

Visualization Types. Component planes, hit histogram, k-Means clustering, intelligent SOM coloring, and the integration of geographic maps are chosen as important visualization methods. Further, animation functions for showing a SOM evolution and U-Matrix implementation are also considered. Each visualization has its own parameters which affect the user interface design discussed later in this chapter. The visualizations are chosen based on the needs of the target groups.

Application Platform. *Processing* provides various deployment options for its sketches (*Processing* 2013). Among others, it can run as JavaScript, Java applet, or be integrated into Java projects. Java itself allows the execution as Java Web Start or as standalone application (Oracle 2013). Therefore, a test sketch with simple user interface that does basic SOM calculations was implemented. Random SOM initialization and BMU search were then tested in Java and JavaScript. Java itself

showed no performance differences whether running as an Applet, Web Start, or standalone application. The speed of JavaScript, on the other hand, depends on the used browser. Figure 43 shows the performance test for Java compared to JavaScript running in the five commonly used internet browsers. The bar chart indicates that Java is a lot faster than JavaScript, which has a fluctuating performance in the different browsers. The tests gave some important insights, with conclusions for the further implementation. Performance and functional support were best fulfilled by the Java application. JavaScript offers good opportunities for platform-independent web-applications but is limited to the browser and its functional support is still not sophisticated enough. Also, the user interface for this purpose, with multiple windows and partitioned layout, is easier to create using Java SWING components instead of using web scripting languages. Given the fact that this software should primarily be used offline and performance is an essential issue, with large input files and *Processing* as resource-consuming visualization core, the application with the embedded SOM visualization library is implemented in Java.

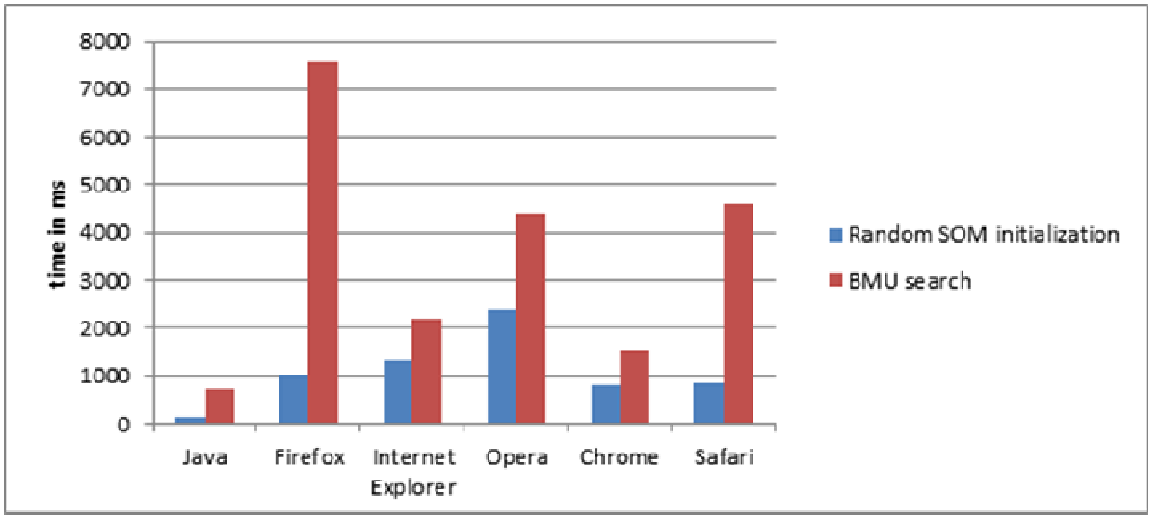


Figure 43: Java versus JavaScript. Performance results of random SOM initialization and BMU search in a Java application and running as JavaScript in five popular web browsers. 4000 neurons and 20 input vectors were used.

4.2 SOMatic Viewer Software Concept

After basic decisions are made, there is the time to find a name for the visualization toolset. It is necessary to know that this thesis work has a counterpart. Another student implements a tool which does the SOM training. The whole SOM project has the title SOMatic, with the Trainer on the one side and the Viewer on the other side. Therefore, the *Processing* SOM visualization library and the Java application have the common name SOMatic Viewer, with the supplements lib and app. Again, the purpose of this software is exploring and visualizing SOM files for finding hidden patterns, similarities, and relationships in high-dimensional datasets. Figure 39

shows the entire process of such visual data mining process. The training file (.dat) and map file, so called codebook (.cod), are saved from the SOM training software and then used for visualization and knowledge discovery in the viewer. SOMatic Trainer (Spöcklberger 2013), SOM_PAK (Kohonen 1995), or any other software which produces files in the specified format can be used. Then, the created files are feed into the SOMatic Viewer toolset. The map file describes the SOM grid dimension, its topology type and holds the high-dimensional attributes for each SOM neuron. Input vector files, for example the one used for training, which contain related information, can be mapped onto the SOM. Generally, the datasets used for analysis with SOMs may be of any domain. There is the possibility to link neurons to a Shapefile which contains the same geographic IDs as the spatially referenced input dataset. With this method, the SOM space and geographic space can be connected. The different visualization methods provided by the SOM Viewer give valuable insights and intelligible visual output for high-dimensional datasets. Interactive highlighting and selection functions improve the exploration capabilities of the software.

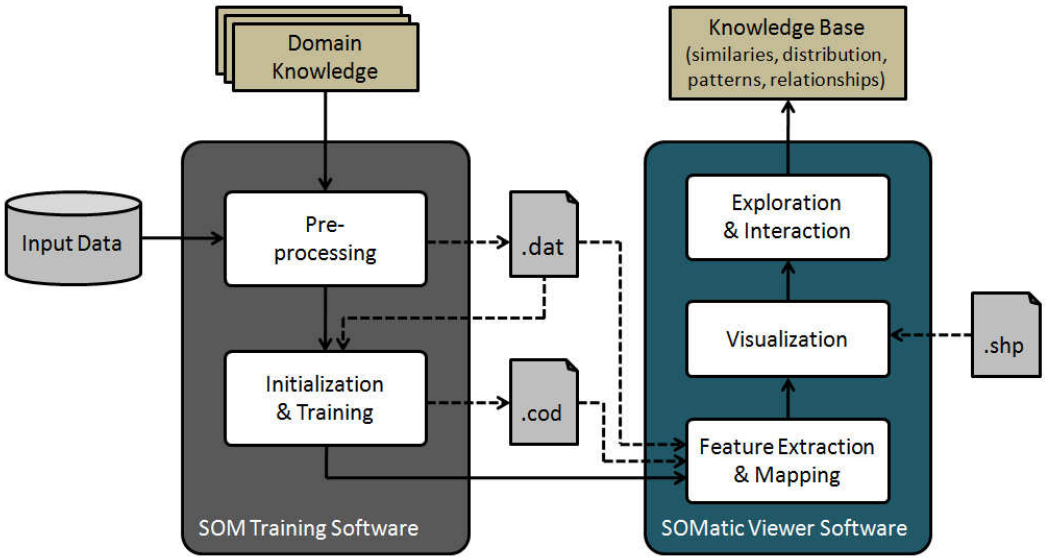


Figure 44: The entire SOM knowledge discovery workflow from data preprocessing, training to visualization. SOMatic Viewer requires three input files, of which two are specifically created with SOM training software (SOMatic Trainer or SOM_PAK).

4.3 Carinthian Census Dataset

For practical application and further discussions, a real world dataset is trained with SOMatic Trainer and then visualized with SOMatic Viewer. Census records for the region of Carinthia, Austria, are used as sample data. Figure 45 depicts the chosen area on the map. Statistik Austria, a governmental organization, collected the data in 2001 and it was last updated in 2004. This dataset contains demographic

information for 132 municipalities with 46 different attributes. The given attributes reach from administrational information, to various quantitative and proportional population data. In addition, there is a polygon Shapefile of Carinthia, produced by WIGeoGIS GmbH, containing geographic features for each municipality in the region. The dataset is going to be properly preprocessed and analyzed with the implemented SOMatic Viewer application in section 6.2. The results of the analysis, together with further thoughts and conclusions, are discussed there.

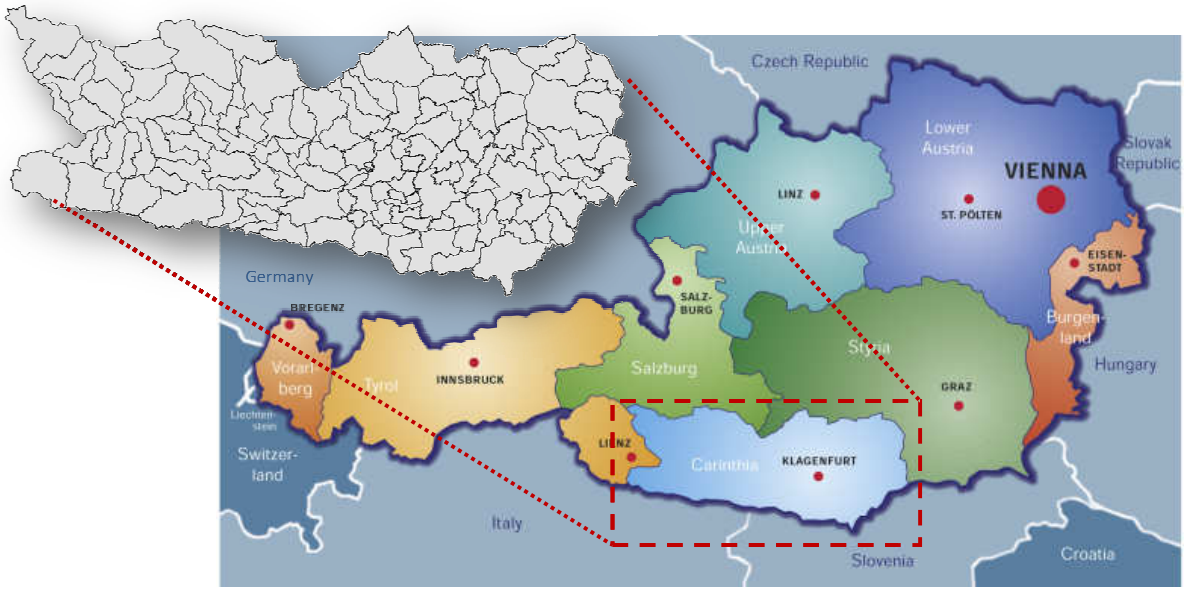


Figure 45: Map of Austria. A census records dataset for the selected region of Carinthia is used for real world data analysis (image source: http://www.locationaustria.at).

4.4 Enhanced SOM_PAK File Format

SOMatic Viewer uses the SOM_PAK file format (Kohonen 1995) with some additional enhancements. As SOM_PAK does not contain any attribute names, this feature is added using the existing comment line definition. Also, there are no further identifiers considered in the basic format. Therefore, a geographic ID can be added which is read by SOMatic Viewer. This enables to join the referenced vectors from the input data file with geographic features in a Shapefile. Figure 46 shows the basic SOM_PAK file format together with the enhanced version used for SOMatic Viewer. The software can read and process both formats.

```
3 hexa 3 2 bubble
#att AttLabel1 AttLabel2 AttLabel3
# comment line
42873 90141 41396 VectorLabel1 GeoID1
26512 57497 27067 VectorLabel2 GeoID2
30430 12131 16752 VectorLabel3 GeoID3
```

(a) Enhanced SOM_PAK data file

```
3 hexa 3 2 bubble
#att AttLabel1 AttLabel2 AttLabel3
# comment line
0.15 0.85 0.36
0.98 0.02 1.00
0.27 0.65 0.77
```

(b) Enhanced SOM_PAK map file

```
3 hexa 3 2 bubble
# comment line
42873 90141 41396 VectorLabel1
26512 57497 27067 VectorLabel2
30430 12131 16752 VectorLabel3
```

(c) SOM_PAK data file

```
3 hexa 3 2 bubble
# comment line
87.64 30.06 24.28
18.02 21.08 24.14
54.77 90.58 51.03
```

(d) SOM_PAK map file

Figure 46: Comparison of the two SOM_PAK file format version. (a) Shows the enhanced version of the SOM_PAK data file (.dat) and (b) is the new version of the map file (.cod). (c) Shows the conventional SOM_PAK data file and (d) the corresponding map file. Both formats can be used with SOMatic Viewer.

The first line declares the vector dimensionality (integer), the grid topology type (string), the map dimension in x and y direction (integer), and the defined neighborhood type (string). The input data file needs to contain only the vector dimensions, the rest is optional. For the map file all parameters are mandatory, except the neighborhood type. The parameters need to be defined in the first line and have to occur in the given order. Then, where SOM_PAK only has a normal comment line recognition, using '#', the new version can read vector attribute labels by adding the 'att' suffix without space to the comment declaration. Figure 41(c) and (d) shows the added lines, highlighted in red. The purpose behind that is to guarantee backward compatibility and to keep the existing notations. The subsequent lines contain numerical values in floating-point format, describing each vector in the given dimension. Input data files can have an optional string at the end of each line which describes the vector name. The enhanced version allows adding another string right after the label. It is used as identifier of vectors in a geo-referenced dataset. As depicted in Figure 41, the map file looks same, without vector label and additional geographic ID string. If there are missing values in the dataset, the numerical value is substituted with an 'x'. During calculations they are simply ignored by the software.

4.5 Implementation

The entire implementation is done in Java. As *Processing* uses a simplified Java syntax, it is not counted as separate programming language. Two projects are created within the non-commercial IDE Eclipse (2013): the SOM visualization

library, called SOMatic Viewer library, and the Java SWING tool, named SOMatic Viewer application. The library is a referenced project of the application and can also be exported as Java Archive File (JAR). Open-source libraries are used for both Eclipse projects (see Figure 56).

4.5.1 SOM Visualization Library in *Processing*

***Processing* is Special.** The implementation of a *Processing* library requires some understanding of how *Processing 2.0* works and how the file structure has to look like. Within the referenced core library, the PApplet is the parent class which provides access to the *Processing* methods and variables. It can further be seen as the drawing sketch itself. Two methods are important, the setup() which works as constructor and initializes the sketch, and the draw(), which contains everything that needs to be painted to the canvas. Then, as only one class can hold a sketch and extend PApplet, all other classes are using a reference of it. In other words, the PApplet is passed as parent object to any class constructor which uses *Processing* features for drawing on the same sketch. When developing outside of the *Processing* IDE, there are some other modification required which are not specifically explained here, but can be read in the official documentation (*Processing* 2013). As this project should also result in a contributed library which is going to be published to the community, the folder structure and content is critical and has to meet the *Processing* guidelines.

Component Model. An overview about the main components of the SOMatic Viewer library and how they are related to each other is illustrated in Figure 47. Four different files can be read. An input vector and codebook file is mandatory for the visualizations. There is the option to load a polygon Shapefile containing geographic features of the input vectors. Further, a project file, which contains file paths, settings, and default values, can be loaded and saved separately. Data is read into memory and kept during runtime. The file reader initializes the neuron and input vector arrays and assigns the associated data. The central element is the SOMatic Globals class, a Singleton-pattern class containing all global variables. After data is in memory, the SOM grid is created from the neuron array. Input vectors are mapped to their best matching neurons in SOM space using BMU search. Another main component is the coloring, which is fundamentally an abstract group for all visualizations methods of the SOM. The reason why BMU search is also used for coloring relates to one of the SOM grid representation techniques. The GeoMap communicates with the SOM through the Globals. K-Means is feed with the neurons of the SOM. During cluster iterations, a color scheme is created, which is then dynamically applied to the grid.

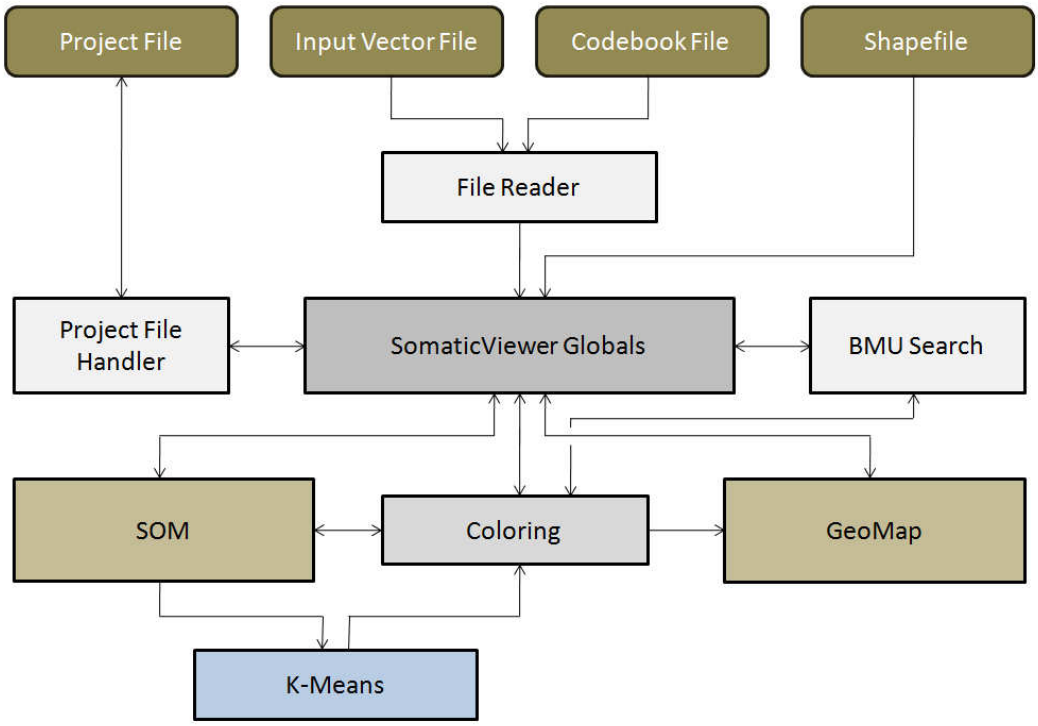


Figure 47: SOMatic Viewer Library component model

The SOM Grid. Several classes are involved and nested for drawing the SOM grid. There is the SOMGrid class, which is used to instance the top by defining the dimension in x and y direction, the given topology type and optional border spacing. Then, based on the specified parameters, the grid controller class determines the available space within the sketch according to its current width and height and scales the grid to fit best into the defined area. This provides flexible grid resizing when the sketch dimensions change. The calculated radius for each SOM neuron within the available space is passed to the topology class, which arranges the neurons based on the chosen topology type and grid dimensions. As both the rectangular and hexagonal topology require different offsets, this is handled here. Finally, the neurons are drawn at their position in the grid with the determined extent, offset, and geometry based on the topology type. Each geometry class inherits styling parameters, such as fill color, or stroke width and color, from the shape class. Once a neuron is drawn, its geometry is held within the associated instance. This is useful, as each neuron's geometry and appearance can be accessed and modified independently. The neuron geometry can return its center coordinates, inner and outer radius, as well as current fill and stroke settings. These parameters are useful for labeling or mapping objects onto the grid. Figure 48 depicts the described SOM drawing sequence based on a nested class hierarchy.

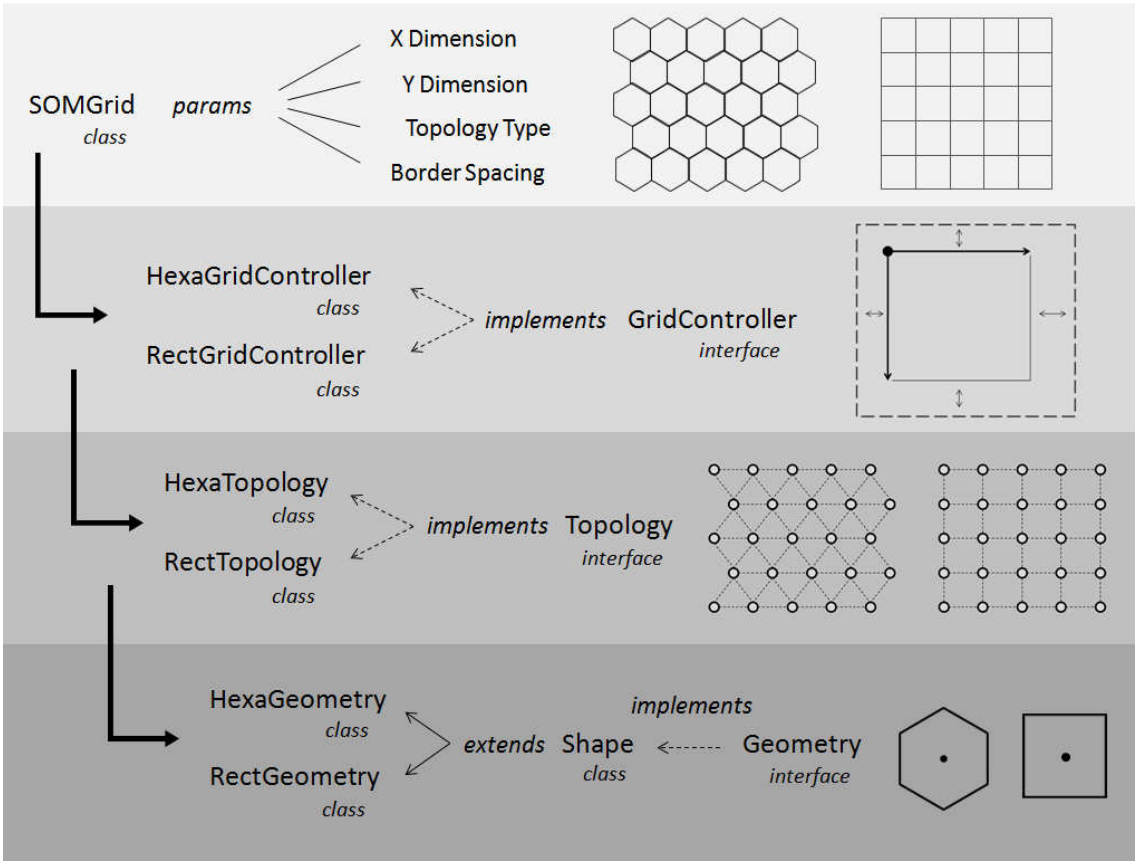


Figure 48: A SOM grid is drawn by a sequence and hierarchy of classes.

Hit Projection and Hit Histogram. The projection of input vectors onto the map grid is accomplished by the BMU search, where each input vector is compared to all neurons of the SOM based on the distance between their attributes in the same high-dimensional space. The neuron with the smallest distance to the given input vector is the BMU. Euclidean, Cosine, and Manhattan distance are provided for the BMU search. One neuron can have 0 to n matching input vectors. Once a BMU is found, both objects keep the index of each other. Figure 49 shows a class diagram of the neuron and input vector entity in unified modeling language (UML) representation. A hit can be drawn as number, quantifying the hits per neuron, or as point marker, scaled by the number of hits per neuron. As additional feature, the hits can be labeled. If enabled, the names of the vector are placed next to the hit marker on the neuron. This function has some limitations. Labels cannot be fully display with increasing number of hits per neuron, because they would overlap other elements in the sketch. The hit histogram is simply determining the minimum and maximum hit count of all neurons, assigning two different colors to these values and then calculating the color values for the number of hits in between. The lerpcolor() function in *Processing* does this job. Light colors can be used for the minimum hit count and rich colors for the maximum. The resulting map visualization

provides an understandable representation of the distribution and number of hits per neuron with bright areas for low hit density and darker areas for high hit density.

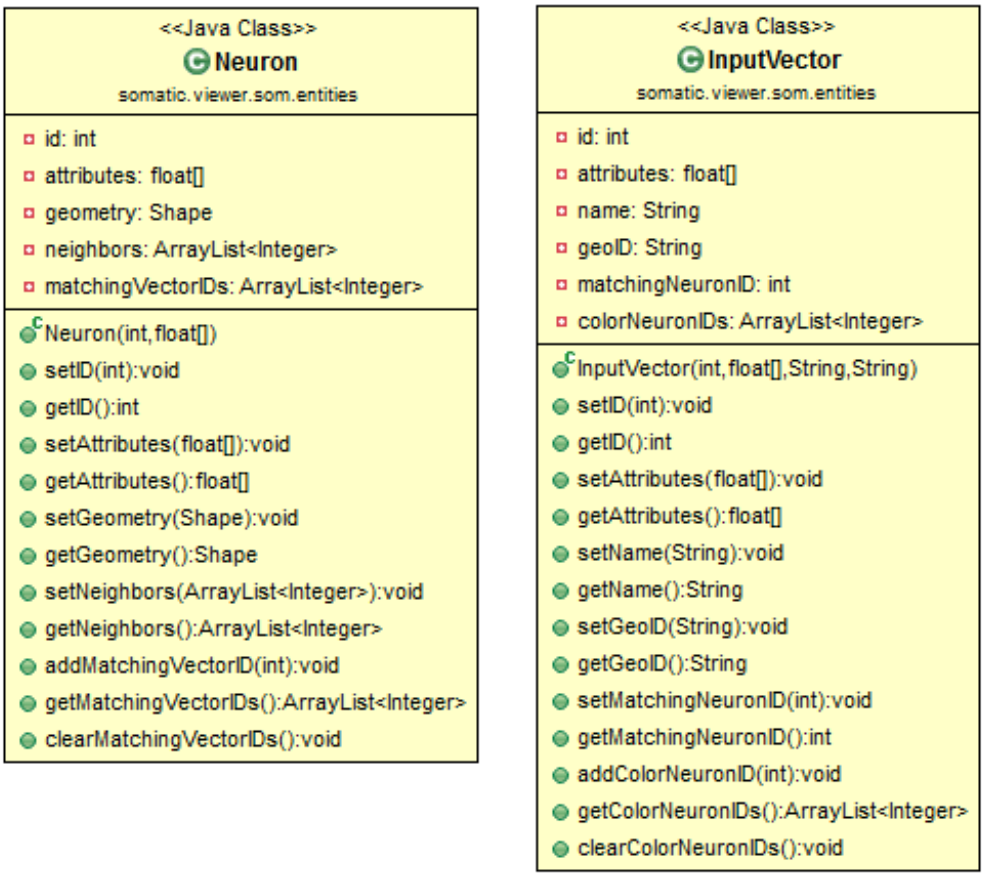


Figure 49: Class diagram of the two SOM entities, neuron and input vector.

Component Planes. Since the SOM itself contains valuable information about the data used for training, there is an interest of visualizing each attribute as a sliced piece of the SOM. Each neuron is holding an n-dimensional vector of attributes. Based on the number of selected attributes, a SOM for each component plane is drawn. The neurons are colored with the same method used for the hit histogram, two different colors are assigned to the minimum and maximum value and all colors in between are calculated. This method requires a normalization of the values first. Component planes are a popular technique for side-by-side comparison of the data distribution. SOMatic Viewer has a flexible layout on top of the component planes. This means that the grids do not only adjust automatically to the available sketch dimensions, they also get proportionally rearranged. As an example, six component planes are displayed and the frame width and height changes from 400x400 to 600x200 pixels. Then the component planes are rearranged from formerly 3 on the

x-axis and 2 on the y-axis, to 6 on the x-axis and 1 on the y-axis. A minimum spacing parameter between the grids can be set. Each component plane is labeled with the given attribute name.

GeoMap. The geographic map uses MapThing (Reades 2013), an external *Processing* library. MapThing itself imports methods of the comprehensive GeoTools Java library (OSGeo Project 2013). These toolsets offer all necessary functions to draw and manipulate Shapefiles in *Processing*. What MapThing does not provide is a method to automatically get the extent of the Shapefile. This was self-made with some functions from GeoTools. The geographic map needs one parameter for correct interaction with the SOM, which is the field name for the geographic ID from the attribute table. Additionally, the attribute category for labeling needs to be defined. The GeoMap communicates with the SOM through the global variables Singleton-pattern class. It reacts on every change in the SOM. Whenever another visualization method is applied, the GeoMap gets updated. The coloring is done by linking the colors from SOM space into the geographic space. As each feature in the GeoMap has an associated input vector projected onto the SOM, the color can be retrieved from the corresponding BMU. Another function is the interactive selection, where connected features and neurons are highlighted if one of them is selected. This selection works with an ArrayList that holds the indices of the currently selected neurons or features. Whenever there is a change in the list, both views get updated. More details about the interactive selection and highlighting are given in the next subsection 4.5.2.

U-Matrix. The unified distance matrix shows interneuron distances in the map, where each neuron is colored in relation to the distance to its neighbors. As an example, the creation of a U-Matrix from a 3x3 grid with hexagonal topology works as follows. To get the distances of one neuron to its neighbors, imaginary neurons are inserted. Figure 50 shows an illustration. The 3x3 grid results in a 5x5 interpolated matrix with new neurons in between. The {x,y} elements are holding the distance between neuron x and y and the values in {x} elements are the mean of the surrounding values. In the given example: {4,5} = distance(4,5) and {4} = mean({1,4},{2,4},{4,5},{4,7}). Euclidean is used for distance measures. The same principle is applied to a rectangular topology, with the distinction that a neuron can have the maximum of four neighbors compared to six in the hexagonal topology. In the map, the interpolated neurons disappear and only the mean distances for each neuron are visualized. Assigning light gray colors to low distances and dark ones to larger distances results in a representation where light areas are considered as clusters and black ridges indicate cluster boundaries.

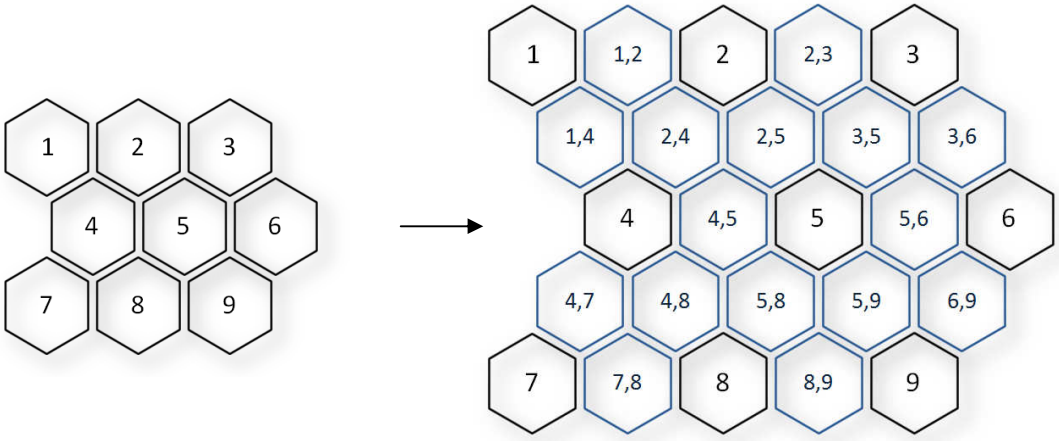


Figure 50: The U-Matrix uses interpolated cells (blue color) for interneuron distance calculation.

K-Means Clustering. The standard k-Means algorithm, explained in section 3.2, is applied for SOM clustering. Initial centroids are randomly chosen. Each iteration step during k-Means calculation affects the coloring of the map which leads to an animation of the cluster evolution. Euclidean is used as distance measure between vector attributes.

SOM Coloring Methods. Two different methods for SOM grid coloring are applied. The first one uses the topological order of the neurons as done by Himberg et al. (2001), whereas the second one is based on similarities between neurons, colored according to their BMU in a one-dimensional color SOM as described by Vesanto (2002). The topological order coloring approach uses a 2D color plane in RGB color space. The plane contains four quadrants in the lowest resolution, painted in green, red, blue, and dark magenta. With increasing resolution, transition colors are calculated in between. This plane is then stretched over the SOM grid. Depending on the resolution of the color plane and the dimensions of the SOM, a discrete or smooth coloring is the result. Figure 51 shows snapshots of this method.

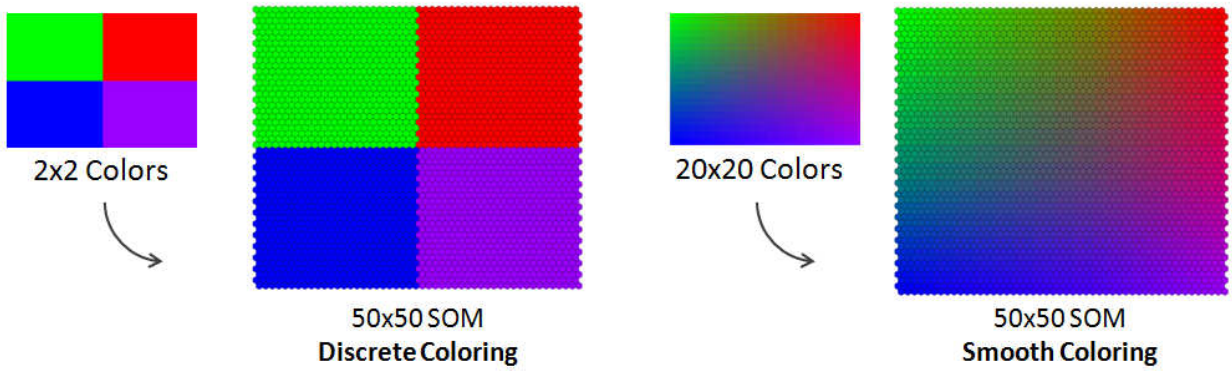


Figure 51: SOM coloring based on the topological order or the neurons.

The similarity-based coloring technique uses the SOMatic Trainer library (Spöcklberger 2013) to create a diverging one-dimensional SOM from the codebook vectors. Each component in the 1D SOM gets a distinct color. The color assignment itself is split into two separate methods. The first method uses diverging color tables from the giCentreUtils library. The second one automatically picks colors from the HSB hue circle (Vesanto 202). Both techniques use the equidistance of neurons for color assignment. An illustration of the SOM coloring procedure utilizing the HSB hue circle is given in Figure 52.

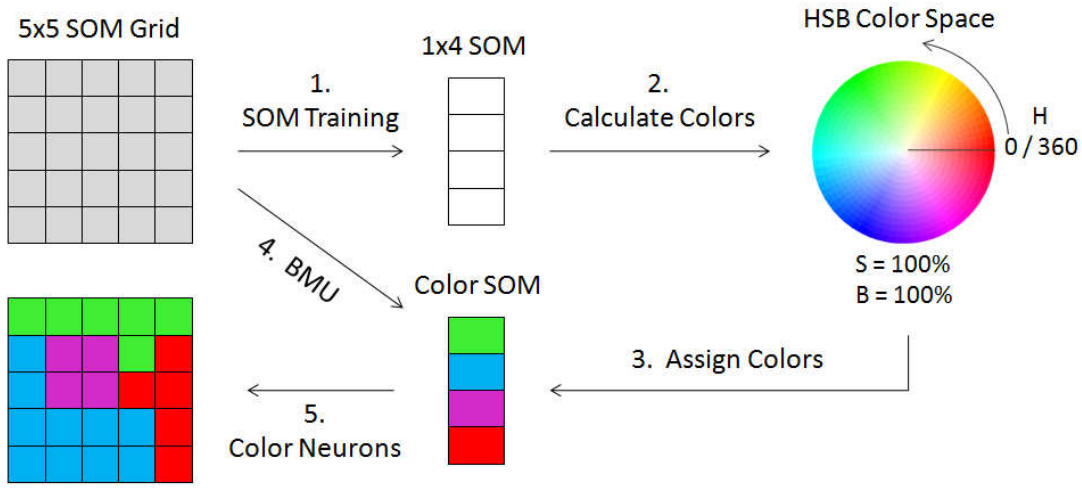


Figure 52: Process of similarity-based SOM coloring using a 1D HSB color SOM.

Training Animation. An animation of SOM training can be easily accomplished with the use of this library. After import of SOMatic Viewer library, an array of neuron objects together with all necessary SOM parameters, such as grid dimension and topology, need to be created. The chosen SOM visualization needs to be initialized, which references the former created array of neurons. Then, the associated neuron vector attributes can be update during an interval of training steps. Through the continuous drawing of the SOM sketch the visualization changes whenever new vector attributes are available. This results in an animation of the SOM training. Based on the chosen visualization, it can be seen how the SOM algorithm works and how the weights are updated in the SOM. An example is illustrated in the proof of concept, subsection 6.1.

Design and Extendibility. The aim was to implement an extendable and flexible toolset. Flexible in the meaning of as simple to use as possible, but also with access to basic classes and methods where one can create an individual SOM by putting pieces of code together. As an example, when importing the library, there is the option whether to create a new instance of the MainSOM object which already

contains all visualization methods, or to build a customized SOM by inheriting from the BasicSOM class, which provides common controls and selection functions. Sample code can be found in the *Processing* examples folder. The extendibility aspect is accomplished by providing programming interfaces which make it easier to write and integrate new classes. Figure 43 describes interfaces for the SOM grid creation. There are other interfaces for customizations such as for SOM coloring. When implementing an interface, all necessary methods are automatically created in Eclipse. The standardized method signature guarantees that new classes can communicate properly with the rest of the system. Class inheritance, mentioned above, is another way of providing a simple way to create new parts and extend a software application by accessing common methods and variables. Having a parent controls class thus makes sure that a SOM always has the same behavior in terms of interaction and events. Figure 53 shows an excerpt of the SOMatic Viewer class diagram for SOM control and coloring with implemented interface and inherited classes.

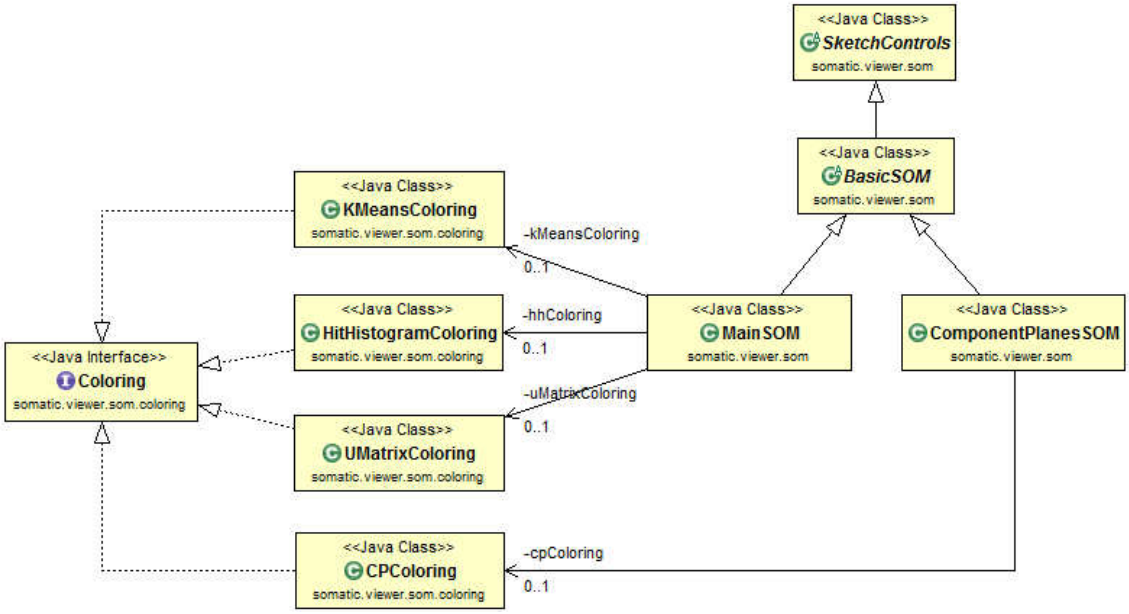


Figure 53: Class diagram showing the use of an interface for SOM coloring and class inheritance for common SOM controls and selection methods.

Project File. The project file is used to simplify the file input and to save parameters used for a single project. It can also be seen as a settings file containing file paths, default values and visualization preferences. A saved state of the project can be restored from the file. Moreover, the same project file can be used for SOMatic Trainer (Spöcklberger 2013). The structure is simple. It can contain comments and searches for explicit keys and values. See the example in Figure 54 below. Values are updated when saving the file. If a key is missing, the application adds it together with its associated value to the end of the file.

```
1  ### FILE PATHS ###
2  # normalized original data file, used for training and hit projection
3  trainingvectorfile = C:\Users\user1\SOMatic\Viewer\data\tvCC.dat
4  # SOM file
5  codebookfile = C:\Users\user1\SOMatic\Viewer\data\50x50_400k.cod
6
7  ### INITALIZATION PARAMETER ###
8  kMeans = 5;
9  showHits = true;
10 showHitMarker = true;
11 showHitLabels = false;
```

Figure 54: Excerpt of a SOMatic project file.

4.5.2 The Java Application

SWING and *Processing*. Java SWING API is used for the graphical user interface (GUI) and thus works as presentation layer of the SOMatic Viewer application. The SWING toolkit provides a varied range of GUI components to create platform-independent software applications. The SOMatic Viewer sketches are embedded into panels. GUI components call methods of the sketch, listeners detect if there is an interaction with the sketch, and variables are updated. There is no direct reference from the sketch to the rest of the application; everything is loosely coupled. The GUI frames as well as the sketches are resizable. Visualizations automatically adjust to the given frame dimensions. An embedded sketch has to be initialized and started. After that it keeps running until the application is closed. There is also the possibility to stop it programmatically.

Attribute Tables. For better data exploration reasons, two attribute tables are integrated. The codebook vector and input vector data is read from memory and added to the table frames, which can be handled independently from each other. Both tables have an attribute search function with auto-completion. The column containing the search result is automatically focused and highlighted. Both tables are connected to the SOM visualizations. Their selections get mutually updated. The interactive selection process is described in the next paragraph.

Interactive Selection. Already a common feature in other SOM software, the interactive selection and highlighting between various data representations was implemented in SOMatic Viewer. This function is provided for the visualizations as well as the attribute tables. The SOMatic Viewer Globals class holds variables which are sequentially called and updated from all presentation components. These components can be seen in Figure 55. The sketches simply use their draw() method, which is called in an infinite loop and thus continuously updates the global

selection parameters. In case there is no continuous drawing done, the depending selection methods are called after certain events. Slightly more complicated is the method for the attribute tables as they are part of the GUI. Because of loose coupling reasons there is no direct reference to any of the events within each sketch. The solution comes from mouse listeners which detect if the user has clicked into a sketch panel and consequently updates the selection variables. Multiple selection can be done by holding the CTRL key while selection neurons in the SOM, features in the geographic map, or rows in the attribute tables.

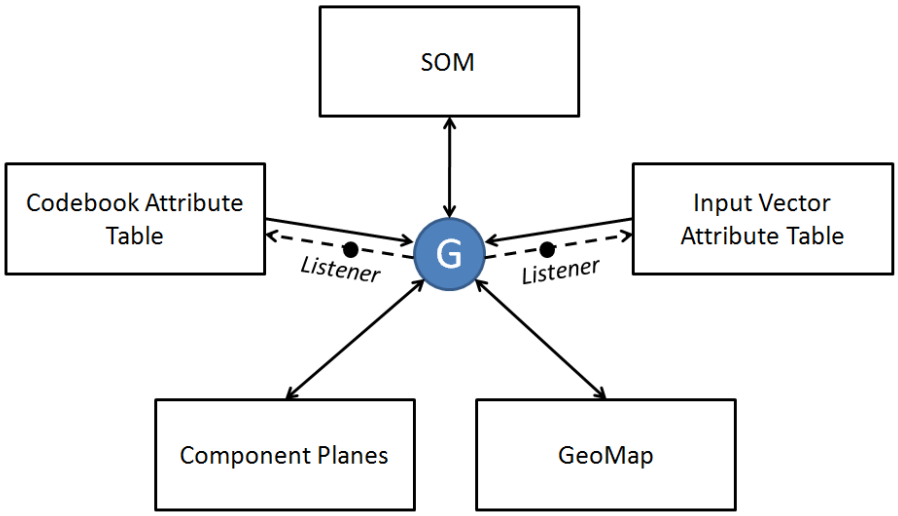


Figure 55: Abstraction of the presentation components which sequentially access and update the global variables used for interactive selection. To guarantee a loose coupling between the GUI and the library, the attribute tables use click listeners on the panels which contain the sketches.

External Libraries. SOMatic Viewer uses several external libraries, which are depicted in Figure 56. The Java application imports two libraries, which are the suggest field, used for searching through attribute tables, and XSwing, which offers some sophisticated GUI elements such as the collapsible panel, used for the SOM controls in the main frame. The SOMatic Viewer *Processing* library is a referenced project of the application that includes six more libraries. There is the SOMatic Trainer library which is used together with color tables from giCentreUtils to create SOM-based coloring. *Processing* core is the heart of the toolset with its sketching capabilities. Cloning makes deep-copies of objects and finally there is the MapThing library, which references GeoTools. These two in combination provide extensive Shapefile handling and visualization functions.

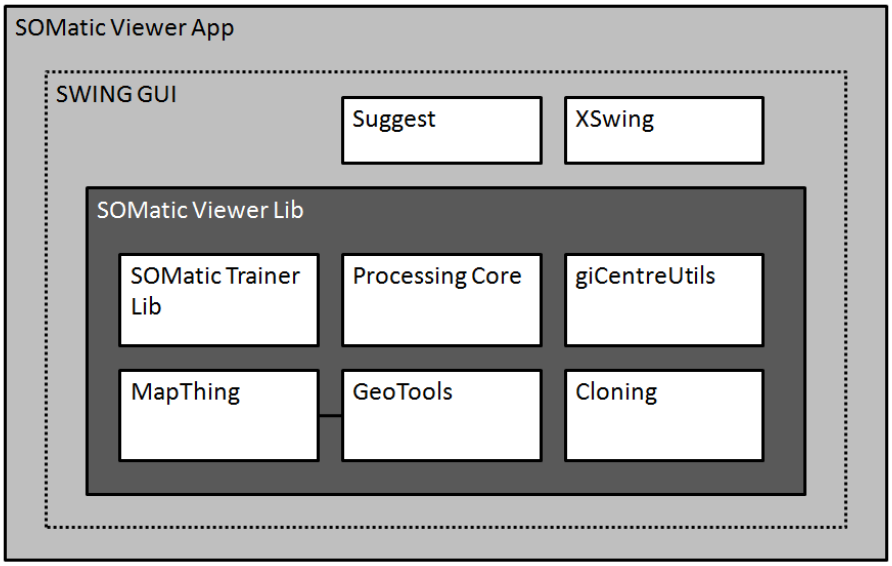


Figure 56: External libraries used for SOMatic Viewer.

4.6 Visualization Classification

First, getting a complete list of SOM visualizations is even after extensive literature review and software testing simply not possible. But, an appropriate classification can be done based on the different types of visualizations that were found. An important remark is that the only visualizations that are considered are those which do not require a sequence of software tools or complex preprocessing and visualization workflows. In other words, only reproducible visualizations are classified. Further, SOM grid coloring methods are also not considered, even though they can be seen as distinct visualization techniques.

In total, ten classification characteristics for 23 different SOM visualization techniques have been elaborated. They are clearly described as follows:

Codebook Vectors / Input Vectors. These two determine which types of vectors are used for the visualization. Component planes, for example only use codebook vectors as only the SOM itself is visualized. Hit histograms solely visualize the input vectors, projected onto the map. Other visualizations require both for a more enhanced representation of the input and output space.

Cluster Indication. The visualization shows clusters through coloring and other methods of separation or agglomeration. Therefore, trajectories may identify clusters where hit diagrams do not, because there is no cluster delineation.

Distance. The representation visualizes distances between neurons using colors, connections, or a distance-preserving projection.

Density. The representation visualizes neurons density using colors, contours, or markers showing the magnitude.

Cluster Connections. Visualizes the connection or correlation between clusters based on certain characteristics. Usually, lines are used to connect two or more related clusters.

Movement. Shows the movement of vectors or attributes through the SOM space.

Parameterization. This describes if the visualization can be calibrated by modifying certain parameters. These can be the number of clusters, value of significance, or distance threshold and so on. Coloring or marker settings are not considered as parameterization. Also different distance measures for hit projection are left out.

Vector Correlations. Explicitly visualizes the correlation between single vectors in the SOM or projected space by connecting them with a line or using some sort of color differentiation.

Match Accuracy. Determines the accuracy of BMU search results for vector projection.

Based on these ten criteria, a classification matrix is created. The results are given at the end of the next chapter.

5. Results

The SOMatic Viewer toolset serves as technical result of this thesis work. It consists of a Processing library for SOM visualization and the Java application. The software provides interactive functionalities for SOM data visualization and exploration. A classification matrix for SOM visualizations is the other research outcome.

5.1 SOMatic Viewer *Processing* Library

The SOMatic Viewer *Processing* library comes as JAR file which can be imported into a *Processing* sketchbook or Java application. It has the folder structure for official contribution to the *Processing* community together with all other necessary files, such as documentation, source, and sample code files. The library import and visualization of SOM files requires just a few code lines. Visualization parameters can be set from a user interface, which does not come with the library.

The following visualizations are available:

- Hit projection
  - Hit histogram
  - Hit diagram
    - Numerical hit count
    - Marker showing the location and magnitude of hits
    - Labeling of the projected vector names
- U-Matrix
- Component planes
  - Automatic plane arrangement within sketch dimensions
- SOM coloring
  - Discrete or continuous coloring from 2D color plane, following the toplogical order of the neurons in the SOM
  - Similarity-based coloring with diverging-diverging colors schemes and HSB colors
- GeoMap
  - Labeling of the map features
- k-Means clustering

Visualizations can be embedded into separate frames for simultaneous view and independent resizing. The library provides several interaction and map control features for each visualization:

- Map controls
  - Zooming
  - Panning
  - Key events
- Interactive selection and highlighting
  - Single or multiple neuron selection (see Figure 59)
  - Selection gets updated in connected visualizations (see Figure 57)

Project files (.sprj) can be loaded and saved. These files contain the parameters and file paths of a single SOMatic project.

5.2 SOMatic Viewer Java Application

This platform-independent Java application comes as runnable JAR with integrated SOMatic Viewer library project. It can be executed without installation. Technical details of the software are given below:

- Multiple data visualization and exploration windows (see Figure 57)
  - Main frame containing the main SOM grid
  - Component planes

- Attributes tables
- Geographic map (see also Figure 58)

- Interactive Selection and Highlighting between multiple representations (see Figure 57)

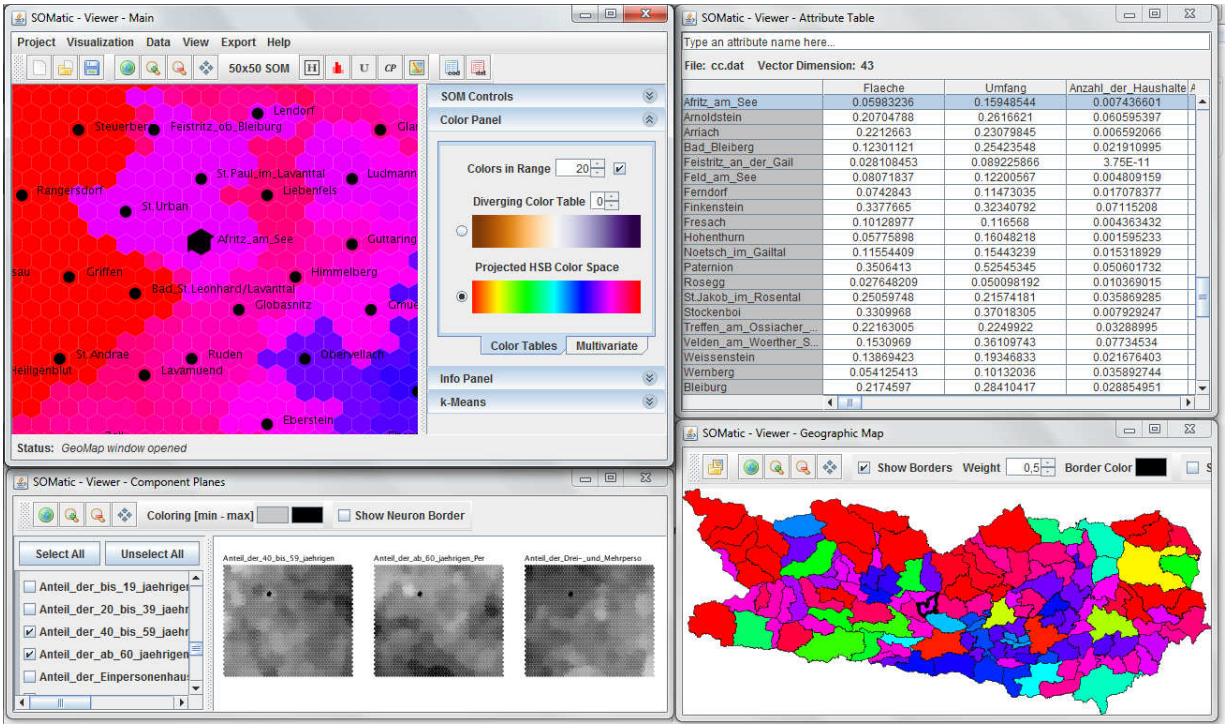


Figure 57: SOMatic Viewer application updates the selection in multiple windows. The main frame shows a zoomed view of the SOM grid (upper left), the attribute table sets its focus to the row of the selected neuron, the geographic map provides a linked view of the SOM with the highlighted region, and component planes (lower left) identify the selection in all slices of the SOM.

- Graphical User Interface (see Figure 59)
  - Toolbars with map controls
  - Preferences window
  - Visualization controls
  - SOM coloring panel
  - Hit info panel
  - Status bar

File dialogs are used for input or output actions. Implemented functions are:

- Sketch export as image or PDF
- Loading of codebook and input vector files
- Loading and saving of project files
- Loading of Shapefiles with attribute field selection

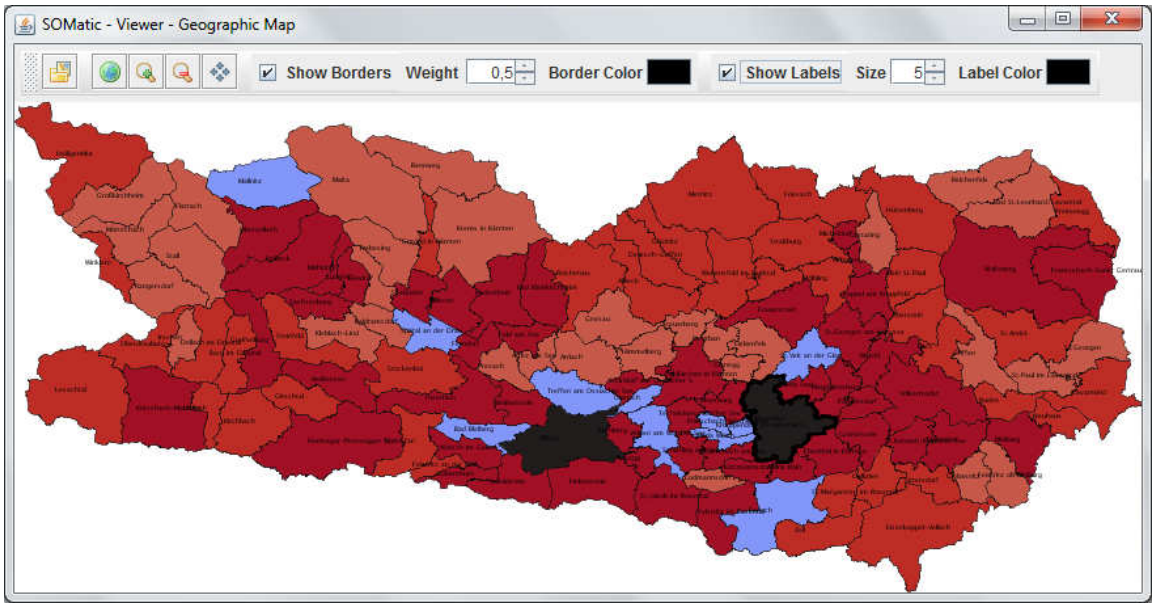


Figure 58: The SOMatic Viewer geographic map window provides several controls.

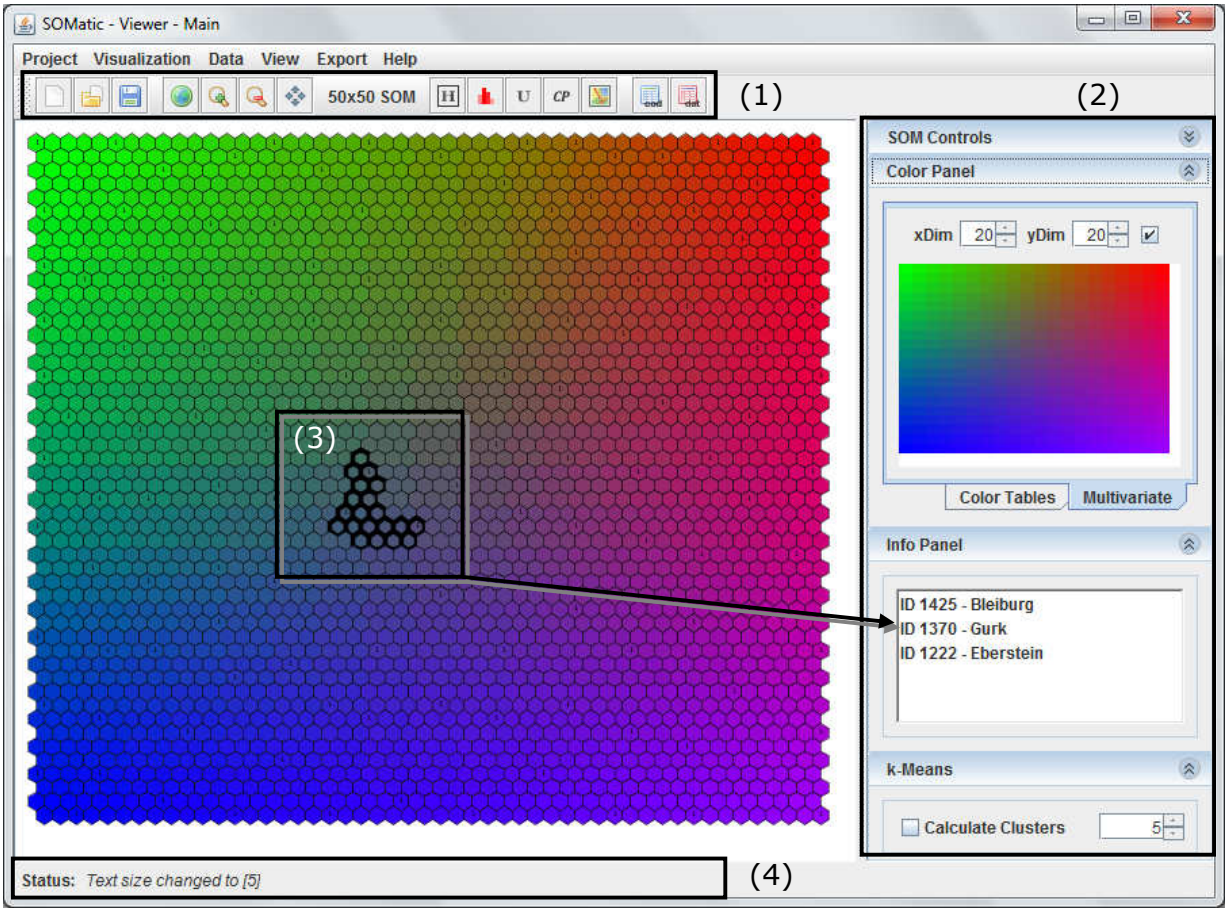


Figure 59: SOMatic Viewer main frame. (1) Toolbar with shortcuts for the project, map, and visualizations, (2) collapsible control panels for grid control, SOM coloring, hit information, and k-Means clustering, (3) selection of multiple neurons in the grid, (4) status bar.

5.3 SOM Visualization Classification

21 types of SOM visualizations characterized by 10 classification criteria, described in section 4.6, are ordered into four visualization categories. Based on extensive literature research, the resulting classification matrix looks as follows:

	Visualizing the SOM itself								
	SOM Space	Cluster Space		Distortion Space					
	Component Planes	k-Means	Hierarchical	Vector Fields	Cluster Connections	U-Matrix	P-Matrix	U*-Matrix	Smoothed Density Histogram
Codebook Vectors	✔	✔	✔	✔	✔	✔	✔	✔	✔
Input Vectors	x	x	x	x	x	✔	✔	✔	✔
Cluster Indication	✔	✔	✔	✔	✔	✔	✔	✔	✔
Distance	x	x	x	x	✔	✔	x	✔	x
Density	x	x	x	✔	x	x	✔	✔	✔
Cluster Connections	x	x	x	x	✔	x	x	x	x
Movement	x	x	x	x	x	x	x	x	x
Parameterization	x	✔	✔	✔	✔	✔	✔	✔	✔
Vector Correlations	x	x	x	x	x	x	x	x	x
Match Accuracy	x	x	x	x	x	x	x	x	x

Table 3: Classification matrix for SOM visualizations. Part 1: Visualizing the SOM itself.

	Projections onto the SOM							
	Hit Projection		Multivariate Symbolization	Graph Representation		Path Visualization		Classes
	Data Histogram	Sky Metaphor	Component Charts	Neighborhood Graph	Minimum Spanning Tree	Trajectories	Metro Map	Class Map
Codebook Vectors	x	x	✔	✔	✔	✔	✔	x
Input Vectors	✔	✔	✔	✔	✔	✔	x	✔
Cluster Indication	x	✔	x	✔	✔	✔	✔	✔
Distance	x	✔	x	x	✔	x	x	x
Density	x	✔	x	x	x	x	x	x
Cluster Connections	x	x	x	✔	✔	x	x	✔
Movement	x	x	x	x	x	✔	✔	x
Parameterization	x	✔	✔	x	✔	✔	✔	✔
Vector Correlations	x	✔	✔	✔	✔	✔	✔	✔
Match Accuracy	x	x	x	x	x	x	x	x

Table 4: Classification matrix for SOM visualizations. Part 2: Projections onto the SOM.

	Projections onto the SOM		Projections from the SOM			Linking from the SOM
	Match Accuracy		Distance Preserving Low-Dimensional Space		Ordered 2D Space	Geographic Space
	Response Surface	Position Accuracy Marker	Sammon's Mapping	Principal Component Analysis	Parallel Coordinate Plot	Geographic Maps
Codebook Vectors	x	x	✔	✔	✔	✔
Input Vectors	✔	✔	✔	✔	✔	✔
Cluster Indication	x	x	x	x	x	✔
Distance	x	x	✔	✔	x	x
Density	x	x	x	x	x	x
Cluster Connections	x	x	x	x	x	x
Movement	x	x	x	x	x	x
Parameterization	x	x	x	x	x	x
Vector Correlations	x	x	✔	✔	✔	✔
Match Accuracy	✔	✔	x	x	x	x

Table 5: Classification matrix for SOM visualizations. Part 3: Projections onto the SOM, Projections from the SOM, Linking from the SOM.

6. Data Analysis and Discussion

This chapter is mainly about the analysis of the Carinthian census dataset. Another major part is the proof of concept, where the implemented visualizations are compared with results produced by other SOM tools. A short excursion about a possible parallelization of processing and visualization tasks is given at the end.

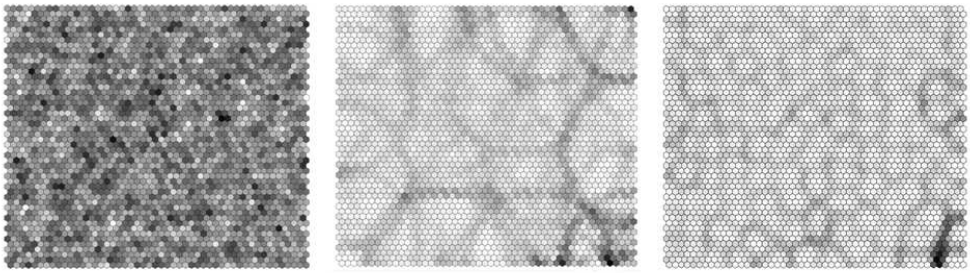
Before starting with the proof of concept, there should be said that no user testing was possible for the SOMatic Viewer toolset. The major reasons were first, the focus on the technical and theoretical research of SOM visualizations together with functional implementation priorities, and second, the fact that proper user testing would have been very time-consuming at the expense of other, more essential parts of this thesis work.

6.1 Proof of Concept

The proof of concept intends to show that the implemented visualizations are working properly and are giving correct results. One part deals with the integration for training animation into SOMatic Trainer. The other subsection compares the visualization results with those from other SOM software. The Carinthian census data was used as input dataset for all of the given examples.

6.1.1 Training Animation

A required function was to provide the ability of training animation. Therefore, the SOMatic Viewer *Processing* library was integrated into SOMatic Trainer for animation of the entire training process. Component planes and the U-Matrix can be selected as visualization. Figure 60 show the evolution of a 50x50 hexagonal SOM with 400,000 iterations in 4 training phases, visualized as U-Matrix. The initialization state is depicted in (a). In the training phase, it can be seen how weights are updated after BMU search. Circles identify the current neighborhood radius (b). The resulting U-Matrix identifies clusters in the map by visualizing the mean distance from reach neuron to its neighbors (c).



(a) Random Initialization (b) Training State (c) Result

Figure 60: Animation of SOM training in SOMatic Trainer. Three training states for the Carinthian census dataset are shown: (a) random seeding of the neuron vector, (b) BMU search during training, (c) the resulting U-Matrix cluster structure.

6.1.2 Comparison with Java SOMToolbox and SOMine

The purpose behind this comparison is to show that the visualizations provide correct results. Of course, there is never a 100 percent similarity, but similar regions should be identified and visualized in the map. Also the position and distribution of projected hits can be compared. This is done in the first example. A 30x30 rectangular SOM was trained with 100,000 iterations in SOMatic Trainer and visualized in SOMatic Viewer. The same settings were used for the training of a Growing SOM in Java SOMToolbox. Moreover, Euclidean distance was used for the projections in the hit histogram visualization of SOMatic Viewer. Even though Java SOMToolbox uses another SOM algorithm, the results are surprisingly similar. In Figure 61, the orange rectangle in the upper left corner identifies the cities Klagenfurt (corner) and Villach in both hit histograms. Additionally, only two municipalities have the same BMU, which are Feistritz ob Bleiburg and St. Paul im Lavanttal. Both SOMs show them on the same neuron in almost the same region, marked with a gray rectangle.

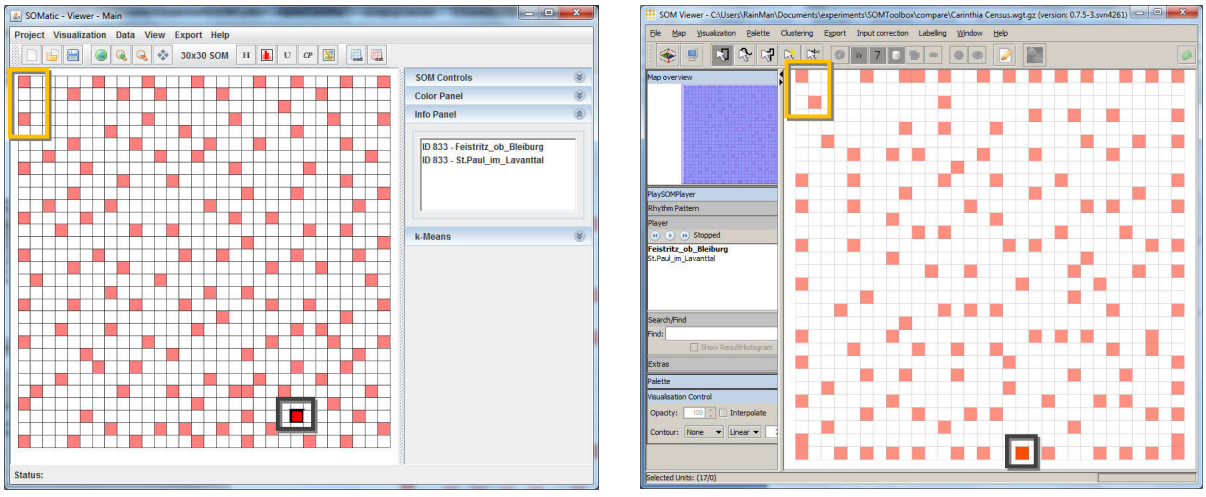


Figure 61: Hit histogram comparison in SOMatic Viewer (left) and Java SOMToolbox (right).

Another comparison was done with the identical SOMs from above, but visualizing distances between neurons using the U-Matrix (see Figure 62). Again, both matrices show the same large distances detecting a small, well separated cluster in the upper left corner. The overall cluster structure is somehow related, but the resolution is quite different, as the Java SOMToolbox uses a more detailed U-Matrix visualization for each neuron.

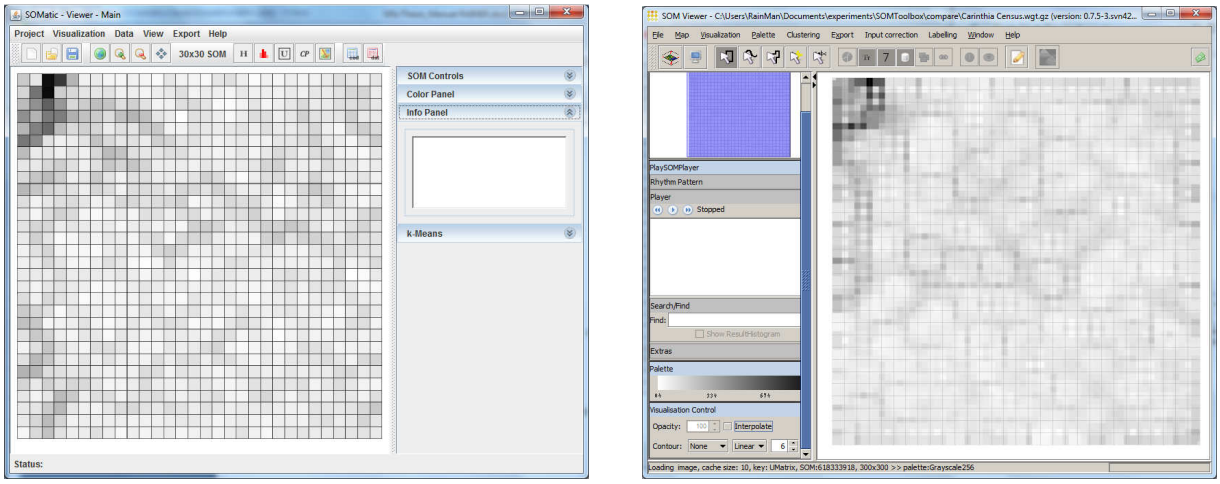


Figure 62: U-Matrix visualization comparison in SOMatic Viewer (left) and Java SOMToolbox (right).

Finally, the component plane representation is compared with the one in Viscovery SOMine. For this case, the same map file is used as input for both the SOMatic Viewer and SOMine because the proprietary software is able to read codebook files. The SOM has a hexagonal topology and the dimension of 30x30. In Figure 63, two component planes were visualized. In both applications, cluster areas were outlined in exactly the same regions with exactly the same color intensity. The difference here is that SOMine provides a sequential color schema, whereas SOMatic only interpolates the range between two colors from the minimum to the maximum value.

Conclusion. The conclusion of these comparisons is that the entire SOMatic toolset bundle works correctly and delivers results which can, to a certain level, compete with comprehensive SOM software applications.

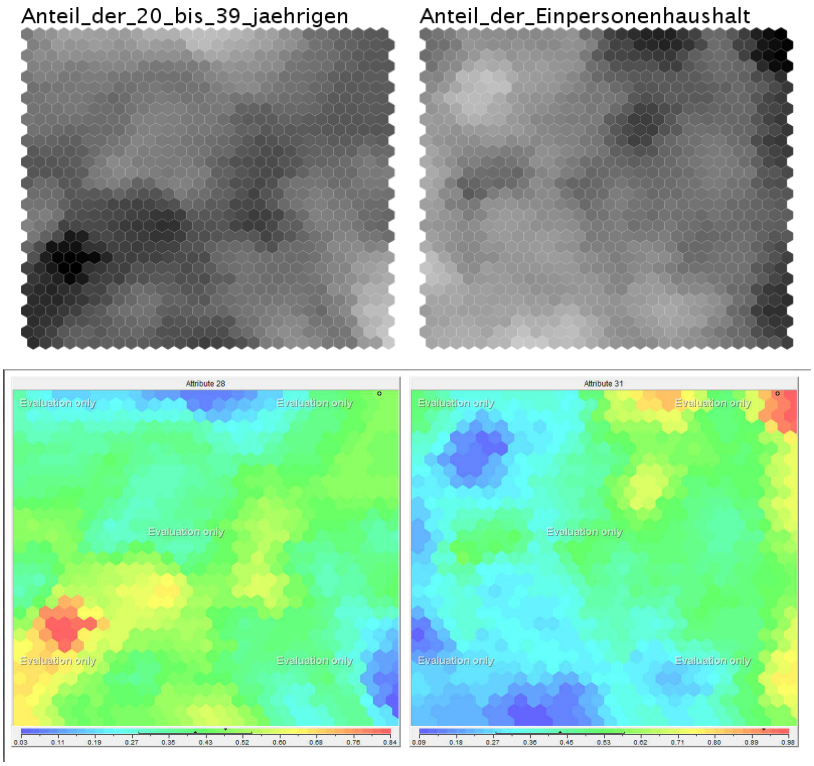


Figure 63: Comparison of component plane visualizations in SOMatic Viewer (upper) and Viscovery SOMine (lower).

6.2 Analysis of Carinthian Census Data

This is an example of how SOMatic Viewer can be used for highly dimensional real-world data analysis. The results might have practical relevance for the regional government and public institutions. For instance, the development and change within the population among the municipalities can be tracked down. The described procedure of this analysis can serve as reference for follow-up research activities. Some basic information about the Carinthia census records are given in section 4.3.

6.2.1 Preprocessing

Data preprocessing is the first step to get correct and meaningful results. For every variable it is necessary to decide carefully whether it is important to include it and how it should be preprocessed such that meaningful comparisons can be made between geographic objects. The Carinthia census records contain 43 quantitative attributes. Population density was manually added. In total, 21 attributes were chosen for SOM training. Table 6 and 7 provide an overview about the census dataset attributes, which variables are used for training and how each of them is normalized. All 132 Carinthian municipalities are described through these attributes.

Attribute Name	English Description	Used for Training	Normalized to
Flaeche_m2	Area in square meters		
Umfang_m	Perimeter in meters		
Anzahl_der_Haushalte	Number of households		
Anzahl_der_Einwohner	Number of population		
Anzahl_der_Maenner	Number of male		
Anzahl_der_Frauen	Number of female		
Anzahl_der_bis_19_jaehrigen_Personen	Number of persons with age 19 and younger		
Anzahl_der_20_bis_39_jaehrigen_Personen	Number of persons with age between 20 and 39		
Anzahl_der_40_bis_59_jaehrigen_Personen	Number of persons with age between 40 and 59		
Anzahl_der_ab_60_jaehrigen_Personen	Number of persons with age 60 and older		
Anzahl_der_Einpersonenhaushalte	Number of one-person households		
Anzahl_der_Zweipersonenhaushalte	Number of two-person households		
Anzahl_der_Drei-_und_Mehrpersonenhaushalte_(Familien)	Number of three- or more-person households (families)		
Anzahl_der_Personen_mit_Universitaetsabschluss	Number of persons with university degree		
Anzahl_der_Personen_mit_Matura	Number of persons with high school certificate		
Anzahl_der_Personen_ohne_Matura	Number of persons without high school certificate		
Anzahl_der_ledigen_Personen	Number of unmarried persons		
Anzahl_der_verheirateten_Personen	Number of married persons		
Anzahl_der_verwitweten_Personen	Number of widowed persons		
Anzahl_der_geschiedenen_Personen	Number of divorced persons		
Anzahl_der_Personen_mit_oesterreichischer_ Staatsangehoerigkeit	Number of persons with Austrian citizenship		
Anzahl_der_Personen_mit_EU-Staatsangehoerigkeit_(ohne_oesterr.)	Number of persons with EU citizenship (without Austrian)		
Anzahl_der_Personen_mit_sonstiger_Staatsangehoerigkeit	Number of persons with other citizenship		

Table 6: Carinthia census records attribute breakdown table, part 1. Variables not used for training.

Attribute Name	English Description	Used for Training	Normalized to
Anteil_der_Maenner	Proportion of male	✔	Anzahl_der_Einwohner
Anteil_der_Frauen	Proportion of female	✔	Anzahl_der_Einwohner
Anteil_der_bis_19_jaehrigen_Personen	Proportion of persons with age 19 and younger	✔	Anzahl_der_Einwohner
Anteil_der_20_bis_39_jaehrigen_Personen	Proportion of persons with age between 20 and 39	✔	Anzahl_der_Einwohner
Anteil_der_40_bis_59_jaehrigen_Personen	Proportion of persons with age between 40 and 59	✔	Anzahl_der_Einwohner
Anteil_der_ab_60_jaehrigen_Personen	Proportion of persons with age 60 and older	✔	Anzahl_der_Einwohner
Anteil_der_Einpersonenhaushalte	Proportion of one-person households	✔	Anzahl_der_Haushalte
Anteil_der_Zweipersonenhaushalte	Proportion of two-person households	✔	Anzahl_der_Haushalte
Anteil_der_Drei-_und_Mehrpersonenhaushalte_(Familien)	Proportion of three- or more-person households (families)	✔	Anzahl_der_Haushalte
Anteil_der_Personen_mit_Universitaetsabschluss	Proportion of persons with university degree	✔	Σ (Anzahl_Uni + Anzahl_Matura + Anzahl_ohne_Matura)
Anteil_der_Personen_mit_Matura	Proportion of persons with high school certificate	✔	Σ (Anzahl_Uni + Anzahl_Matura + Anzahl_ohne_Matura)
Anteil_der_Personen_ohne_Matura	Proportion of persons without high school certificate	✔	Σ (Anzahl_Uni + Anzahl_Matura + Anzahl_ohne_Matura)
Anteil_der_ledigen_Personen	Proportion of unmarried persons	✔	Anzahl_der_Einwohner
Anteil_der_verheirateten_Personen	Proportion of married persons	✔	Anzahl_der_Einwohner
Anteil_der_verwitweten_Personen	Proportion of widowed persons	✔	Anzahl_der_Einwohner
Anteil_der_geschiedenen_Personen	Proportion of divorced persons	✔	Anzahl_der_Einwohner
Anteil_der_Personen_mit_oesterreichischer_ Staatsangehoerigkeit	Proportion of persons with Austrian citizenship	✔	Anzahl_der_Einwohner
Anteil_der_Personen_mit_EU-Staatsangehoerigkeit_(ohne_oesterr.)	Proportion of persons with EU citizenship (without Austrian)	✔	Anzahl_der_Einwohner
Anteil_der_Personen_mit_sonstiger_Staatsangehoerigkeit	Proportion of persons with other citizenship	✔	Anzahl_der_Einwohner
Durchschnittliche_Haushaltsgroesse	Average size of household	✔	Anzahl_der_Einwohner / Haushalte
Bevölkerungsdicht_km2	Population density in square kilometers	✔	Anzahl_der_Einwohner / Flaeche

Table 7: Carinthia census records attribute breakdown table, part 2. Variables used for training.

Basically, the order of suitable variable preprocessing is:

1. Choose which variables to use.
2. For each variable, decide whether it should be used on its own or within a composite variable.
3. Normalize/standardize with the goal of accounting for magnitude differences.

Since total counts of population variables are highly correlated with total population, it makes no sense to include them for training. Instead, solely proportional attributes are used which allow a meaningful comparison and should also lead to spatial clusters. All attribute variables are normalized in the range between 0 and 1.

6.2.2 Training

The SOMatic Trainer tool, implemented by Spöcklberger (2013), is used for training. To get a detailed SOM for the given input dataset, a resolution of 50x50 neurons is chosen. Generally, the size of a neuron lattice should be chosen that each input vector has the possibility to occupy at least one single neuron (Kohonen 1998). In this case a significantly larger lattice is used to get a SOM with higher granularity and more detailed cluster structure. The training is split up into three consecutive cycles. After each cycle, learning rate and neighborhood radius gets decreased and the number of training runs gets increased. The first training cycle is establishing broad, global structures and the second and third cycle firm up regional and local structures in the SOM. All attributes are equally weighted. The used SOM training parameters for the given analysis are as follows:

SOM: 50x50 neurons, hexagonal topology, random initialization,
Euclidean distance measure, neighborhood function = bubble

1st cycle: 100,000 runs, learning rate = 0.05, neighborhood radius (alpha) = 20
2nd cycle: 200,000 runs, learning rate = 0.05, neighborhood radius (alpha) = 15
3rd cycle: 300,000 runs, learning rate = 0.03, neighborhood radius (alpha) = 8

The average quantization error (AQE) after training shows a value of 0.002, which is a good result. The quantization error measures the matching goodness between an input vector and its BMU. In other words, the lower the AQE the better is the match.

6.2.3 Results

The SOM visualizations in Figure 64 give interesting insights into the trained census records codebook file. All input vectors are surprisingly regular spread over the entire SOM without many accumulations. The overall structure in the U-Matrix shows rather small clusters containing a low number of hits. These characteristics indicate a quite balanced dataset with a few strong local relationships. When taking

a closer look into the SOM, as depicted in Figure 65, the view gets more detailed. The lower right corner with significant clusters shows mostly municipalities from the areas around the biggest lakes in Carinthia, namely the Woerthersee, Ossiacher See, and Millstaetter See. The two largest cities Klagenfurt and Villach can be found in this corner as well. They are divided by clusters with average distances but seem to have a lot in common. Hermagor, St. Veit/Glan and Spittal/Drau complete the assumption that large Carinthian cities tend to have strong relationships. Mallnitz is the only exception here, which is totally different than the other municipalities nearby in geographic space.

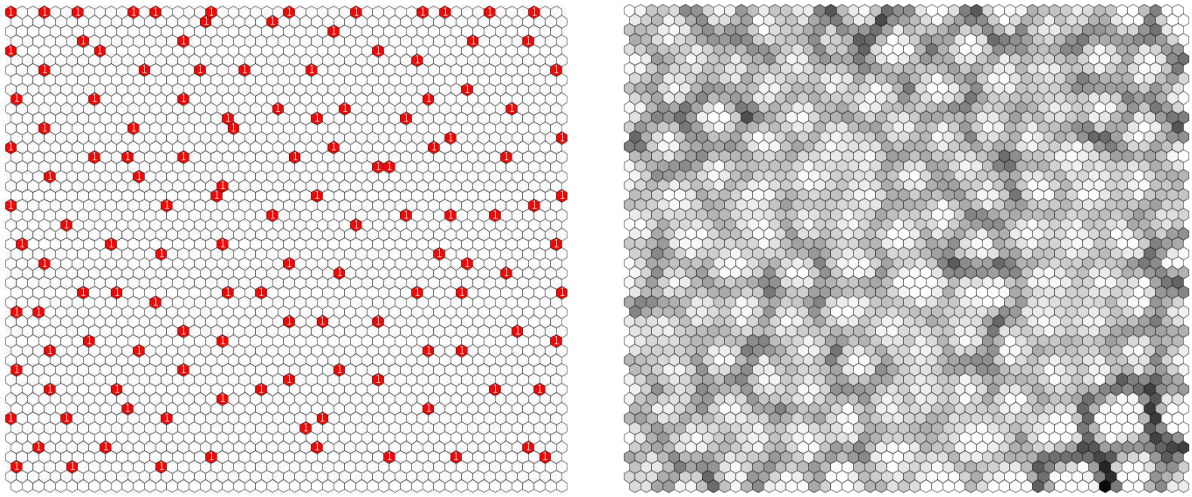


Figure 64: Hit histogram (left) and U-Matrix visualization (right) of the Carinthia census records.

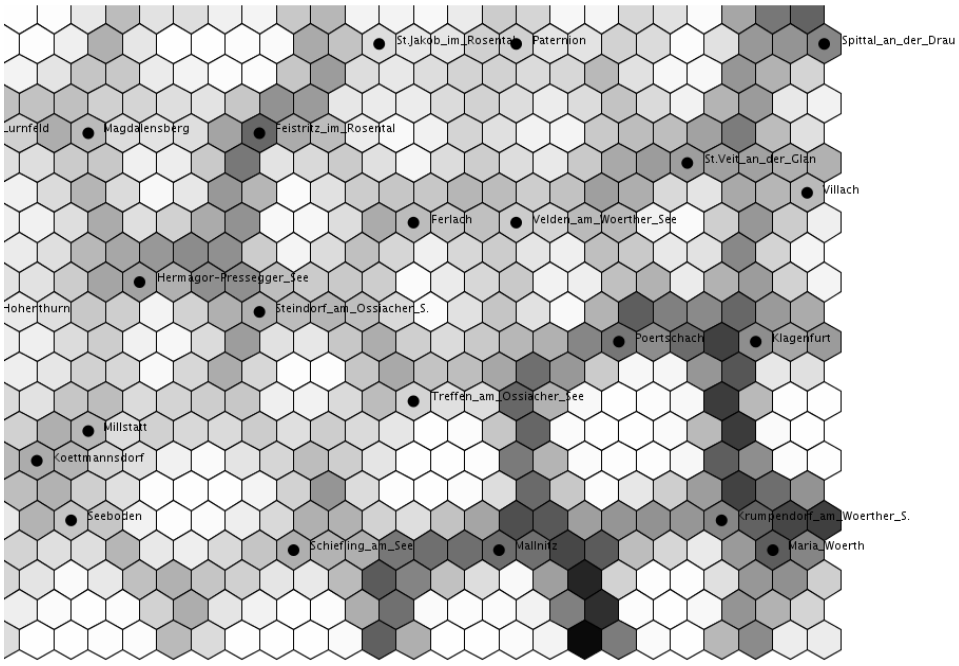


Figure 65: Zoomed view to the lower right corner of the U-Matrix.

To find out what is the reason behind these matching results, it is important to break down the SOM into its single attributes. The component planes are used for that. A side-by-side representation of component planes allows finding similar patterns and from there, conclusions to the hits in the map can be done. As an example, Figure 66 shows twelve component planes, where some of them contain almost identical patterns in the map. The two yellow rectangles mark clusters of high values for average size of household and proportion of unmarried persons which seem to highly correlate. The municipality located in the corresponding area in the SOM is Steuerberg. The same procedure can be used to determine the correlation of single-households, widowed persons, and people older than 60 years, marked with green rectangles. The two almost identical areas refer to Huettenberg, Koetschach-Mauthen as well as Eisenkappel-Vellach. Moreover, high correlation can be found for divorced people and population density in the area around Klagenfurt, highlighted with the two red rectangles. Mallnitz has by far the highest number of people with university degree. This is one remarkable difference to all other municipalities in Carinthia. Krumpendorf, as another example, has the highest proportion of EU citizen. All these attribute values affect the matching position.

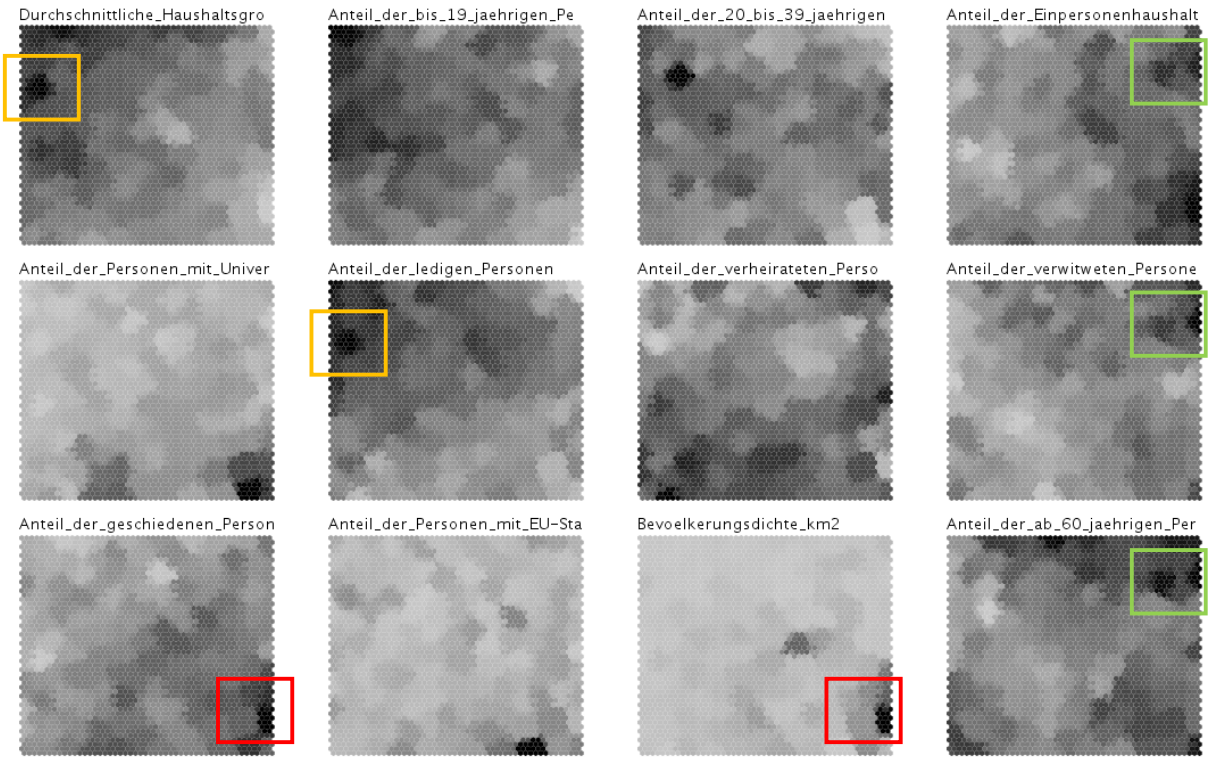


Figure 66: Selected component planes, similar occurrences are marked with rectangles.

To be able to see relationships not only in attribute but also in geographic space, the geographic map visualization can be used. Figure 67 illustrates the color-coded k-Means clustering results linked from the SOM. Five clusters were calculated.

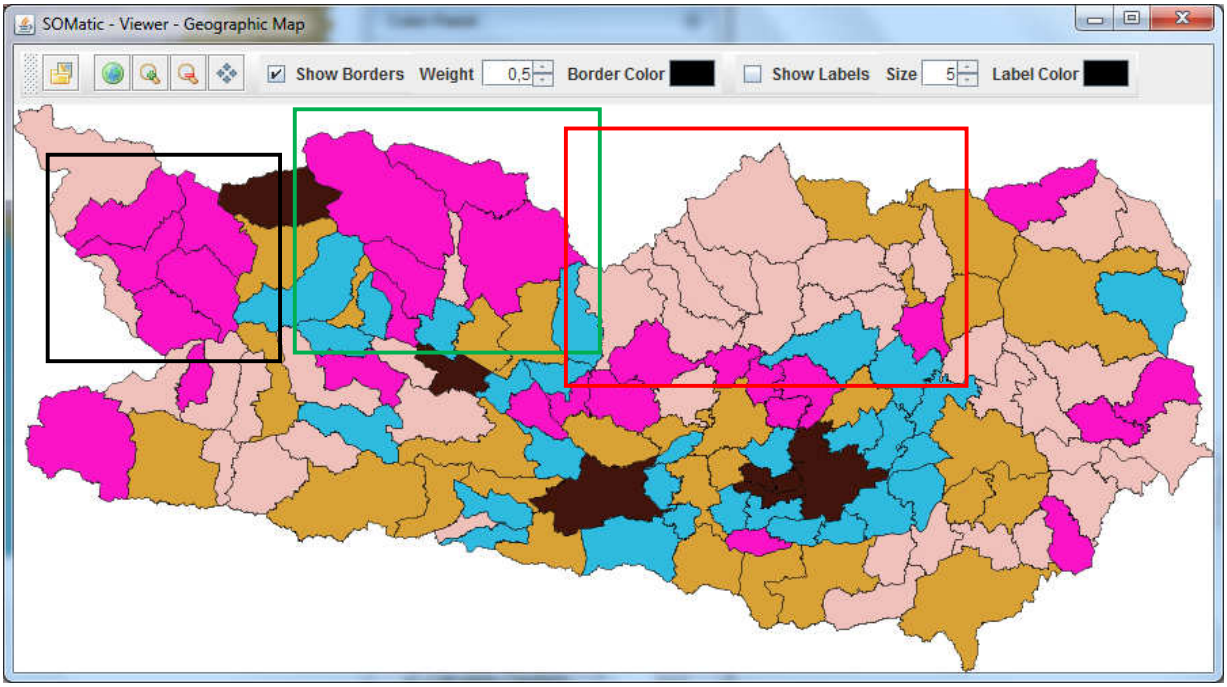


Figure 67: K-Means cluster visualization of Carinthian municipalities linked to the geographic map (k = 5).

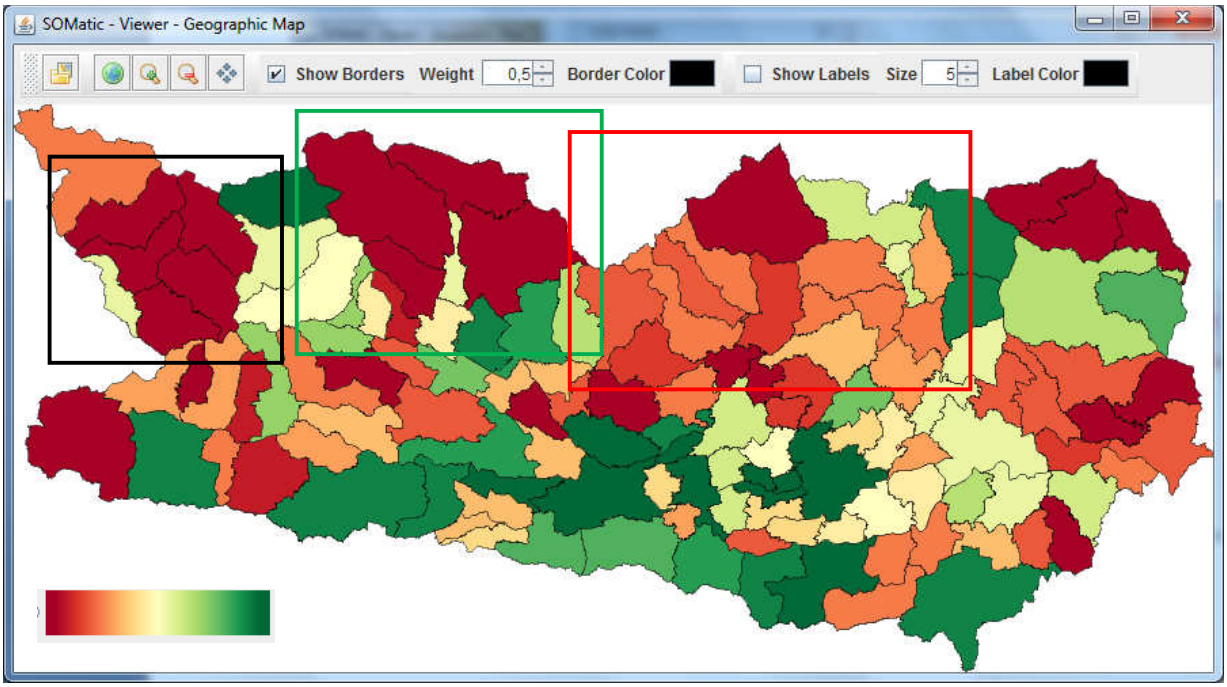


Figure 68: Similarity-based visualization linked to the geographic map using a diverging-diverging color scheme (number of colors in range = 20).

Municipalities in light blue, located around the brownish colored cities of Spittal, Villach, and Klagenfurt tend to be very similar. Municipalities in the Moelltal (black rectangle) and the Liesertal together with the Maltatal (green rectangle) as well as in the Gurktal and Metnitztal (red rectangle) show also strong relationships. When using another visualization method with more accurate measurements, the similarities among municipalities in the red rectangle seem to decrease. Figure 27 shows the results of the similarity-based SOM coloring approach with a range of 20 diverging-diverging colors. While the relationships in the areas with the black and green rectangle still persist is the homogeneous coloring in the former red rectangle disappeared. Nonetheless, the dataset shows a few spatial clusters which can be identified with most of the visualization techniques applied to the SOM. As a conclusion of this analysis, apart from larger cities and their neighboring communes together with municipalities near the biggest lakes in Carinthia which have common relationships, municipalities from rural areas can mostly be found in small clusters all over the map. The U-Matrix in Figure 64 shows best how the input data are distributed within the SOM space. When going through the component planes it turns out that many attributes have no significant patterns. All this leads to such fine grained separation rather than to large clusters.

The analysis could be much more detailed, but this is just be an example to give some helpful information about how to interpret SOM visualization results and how to use the toolset for visual data analysis of highly dimensional real-world data.

6.3 Parallelization

The implementation of threading and design patterns for parallel processing would have been a favorable enhancement of this toolset, but due to the higher priority of other functionalities this was not accomplished. Threading can be implemented for time- and resource-consuming processes in order to improve the performance of the software. Interactive visualizations require real-time frame rates. In some cases the data may be too big for the available memory or CPU (Central Processing Unit), so it is better to increase physical memory size by using shared-memory or in case of multi-core CPUs to partition the data into smaller chunks and to distribute parallel jobs to the different cores. Java provides an extensive range of parallelization methods. As an example, the k-Means clustering could be done in parallel, as described by Chandramohan (2012). The component planes provide another useful integration possibility of parallel processing. Right now the sketch gets slowed down if large SOMs need to be visualized. As *Processing* allows running other threads independently from the main animation thread, this task could be done in parallel, where a proportion of component planes is drawn by a number of available threads. Moreover, after extensive testing it turned out that the BMU search is a major bottleneck with increasing amount of data. Therefore, it is the perfect case for parallelization. This is one of the most required improvements in the future.

7. Conclusion

This is a review of what has been found out and accomplished within this thesis work. The conclusion sums up technical and theoretical insights and allows a brief answering of the research questions.

It has been demonstrated that the implementation of a comprehensive SOM visualization library in *Processing* is possible and benefits from the powerful drawing capabilities of the *Processing* core library. SOMatic Viewer provides an easy to use toolset with interactive visualizations for SOM analysis. Ten different visualizations reaching from visualizing the SOM itself, projecting input data onto the map, to linking from the SOM to geographic space are implemented. These cover three of the four main SOM visualization categories. The PApplet handling as the single parent object which is passed to the rest of the visualization classes brought some challenges. Also, the fact that multiple connected visualization instances need a synchronized access to a central class holding common variables became much clearer at an advanced state of this thesis work. SOMatic Viewer serves programming interfaces for future enhancements, such as adding new coloring or visualization methods. The library works well and shows a consistent performance for small and medium sized SOMs. Large SOMs with the dimension of 100x100 and more, and consisting of hundreds of vector attributes, already show the limits of this toolset, where it takes some time for redrawing the sketch. The final stable release of the library is already in the required structure for getting contributed to the *Processing* community. Creating a Java application was the favored choice after testing SOM sketches running in JavaScript. The library is used as referenced project. Sketches are embedded into SWING panels and loosely coupled to the GUI controls elements. Attribute tables serve as a method for further data exploration. The tables and visualizations are interactively selectable. Further, the GUI provides different file input and output dialogs, preferences can be set, and the visualizations can be controlled and modified in various ways. As the Processing sketches are running in an infinite loop, the application can become resource-consuming quickly. Another issue is that Processing sketches do not start or stop from time to time or freeze after certain events. This needs some further testing and debugging.
The SOM visualization classification, the first of its kind in SOM research, contains four main categories with 23 different visualization types divided into further subgroups. This can be seen as prototypical method as most visualization types vary in their implementation, which may affect the classification results. All in all, it gives a neat overview about the current research state for SOM visualization

techniques, their data exploration capabilities and possible combinations with other representation methods.
The practical example, applying SOMatic Viewer to the real-world data analysis of Carinthian census records, showed that the software already provides a good variety of visualizations for an extensive data analysis. In combination with SOMatic Trainer (Spöcklberger 2013), the toolset has lots of potential for future applications. Besides the common name, both projects also share the same logo, which is illustrated in Figure 69, at the end of the next page.

8. Outlook and Future Work

The SOMatic Viewer tools are already in use; future continuous improvements are planned. As this toolset is the foundation for further implementations it provides a lot of potential, especially for the Processing library. Besides a list of suggestions for improvements, the next possible software evolution steps are given.

SOMatic Viewer works well, but not without occasional errors. As this is the first release, there are many ways to improve the software. Performance reasons are on the one side; algorithmic and functional enhancements are on the other side. Some suggestions for improvements are given below.

SOM Coloring. The coloring of the SOM space can be enhanced by using more advanced algorithms for picking colors from different color spaces. As an example, the current SOM-based coloring uses equidistant steps for color picking, which can be changed by using steps of relative distances between neighboring neurons in the color SOM (see Vesanto 2002).
User Interface. A potential integration of a flip or rotate function of the map would make it easier to compare the SOM with results from other software. It would also solve the issue that the current version of SOMatic starts drawing the first neuron in the upper left corner, which is usually done in the lower left one.
Clustering. Threading can be implemented to speed up the clustering process. Further, there is an algorithm which uses a more sophisticated initial seeding, called k-Means++. It would be a useful replacement to the basic k-Means algorithm. Then, as there is currently no information about the cluster characteristics, this could be integrated. For example, where are cluster boundaries, what makes one cluster different to the other, etc. The integration of hierarchical clustering would be vast enhancement. The user could scroll with a slider through the dendrogram and see cluster hierarchies visualized in the SOM.

Performance. There is an urgent need to find out how the continuously running *Processing* sketches affect the software performance and robustness. Maybe disabling the infinite draw() loop might show some significant results. But then the visualizations require modifications too. As mentioned in section 6.3, the parallelization of certain processes, such as the BMU search or drawing of component planes can significantly speed up the software.
Visualizations. Most of the visualization methods of SOMatic Viewer use only two different colors assigned to the minimum and the maximum value. Colors between these two are interpolated. To see more detailed patterns in the SOM, the color table approach used for coloring the SOM space can also be applied to the other visualizations, maybe with the additional function of replacing different colors within the table. Moreover, the visualizations need a legend, which shows the applied color schema together with its value range. Especially trajectory visualizations would be a useful improvement for the software.
Training Animation. The implemented methods make it easy to integrate a projection of trajectories which draw the path of a component during SOM training.
GeoMap. The geographic map has one issue right now. It does not proportionally resize to the frame dimensions.
SOM Grid Export. There must be a way to create a Shapefile from the SOM grid. This can be elaborated. The current export to image results in a rather low image resolution; an improvement would be to integrate a method for higher image resolution export.
Project File. There is also a lot of potential in the project file approach. Right now the software is using it in a very simplistic way. Based on the fact that it contains variables for both SOMatic Viewer and Trainer, settings such as the normalization parameters can be used for visualizations and analysis techniques.
Input Files. Currently, only one input file can be loaded and visualized. It would be useful to project and compare more input data samples at the same time.

What comes next? There might be the possibility of fully combining the SOMatic Viewer and Trainer projects. This would result in a homogeneous SOM application. As the currently implemented visualizations provide more or less basic algorithms, there might be the possibility to enhance those and integrate other visualization techniques. The GUI in the application would then need more control elements and nested menus. The SOMatic Viewer library might be further developed by *Processing* community members, which is the best case scenario for any open-source software. True to the motto: This is just the beginning…

Figure 69: SOMatic software logo.

References

Literature

Arthur, D., Vassilvitskii, S., 2007. k-means++: The Advantages of Careful Seeding, *Proceedings of the 18th annual ACM-SIAM symposium on Discrete algorithms*, Society for Industrial and Applied Mathematics Philadelphia, 1027-1035.

Burns, R., Skupin, A., 2009. Visualization of Attribute Spaces Involving Places, People and Utterances, *Proceedings of the 24th International Cartographic Conference*, Santiago de Chile.

Bacao, F., Lobo, V., Painho, M., 2005. Self-Organizing Maps as Substitutes for K-Means Clustering, *Proceedings of the ICCS*, 476-483.

Bacao, F., Lobo, V., Painho, M., 2005a. The Self-Organizing Map, the GeoSOM, and relevant variants for geosciences, *Computers and Geosciences*, 31, Elsevier, 155-163.

Bacao, F., Lobo, V., Painho, M., 2005b. Geo-SOM and its Integration with Geographic Information Systems, *Proceedings of the Workshop on Self-Organizing Maps*, 505-512.

Bezdek, J. C., Pal, S. K., 1992. Fuzzy Models for Pattern Recognition: Methods That Search for Structures in Data, *IEEE Press*, New York, 1992.

Chandramohan, A. P., 2012. Parallel K-means Clustering, *Course Presentation at the University of Buffalo*, New York, USA.

CIE, 1986. Colorimetry, 2nd Ed., *CIE Publication No. 15.2*.

Demartines, P., Hérault, J., 1997. Curvilinear component analysis: a self-organizing neural network for nonlinear mapping of data sets, *IEEE Transaction on Neural Networks*, 8 (1), 148–154.

Dykes, J., MacEachren, A., Kraak, M.-J., 2005. Exploring Geovisualization, *Elsevier Science*, Amsterdam, Netherlands.

Fayyad, U., Piatesky-Shapiro, G., Smyth, P., 1996. From Data Mining to Knowlegde Discovery in Databases, *AI Magazine*, American Association of Artificial Intelligence, 37-54.

Gorricha, J., 2009. Visualization of Clusters in Geo-referenced Data Using Three-dimensional Self-Organizing Maps, *Dissertation for the Degree of Master in Statistics and Information Management*, Universidade Nova de Lisboa.

Gorricha, J., Lobo, V., 2012. Improvements on the visualization of clusters in geo-referenced data using Self-Organizing Maps, *Computers and Geosciences*, 43, 177-186.

Guo, D., Gahegan, M., MacEachren, A. M., Zhou, B., 2005. Multivariate Analysis and Geovisualization with an Integrated Geographic Knowlegde Discovery Approach, *Cartography and Geographic Information Science*, 32 (2), 113-132.

Guo. D., Chen, J., MacEachren, A. M., Liao, K., 2006. A Visualization System for Space-Time and Multivariate Patterns (VIS-STAMP), *IEEE Transactions on Visualization and Computer Graphics*, 12 (6), 1461-1474.

Joblove, G. H., Greenberg, D. P., 1978. Color Spaces for Computer Graphics, *Computer Graphics*.

Johnson, R. A., Wichern, D. W., 1992. Applied Multivariate Statistical Analysis, Englewood Cliffs, New Jersey.

Himberg, J., 1998. Enhancing SOM-based data visualization by linking different data projections, *Intelligent Data Engineering and Learning*, Springer, 427-434.

Himberg, J., Ahola, J., Alhoniemi, E., Vesanto, J., Simula, O., 2001. The Self-Organizing Map as a Tool in Knowledge Engineering, *Pattern Recognition in Soft Computing Paradigm*, World Science Publishing Company, Singapore, 38-65.

Kaski, S., Venna, J., Kohonen, T., 2000. Coloring that reveals cluster structures in multivariate data, Australian Journal of Intelligent Information Processing Systems, 82-88.

Keim, D. A., 2002. Information Visualization and Visual Data Mining, *IEEE Transaction on Visualization and Computer Graphics*, 7 (1), 100-107.

Kohonen, T., Hynninen, J., Kangas, J., Laaksonen, J., 1995. SOM Pak – The Self - Organizing Map Program Package, *Document of the Laboratory of Computer and Information Science*, Helsinki University of Technology, Espoo, Finland.

Kohonen, T., 1998. The self-organizing map, *Neurocomputing*, 21, 1-6.

Koua, E. L., 2003. Using self-organizing maps for information visualization and knowledge discovery in complex geospatial datasets, *Proceedings of the 21st International Cartographic Conference*, 1694-1702.

Lacayo, M., Skupin, A. 2007, A GIS-based Visualization Module for Self-Organizing Maps. *Proceedings of 23rd International Cartographic Conference*, Moscow, Russia, CD-ROM.

Latif, K., Mayer, R., 2007. Sky-Metaphor Visualization of Self-Organizing Maps, Proceedings of the 7th International Conference on Knowledge Management, 400-407.

MacEachren, A. M., Kraak, M. J., 2001. Research Challenges in Geovisualization, *Cartography and Geoinformation Science*, 28 (1), 3-12.

Marsland, S., 2003. Novelty Detection in Learning Systems, *Neural Computing Surveys*.

Mayer, R., Aziz, T. A., Rauber, A., 2007. Visualising Class Distribution on Self-Organizing Maps, *Proceedings of the International Concerence on Artificial Neural Networks*, 359-368.

Mayer, R., Rauber, A., 2010. Visualizing Clusters in Self-Organising Maps with Minimum Spanning Trees, *Proceedings of the International Conference on Artificial Neural Networks*, 426-431.

Merkl, D., Rauber, A., 1997. Alternative Ways for Cluster Visualization in Self-Organizing Maps, *Proceedings of the Workshop on Self-Organizing Maps*, 106-111.

Mongini, F., Italiano, M., 2001. TMJ disorders and myogenic facial pain: a discriminative analysis using the McGill Pain Questionnaire, Pain, 91 (3), 323-330.

Neumayer, R., Mayer, R., Pölzlbauer, G., Rauber, A., 2007. The Metro Visualization of Component Planes for Self-Organising Maps, *International Joint Conference on Neural Networks*, 2788-2793.

Pampalk, E., Rauber, A., Merkl, D., 2002. Using Smoothed Data Histograms for Cluster Visualization in Self-Organizing Maps, *Proceedings of the International Conference on Artificial Neural Networks*.

Pölzlbauer, G., Rauber, A., Dittenbacher, M., 2005. Advanced visualization techniques for Self-Organizing Maps with graph-based methods, *Proceedings of the second international symposium on neural networks*, 75-80.

Pölzlbauer, G., Dittenbach, M., Rauber, A., 2006. Advanced visualization of Self-Organizing Maps with vector fields, *Neural Network Science*, 19, 911-922.

Sammon, J. W., 1969. A nonlinear mapping for data structure analysis, *IEEE Transactions on Computers*, C-18 (5), 401-409.

Schmidt, C. R., 2008. Effects of Irregular Topology in Spherical Self-Organizing Maps, *Thesis for the Degree Master of Science in Geography*, San Diego State University, San Diego, USA.

Shepard, R. N., Carroll, J. D., 1965. Parametric representation of nonlinear data structures, *Proceedings of the International Symposium for Multivariate Analysis*, 561–592.

Skupin, A., 2003. A Novel Map Projection Using An Artificial Neural Network, *Proceedings of the 21st International Cartographic Conference*, 1165-1173.

Skupin, A., Fabrikant, S. I., 2003. Spatialization Methods: A Cartographic Research Agenda for Non-geographic Information Visualization, *Cartography and Geographic Information Science*, 30 (2), 95-115.

Skupin, A., Hagelman, R., 2005. Visualizing Demographic Trajectories with Self-Organizing Maps, *GeoInformatica*, 9 (2), 159-179.

Skupin, A., Fabrikant, S. I., 2007. Spatialization, *The Handbook of Geographical Information Science*, Blackwell Publishing, London, Great Britain, 61-79.

Skupin, A., Agarwal, P., 2008. Introduction: What is a self-organizing map? *Self-organising maps: Applications in geographic information science*. Wiley, Chapter 1, 1–20.

Skupin, A., Esperbé, A., 2008. Towards High-Resolution Self-Organizing Maps of Geographic Features, *Geographic Visualization: Concepts, Tools and Applications*, Wiley, Chapter 2, 159-181.

Skupin, A., 2010. Tri-Space: Conceptualization, Transformation, Visualization. *Sixth International Conference on Geographic Information Science (GIScience 2006)*, Zürich, Switzerland.

Skupin, A., Esperbé, A., 2011. An alternative map of the United States based on an n-dimensional model of geographic space, *Journal of Visualization Language in Computing*, 22, 290-304.

Spöcklberger, M., 2013. SOMatic Trainer: Implementation of a Self-Organizing Map Tool with Parallel Training for Processing applied to Carinthian Municipality Census Data, *Thesis for the Degree Master of Science in Engineering,* Carinthia University of Applied Sciences, Villach, Austria.

Takatsuka, M., Gahegan, M., 2002. GeoVISTA Studio: A Codeless Visual Programming Environment For Geoscientific Data Analysis And Visualization, *Computers and Geosciences*, 28, 1131-1141.

Takatsuka, M., 2001. An application of the self-organizing map and interactive 3-D visualization to geospatial data, *GeoComputation '01—Sixth International Conference on Geocomputation*, Brisbane, Australia, CD-ROM.

Tobler, W., 1970. A computer movie simulating urban growth in the Detroit region, *Economic Geography*, 46 (2), 234-240.

Ultsch, A., 2003. *Maps for the* Visualization of high-dimensional Data Spaces, *Proceedings of the Workshop on Self Organizing Maps*, 225-230.

Ultsch, A., 2004. U*-Matrix: a Tool to visualize Clusters in high dimensional Data, *Technical Report 36*, CS Department, Philipps-University Marburg, Germany.

Ultsch, A., Mörchen, F., 2005. ESOM-Maps: tools for clustering, visualization, and classification with Emergent SOM, *Technical Report of the Department of Mathematics and Computer Science*, University of Marburg, Germany, 1-7.

Vesanto, J., Himberg, J., Siponen, M., Simula, O., 1998. Enhancing SOM based data visualization, *Proceedings of the 5th International Conference on Soft Computing and Information/Intelligent Systems*, Methodologies for the Conception, Design and Application of Soft Computing, 1, 64-67.

Vesanto, J., Himberg, J., Alhoniemi, E., Parhankangas, J., 1999. Self-organizing map in MATLAB: the SOM Toolbox, *Proceedings of the MATLAB DSP Conference*, Espoo, Finland, 35-40.

Vesanto, J., 1999. SOM-based data visualization methods, *Intelligent Data Analysis*, 3 (2), 111-126.

Vesanto, J., 2002. Data Exploration Process Based on the Self-Organizing Map, *Dissertation for the Degree of Doctor of Technology*, Acta Polytechnica Scandinavia, Mathematics and Computing Series No. 115.

Vlissides, J., Helm, R., Johnson, R., Gamma, E., 1995. Design patterns: Elements of reusable object-oriented software, Addison-Wesley, Book.

Wu, S., Chow, T. W. S., 2005. PRSOM: A New Visualization Method by Hybridizing Multidimensional Scaling and Self-Organizing Map, *IEEE Transactions on Neural Networks*, 16 (6), 1362-1380.

Wu, Y., Takatsuka, M., 2006. Spherical self-organizing map using efficient indexed geodesic data structure, *Neural Networks*, 19, Elsevier, 900-910.

Yin, H., 2002. A Novel Method for Multivariate Data Projection and Structure Visualization, *IEEE Transactions on Neural Networks*, 13 (1), 237-243.

Online Literature

ICA, 2013. International Cartographic Association, Available from: http://icaci.org [Accessed 8 July 2013].

Databionic, 2006. Databionic ESOM Tools, Available from: http://databionic-esom.sourceforge.net [Accessed 2 February 2013].

Eclipse, 2013. Eclipse – The Eclipse Foundation open source community website, Available from: http://www.eclipse.org [Accessed 27 February 2013].

Guo, D., 2013. SOMVis: Multivariate Mapping and Visualization, Available from: http://www.spatialdatamining.org/software/somvis [Accessed 12 May 2013].

OpenProcessing, 2013. OpenProcessing, Available from: http://openprocessing.org [Accessed 3 February 2013].

Oracle, 2013. Java SE Overview - at a Glace, Available from: http://www.oracle.com/technetwork/java/javase/overview/index.html [Accessed 30 March 2013].

OSGeo Project, 2013. GeoTools The Open Source GIS Java Toolkit, Available from: http://www.geotools.org [Accessed 13 May 2013].

PennState, 2013, GeoVISTA Studio Project, Available from: http://www.geovistastudio.psu.edu [Accessed 28 July 2013].

Reades, J., 2013. The MapThing Processing Library, Available from: http://www.reades.com/2013/04/01/the-mapthing-processing-library/ [Accessed 26 June 2013].

Processing, 2013. Processing 2.0. Available from: http://processing.org [Accessed 1 February 2013].

TU Vienna, 2013. Data Mining with the Java SOMToolbox, Institute of Software Technology and Interactive Systems, Vienna University of Technology, Austria. Available from: http://www.ifs.tuwien.ac.at/dm/somtoolbox/index.html [Accessed 2 February 2013].

Viscovery, 2013. Viscovery SOMine 5.2. Viscovery Software GmbH, Vienna, Austria. Available from: http://www.viscovery.net/somine/ [Accessed 31 January 2013].

Wikipedia, 2013. HSL and HSV, Wikipedia – The Free Enzyklopedia, http://en.wikipedia.org/wiki/HSV_color_space [Accessed 14 June 2013].

List of Abbreviations

AAG	*Association of American Geographers*
AQE	*Average Quantization Error*
ANN	*Artificial Neural Network*
API	*Application Programming Interface*
BMU	*Best Matching Unit*
CCA	*Curvilinear Component Analysis*
CIE	*Commission international de l'éclairage*
CPU	*Central Processing Unit*
GIS	*Geographic Information System/Science*
GUI	*Graphical User Interface*
HSB	*Hue Saturation Brightness*
IDE	*Integrated Development Environment*
JAR	*Java Archive File*
KDD	*Knowledge Discovery in Databases*
LVQ	*Learning Vector Quantization*
MDS	*Multi Dimensional Scaling*
MST	*Minimum Spanning Tree*
PCA	*Principle Component Analysis*
PCP	*Parallel Coordinate Plot*
PDE	*Processing Development Environment*
PS	*PostScript*
RGB	*Red Green Blue*
SDH	*Smoothed Data Histogram*
SHP	*Shapefile*
SOFM	*Self-Organizing Feature Map*
SOM	*Self-Organizing Map*
UI	*User Interface*
UML	*Unified Modeling Language*
US	*United States*

List of Figures

List of Tables