MASTERARBEIT

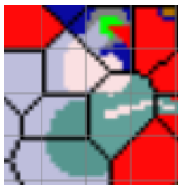
Coloring of the Self-Organising Maps based on class labels

ausgeführt am Institut für
Softwaretechnik
der Technischen Universität Wien

unter Anleitung von
Ao. Univ. Prof. Andreas Rauber

durch

Taha Abdel Aziz
Quellenstrasse 24B/13/13
A–1100 Wien



Wien, am 3. Jänner 2016 Unterschrift

Kurzfassung

Die Selbstorganisierende Karte (SOM) ist ein nützliches und starkes Werkzeug für die Datenanalyse, besonders für große Datensätze oder Datensätze von hoher Dimensionalität. SOM Visualisierungen bilden die Dimensionen des Datenmodells auf graphische Dimensionen wie Farbe und Position ab, so helfen sie der Navigation und dem Erforschen von dem SOM. SOM Visualisierungen können auch die Daten selbst einbeziehen, so dass der Zugriff auf Informationen möglich wird, die in einem reinen SOM nicht verfügbar sind. Dadurch wird ein tieferer Einblick in die Daten möglich. Wenn die Daten mit klassen gekennzeichnet sind, können diese Klassen auch in der Visualisierung einbezogen werden, so dass, eine klarere Idee über die Klassinformation gewonnen wird. In dieser Arbeit schlagen wir eine neuartige SOM Visualisierungsmethode, nämlich die SOM Klassenfärbung vor, welche auf den Datenklassen beruht. Diese Methode findet eine farbige Partition des SOM-Gitters, die die Klassenstruktur widerspiegelt. Diese Visualisierung ermöglicht das entdecken von Klassinformation wie Klassenstruktur, Klassenverteilung und Klassenclusters. Außerdem können neue Daten Klassen zugeordnet werden und zwar indem der Punkt auf dem SOM-Gitter ermittelt wird, welcher das neue Datum (Messwert) am besten repräsentiert; das neue Datum wird dann jener Klasse zugeordnet, die die Partition repräsentiert, auf der sich der Punkt befindet.

Abstract

The *Self-Organizing Map* (SOM) is a useful and strong tool for data analysis, especially for large data sets or data sets of high dimensionality. SOM visualizations map the data model dimensions to visual dimensions like color and position, thus they help exploring the SOM. Visualization can also involve the data itself so that it helps accessing information that are not available in the trained SOM, thereby enabling a deeper look inside the data. If the data comes with supervised class labels, these labels can be also involved in the visualization, thus enabling the user to have a clearer idea about the data and the structures learned by the SOM. In this work we propose a novel SOM visualization method, namely the SOM class coloring, which is based on the data class labels. This method finds a

colored partitioning of the SOM lattice, that reflects the class distribution. SOM class coloring helps discovering class information such as class topology, class clusters, and class distribution. Furthermore class labels can be assigned to new data items by estimating the point on the lattice, that best represents the data item and then assigning the class of the partition that includes this point to the data item.

Contents

List of Figures

Chapter 1

Introduction

Analyzing data usually needs more than getting pure statistical properties of the data set. There are many methods for quickly producing overall summaries of a data set. For example, five-number summary consisting of some statistical values (greatest, median, lower quartile and upper quartile) provides a help understanding simple data sets of low dimensionality and limited volume [Kas97]. Data structure and topology such as finding the clusters and class distribution of the data is very essential. The more data there is available the more difficult it is to understand this data set. Also, the dimensionality of the feature space representing the data is a factor. For this problem there is an essential need of methods for data exploration and especially the methods, that can discover and illustrate effectively the structure of the data. Data mining is one of the fields where such exploratory methods are needed in the form of tools in Knowledge discovery in database (KDD), whose purpose is to find new knowledge from databases where dimensionality, complexity, or amount of data is too large or complex for pure human observation to be studied. Data mining is now an important area of research, responding to the presence of large databases in commerce, industry, and research. Methods of data exploration vary depending on the nature of the data and the goal of study: clustering methods are the more conventional ones; they aim to organize the data patterns into groups, so that patterns in one group are more similar to each other than to those in another group. In other words clustering tries to reduce the amount of data items by grouping them. Clustering is useful in several areas such as pattern analysis, grouping, and machine-learning,

document retrieval, image segmentation, and many other applications [AJ99]. Projection methods try to reduce the dimensionality of the data: they map the multidimensional data features to low dimensional space (usually 2D or 3D) that represents the data and can be easier visualized and studied. Of course this mapping should preserve the original data topology as much as possible. If the goal of projection is to visualize the data, then low output dimensionality should be chosen in order to achieve meaningful visualizations. Visualization methods aim to map the dimensions of the data to visual dimensions such as position, color, etc. in a way that the observer can get a deeper insight in the structure of the data. Of course the visual dimensions are limited, so if the data is of high dimensionality, direct visualization will be difficult, not helpful or impossible [Ves99]. In such a case visualization should make use of the two previous methods (Clustering and projection) as a pre-processing step to prepare the data in a form that it can be efficiently visualized. Self-Organizing Maps (SOMs) [Koh97] are efficient tools that implicitly combine all of the previous methods. A SOM is a neural network consisting of low dimensional grid (mostly 2D) of nodes that are trained with high-dimensional data. The nodes on the trained map represent the original data in a manner that similar data points in the feature space are mapped to nodes which are close to each other on the map. SOMs are strong tools used in the data exploration. "*Some properties that distinguish SOM from the other data mining tools are that it is numerical instead of symbolic, nonparametric, and capable of learning without supervision*" [Kas97]; Many variants and extensions of the standard SOM were proposed which try to cover some drawbacks of the standard SOM. In practical applications the process of deciding which method to use is essential: this depends on the nature of the data and the goal of exploration; the following questions are in any case to be resolved: What kind of structure the method can extract?; How does it illustrate this structure?; Does the method reduce the data dimensionality and/or the number of data items?; Which and if so which type of data pre-processing is necessary?. The focus of this work is visualization of the SOM, in particular SOMs that are trained with labeled data. It is meaningful here to distinguish between two things that are of great importance in this work: SOM is based on an unsupervised learning process in the meaning that no prior class information is needed to perform the training,

because the learning is data-driven. Regardless of this fact, class information can be used in the form of labels which can be then used for exploration and visualization purposes; such labels can be collected in a pre-processing step, or even as a post-processing step. These labels do not affect the training process and the resulting structure or topology, but they help in generating visualizations displayed on top of the finished map. In particular we propose in this work a visualization method of SOM which is based on such class labels namely a SOM coloring that depends on these labels.

The remainder of this thesis is structured as follows: In Chapter 2 we will discuss some related concepts and methods such as the concept of the *Self-Organizing Map*, some clustering and classification methods and methods for projection and visualization of the trained SOM. In Chapter 3 we present our novel method for coloring the SOM based on class labels. In Chapter 4 we test and evaluate our method with some collected data sets. Finally we present our conclusions in Chapter 5.

Chapter 2

Related Work

In this chapter we will introduce some of the concepts and worksljubíme that have been proposed which are closely related with the underlying issue, coloring of the SOM. In Section 2.1 the general issue of the SOM will be introduced. In Section 2.2 and 2.3 we will give some idea about the two processes, which are closely related to visualization: these are clustering and projection. In Section 2.4 visualization will be discussed briefly especially in respect to the SOM, Some SOM-based clustering, projection and visualization methods will be also discussed.

2.1 Self organizing map

The *Self-Organizing Map* (SOM) in the basic form is an artificial neural network model. The nodes of this network are trained to various input patterns from the feature space. SOMs use an unsupervised learning process: no a priori classification for the input is needed. The result of the learning process is a map lattice (in this work we assume a 2 dimensional lattice, which is usually the case), that map the high-dimensional feature space of the input in a way, so that similar inputs in the feature space have associated nodes that are close to each other on the map lattice, thus reducing the high-dimensionality of the feature space. This projection is capable (a) to help clustering the data and (b) to approximately preserve the original data topology of the input. SOMs are therefore especially useful for data visualization and exploration purposes. I a classic SOM,

each neuron represents an n-dimensional column vector , where n depends on the data space dimensionality (input vectors dimensionality). The reason for using one- and two-dimensional grids is that space structures of higher dimensionality cause problems with data display. Neurons are usually located in the nodes of the two-dimensional grid with rectangular or hexagonal cells. The number of neurons in the lattice determines the resulting resolution and the granularity of data presentation.

During the training not only the winning neuron is modified but its neighbors as well, although the strength of the adaption may depend on their distance from the winner neuron. This allows considering the SOM to be a method of projecting the data nonlinearly onto a lower-dimensional display. When using this algorithm the vectors similar in the initial space get close to each other on the final map SOMs [Koh97].

2.1.1 The map initialization

If the map contains tens of thousands of neurons then it usually takes too much time to train the system for the practical tasks. Thus making choice of the nodes quantity requires a reasonable trade-off. Before training the map it is necessary to initialize the weight coefficients of the neurons. The learning rate factor can be significantly increased by an appropriate initialization method, thus bringing better results. In general, there are three methods of weights initialization:

- Initialization with random values. All the weights are assigned small random values.
- Initialization with patterns. Initial values are set to randomly chosen patterns of the training sample.
- PCA-based initialization. The weights are initialized by values of the vectors ordered along a two-dimensional subspace spanned by the two principal eigenvectors of the input data vectors.

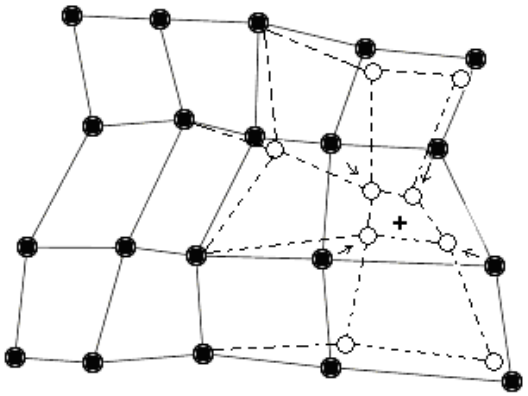


Figure 2.1: Adapting the *best-matching unit* and its neighbors weights The input vector coordinates are marked with a cross, coordinates of the map nodes before modification are shown as full circles and after modification as empty circles.

2.1.2 The training process

The learning process consists of sequential corrections of the vectors representing neurons. On every step of the learning process a random vector is chosen from the initial data set and then the best-matching neuron coefficient vector (up now *BMU* or *best-matching unit*) is identified. This is the most similar unit to the input vector. The word 'similarity' in this task means the distance between the vectors. There are various distance metrics that can be used for this purpose. The most common one is the Euclidean metric, where the the winner unit must meet the following relation:

$$\|X - W_c\| = min_i \|X - W_i\| \tag{2.1}$$

Where W_c is the winner vector at index c, X is the input vector and W_i is any Unit vector in the SOM. After the *BMU* is found the neural network weights are adapted. The winning unit and its neighbors adapt to represent the input by modifying their reference vectors towards the current input. Figure 2.1 shows how this works with a 2D vector. The weight modification is defined by the following formula:

$$W_i(t+1) = W_i(t) + h_{ci}(t) * [X(t) - W(t)] \tag{2.2}$$

where t is the discrete time index. Vector $X(t)$ is a randomly selected sample at iteration t. The function $h(t)$ is the neighborhood function:

$$h(t) = h(\|r_c - r_i\|, t) * \alpha(t) \tag{2.3}$$

It represents a decreasing function of time and distance between the winning neuron and its neighbors on the grid. The function consists of two parts: the distance function and the learning rate function of time. where r determines the position of the neuron on the 2D grid. A simple version for h is the box-neighborhood. Usually one of two functions of distance is used, simple constant:

$$h(d,t) = \begin{cases} const & \text{for} \quad d \leq \sigma(t) \\ 0 & \text{for} \quad d > \sigma(t) \end{cases} \tag{2.4}$$

or Gaussian function:

$$h(d,t) = e^{\frac{d^2}{2\sigma^2(t)}} \tag{2.5}$$

Where $\sigma(t)$ is a diminishing function of time. This value is often called the radius of the neighborhood. At the beginning of the learning procedure it is fairly large, but it is made to gradually shrink during learning. Towards the end a single winning neuron is trained. Most frequently the linear decreasing function of time is used.

Let us proceed to the learning rate function $\alpha(t)$. This is also a decreasing function of time. A variant of this function are linear and inversely proportional of time with the formula:

$$\alpha(t) = \frac{A}{t + B} \tag{2.6}$$

where A and B are constants. There are two main phases in the learning process. At the beginning the learning rate and the neighborhood radius are fairly large, which allows ordering the neurons vectors according to the sample patterns distribution. Then the weights are accurately adjusted with the learning rate parameters much smaller than they initially were. In case of using PCA-based initialization the first step of rough adjustment can be skipped.

The previous training algorithm, which was described formally is now summarized in the following steps:

1. Initialize the weights of each node .

2. Chose a sample vector randomly from the set of training data.

3. Every node is examined to find the *best-matching unit*.

4. The radius of the neighborhood of the *BMU* is now calculated. This is a value that starts large, typically set to the 'radius' of the lattice, but diminishes each time-step. Any nodes found within this radius are deemed to be inside the *BMU*'s neighborhood.

5. Each of neighboring node's weights are adjusted to make them more like the input vector. The closer a node is to the BMU, the more its weights get altered.

6. Iterate from step 2 for N iterations.

2.2 Clustering and classification

Cluster analysis is the organization of a collection of patterns into clusters based on similarity among these patterns. That is patterns belonging to a cluster are (more) similar to each other to than those in another clusters. There are many clustering methods, however it is useful at this point to distinguish between supervised and unsupervised methods: in the case of supervised classification (discriminant analysis) a collection of already classified patterns is provided and the problem is to classify a new pattern. In the case of unsupervised classification (clustering) a collection of unclassified pattern is provided and the problem is to group this collection into meaningful clusters, but this process of labeling is data driven and not as according to a given class structure as in the case of supervised classification. The clustering process typically consist of the following steps: Pattern representation, pattern measure definition, grouping, data abstraction.

1. Pattern representation: this step involves preparing the input data and finding important information such as the number of available patterns and the number, type, and scale of the features involved in the clustering. Furthermore a subset of features may be selected to be the most important subset of the original feature set. This process is called feature selection.

If the features are in form that is not suitable for clustering, one or more transformation could be used to put the features in an appropriate form to start clustering.

2. Pattern proximity measure definition: this is the distance function, that is applied on pairs of patterns to measure the similarity between the two patterns. There is a variety of distance measure functions. The simplest one may be the Euclidean distance.

3. Grouping: this is the process of partitioning of the date into groups. The partitions can be hard, where a pattern can belong to only one partition and it can be fuzzy, where a pattern can have membership degrees in many clusters. Furthermore there are two conceptual clustering algorithms: hierarchical algorithms, that produce many nested partitions based on similarity according to a criterion for merging or splitting clusters and partitive clustering algorithms, that identify the partition that optimizes a clustering criterion. Clustering algorithms will be discussed in the next sub sections in more details.

4. Data abstraction: is the process of extracting a simple and compact representation of a data set. In the clustering context, a typical data abstraction is a compact description of each cluster, usually in terms of cluster prototypes.

To have optimal clustering, there are many parameters to be tuned. This depends also on the purpose of the clustering. We note, that the purpose is usually not to find an optimal clustering of the data, but to have a clustering that enables an optimal and efficient exploring the data considering speed, robustness and the ability of visualization [AJ99].

2.2.1 Hierarchical algorithms

Hierarchical clustering approaches can be divided into agglomerative and divisive algorithms, corresponding to bottom-up and top-down strategies, to build a hierarchical clustering tree. Agglomerative algorithms are more commonly used than the divisive methods and usually have the following steps:

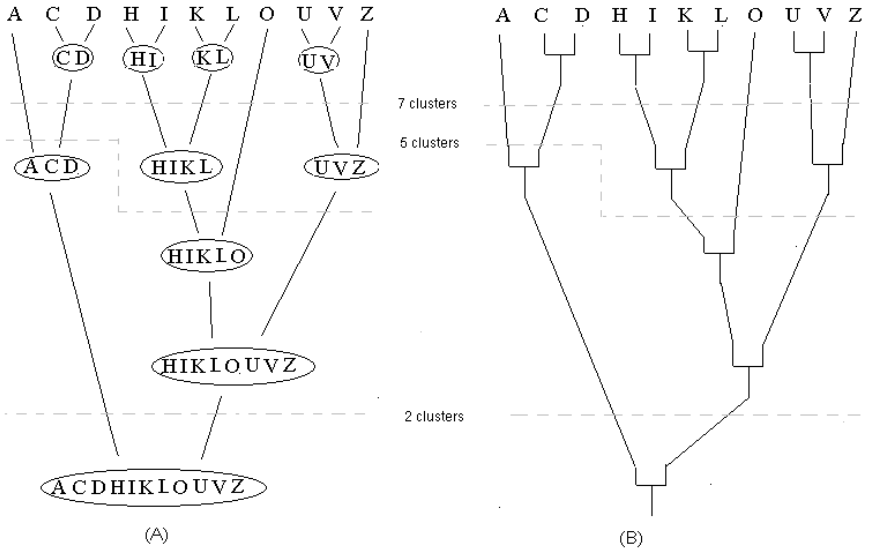


Figure 2.2: Agglomerative algorithm: (A) steps, (B) corresponding dendrogram

1. Initialize: Assign each vector to its own cluster.
2. Compute distances between all clusters.
3. Merge the two clusters that are closest to each other.
4. Return to step 2 until there is only one cluster left.

That means, that the algorithm begins with each element in its own cluster and the clusters are merged together iteratively according to the similarity between them. The process continue until all elements are in one cluster, see Figure 2.2-A. A tree structure is provided as a result of this algorithm, which can represented with a tree diagram called dendrogram, depicted in Figure 2.2 -B. The dendrodiagram can be cut in any level providing a different clustering [VA00].

2.2.2 Partitive algorithms

In this approach the algorithm identifies the partition by minimizing some error function. The number of clusters is either already predefined or determined by the algorithm by trying various numbers of clusters. In general partitive algorithms consist of the following steps:

1. Determine the number of clusters.
2. Initialize the cluster centers.
3. Compute partitioning for data.
4. Compute (update) cluster centers.
5. If the partitioning is unchanged (converged), stop; otherwise, return to step 3.

The most common partitive clustering algorithm is the k-means algorithm, which partitions N data points into K disjoint subsets S_j containing N_j data points. The algorithm minimizes the sum-of-squares criterion E:

$$E = \Sigma_{j=1}^{K} \Sigma_{n \in S_j} ||x_n - c_j||^2 \tag{2.7}$$

where x_n is a vector representing the nth data point and c_j is the geometric centroid of the data points in S_j.

2.2.3 Two-level clustering approach / Clustering of the SOM

There are two main problems in the standard clustering algorithms mentioned in the previous sub sections above: the first problem is the high computational costs when applying the algorithm directly on the original data, especially if a large collection of patterns underlies. The second problem is the sensibility to noise: applying the algorithm to original data makes it sensible to the occurrence of noise and outlier because this could make clusters overlap or have inaccurate boundaries. Both of these problems can be solved by using the two-level clustering approach: The clustering algorithm is not applied directly on the original data but rather on cluster prototypes, that are generated by a previously trained SOM. SOM nodes are considered as cluster prototypes and are grouped into clusters by applying a suitable clustering algorithm. The number of the cluster prototypes (nodes) must be much larger than the expected number of clusters but much smaller than the number of the data patterns. The computational costs is dramatically reduced especially in case of hierarchical algorithms. This

is because the SOM algorithm can be applied to large data sets and the computational complexity is linearly proportional with the number of input patterns, but on the other hand, the complexity scales quadratively with the number of map nodes (prototypes). However this overhead can be solved with special optimization techniques. The problem of noise and outliers is also solved because the prototypes are the local averages of the original data produced by training the SOM und thus less sensible to outliers and nodes.

2.3 Projection

The structure of the data cloud formed by prototype vectors is the key for understanding the data by giving answers to questions like what and where are the clusters, which shape has the data cloud, etc. Projection methods are very useful in answering such questions. Projecting data on to its underlying subspace can detect its real structures, facilitate functional analysis, and help making a judgment. Principally there are two types of projection methods: linear methods such as principal component analysis (PCA), and nonlinear such as multidimensional scaling (MDS) and SOM.

PCA is a linear transformation that transforms the data to a new coordinate system such that the greatest variance by any projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on. PCA can be used for dimensionality reduction in a dataset while retaining those characteristics of the dataset that contribute most to its variance, by keeping lower-order principal components and ignoring higher-order ones. PCA is an optimal linear projection algorithms, because it has the minimum mean-square-error values between the original data vectors and the points in the projection, thereby it achieves the equation 2.8:

$$\min \Sigma_{i=1}^{T}[x_i - \Sigma_{j=1}^{m}(q_j^T x_i)q_j]^2 \tag{2.8}$$

Where $X = [x_1, x_2, .., x_n]^T$ is the n-dimensional input vector. $\{q_j, j = 1, 2, ...m, m \leq n\}$ are orthogonal vectors representing principal directions. The term q_j^T represents the projection of x onto the j-th principal dimension.

PCA has limited power to project data of high dimensionality, and thus it

has limited power for capturing nonlinear relationships in practical data. For this reason extensions to nonlinear PCA are used such as Generalized PCA, Kernel PCA, and principal curves and surfaces[Yin03].

Multidimensional scaling (MDS) is a nonlinear projection method, that tries to project points of data onto a two-dimensional plot, thereby preserving as close as possible the inter-point metrics. For this purpose arbitrary optimization algorithms can be used, MDS does not specify that a certain algorithm to be used. In the optimization process local minima and divergence problems may occur. Furthermore, this process incurs high commotional costs. These costs depend on the selected algorithm and the starting configuration. The overall structure of the data is maintained thereby.

2.3.1 Sammon's Mapping

Sammon's mapping is a common example of MDS projection. This algorithm aims to minimize the difference between inter-point distances in the original data and in the projection. The Sammon's mapping algorithm minimizes the stress function 2.9.

$$S_{sammon} = \frac{1}{\Sigma_{i<j}^{n} d_{ij}} \Sigma_{i<j}^{n} \frac{[d_{ij} - D_{ij}]^2}{d_{ij}} \tag{2.9}$$

where X_i and X_j are two points in the input space and Y_i and Y_j are their projection points respectively, d_{ij} is the Euclidean distances between X_i and X_j, D_{ij} is the Euclidean distance between Y_i and Y_j, and n is the number of patterns.

It performs a recursive learning algorithm using the Newton optimization to achieve the optimal configuration [Sam69]. Although this algorithm converges relatively fast, it has the drawback of high computational costs and higher risk of sub-optimal solution due to local minima. Furthermore Sammon mapping is a point-to-point mapping which means that all the inter-point distances must be explicitly calculated for every data set. In the broader since this means that there is no explicit mapping function. Accommodation of new data point needs recalculation of all points, which is a serious problem for applications that deal with sequential data [Yin03].

2.4 Visualizations

Data visualization is an important process in data mining. Generally, to visualize multi dimensional data, two steps are needed: the first is vector quantization, where the original data set is reduced to a smaller one, that still represents the original, suppresses noise and is easier to work with than the original set. The second is a vector projection to a low dimensionality (usually up to 3D). Of course in our context this projection must still represent the original vector set in terms of distances between vectors or at least topological ordering. Visualization is the process of projection the real dimensions to visual dimensions. Typical dimensions in this sense include positions, size, color and text labels. The number of visual dimensions available is essential: If there are many features involved in the visualization, it may be impossible to show them on one map and this leads to the issue of multiple visualization maps. It is an issue to visualize the data in a form, that it is easy to recognize corresponding areas on different maps. This process is called linking. Linking should be obvious to the observer. For example in case of SOM the linking dimension is the position. The same object in different map visualizations correspondents to the same position on the map. There are various algorithms to do these steps. In this and the following subsections a selection of visualization methods will be presented. A special case of visualization is the visualization achieved with SOM. SOM combine both of the two steps.

2.4.1 Visualization of SOM with Unlabeled data

Visualization of complex data with high dimensions and large number of items is one of the most important applications of SOM. Visualization with SOM is a special case of visualization in term of implicitly combining vector quantization and projection, while in other methods projection is completely subordinate to quantization. Another deference is that the projection grid of the SOM is regularly shaped: this makes it very easy to compare or link different visualizations. The map resulting from a trained SOM can itself help as an exploration tool; thereby similar data points are mapped to the same unit or to points that are placed near to each other; the labels mapped to these units on the map can be explored in a table-like fashion [RPM03]. Depending on the goal of visualiza-

tion, there are three categories of visualization methods: the first is getting an abstract idea about the data including the overall shape and cluster structure. The second category aims at the analysis of prototype vectors and characteristic of clusters. The third one has the purpose of examining and classification of new data. There also two categories of visualizations according to the source of information used: visualizations that shows the map in relation to the data set such as hit histograms and smoothed hit histograms, P-Matrix, etc., and visualizations that are derived from the model vectors which aim at showing cluster structure and boundaries such as U-Matrix, component planes visualizations and clustering of the SOM. These and other visualization methods will be discussed in the following subsections [PDR].

U-Matrix, P-Matrix and U*-Matrix

U-Matrix [Ultsch1992] is a distance-based visualization method that aims to visualize the SOM by displaying the local distance structure on top of the SOM lattice; at each unit of the map the local distance to its neighbors is visualized as a height. This height is defined as the sum of distances to all immediate neighbors normalized by the largest height. This results in mountain ranges as boundaries for data clusters. This method produces good visualization if the data has clearly separated clusters, but problems can occur if the data has overlapping clusters or even slowly changing densities. P-Matrix [Ult03a, Ult05] is a density-based visualization method in which the density distribution of the data is measured. The value of local density for each unit is sampled and displayed with the help of the Pareto Density Estimation (PDE) method proposed in [Ult03b], because it is impossible to calculate the values for all possible neighbours. The U*-Matrix visualization combines both of the methods, thereby meeting the advantages and avoiding the drawbacks of them: the local distances produced by the U-Matrix result in a desired effect in areas with low data density because these areas should present cluster boundaries; on the other site local distances result undesired effect in dense regions because such distances could disrupt the cluster characteristic in these regions, therefore U*-Matrix aims to dampened the local distances of the U-Matrix in dense regions and to emphasize them in loose regions; in regions of average density local distances are changed. Figure 2.3 shows an example that

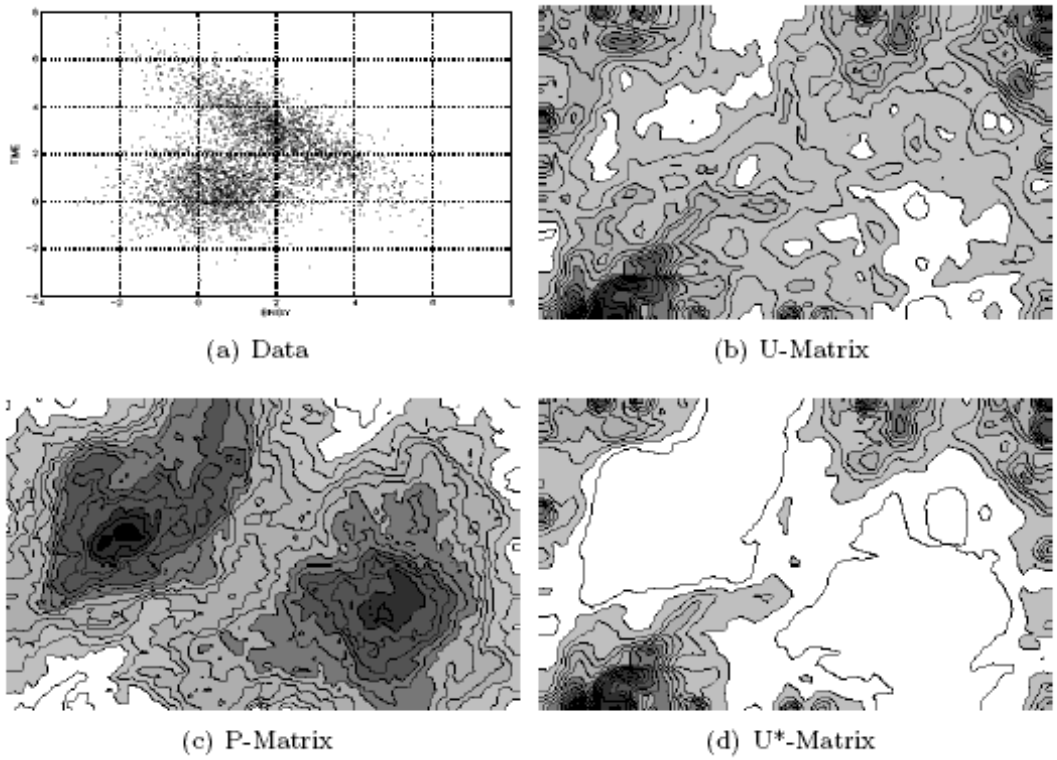
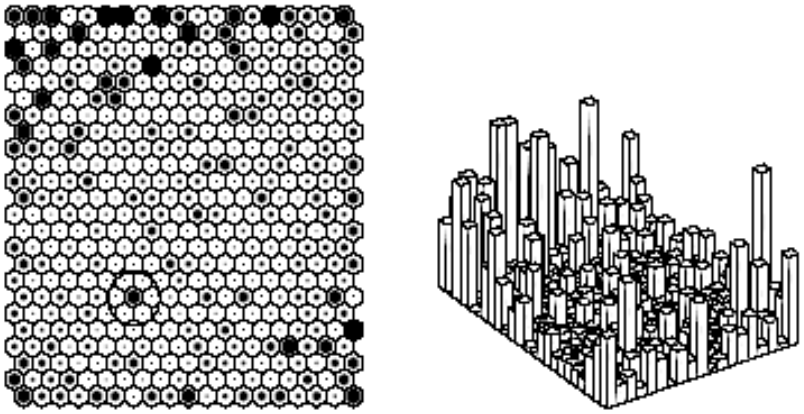


Figure 2.3: U-Matrix, PMatrix, and U*Matrix [Ult05]. (A)Dataset consisting of a mixture of two Gaussians with 2048 points each. (B)The U-Matrix of the dataset, darker colors correspond to large distances. (C)The P-Matrix of the dataset, darker colors correspond to larger dinsities. (D)The U*-Matrix of the dataset, which shows clearly the two Gaussians.

demonstrates the visualizations. It is taken from [Ult05]: Figure 2.3 (a) shows a two dimensional dataset which is a mixture of two Gaussians with 2048 points. Figure 2.3 (b) shows the U-Matrix of the dataset, darker colors correspond to large distances. Figure 2.3 (c) shows the P-Matrix of the dataset, darker colors correspond to larger densities. Figure 2.3 (d) shows the U*-Matrix which shows clearly the two Gaussians.

Histograms Visualizing of the SOM

Hit histogram is a graphical representation of a SOM that shows the number of hits, i.e. mappings of data items, occurred per SOM node. Typically, while training the SOM hits are counted for every node while calculating the BMU.



(a) 2D (b) 3D

Figure 2.4: Hit Histogram visualization methods [Ves99] (a)The size of the black hexagon is proportional to the value of the histogram in the corresponding unit. (b)The height of the bar tells the same thing as in (a).

When SOM is already trained, these hit values are visualized on top of the SOM lattice. There are many methods to visualize these values. Figure 2.4 shows some of these methods. Hit histogram [PRM02, Ves99] visualization is simple and can be achieved with linear costs. Yet, it has some drawbacks: data samples match normally to many SOM nodes in different rates assigning such samples only to the BMU results inaccurate visualization; assigning this sample to all matching hits also results in same inaccuracy because no information is given about the degree of match.

This problem is solved by using smoothed data histograms(SDH) proposed in [PRM02]. SDH aims to visualize the clusters on the SOM by taking into account the probability density of the high-dimensional data. The SOM is used as a basis for the visualization. Instead of assigning a data sample to a specific unit, the SDH estimates a membership degree of every data sample to each unit. This membership is calculated based on distances between the data sample and all other units. The effect of these distances in the membership is controlled with a smoothing parameter s. Figure 2.5 shows a data set and its SDH visualization with various values of s: The influence of s can be seen: for small values of s,

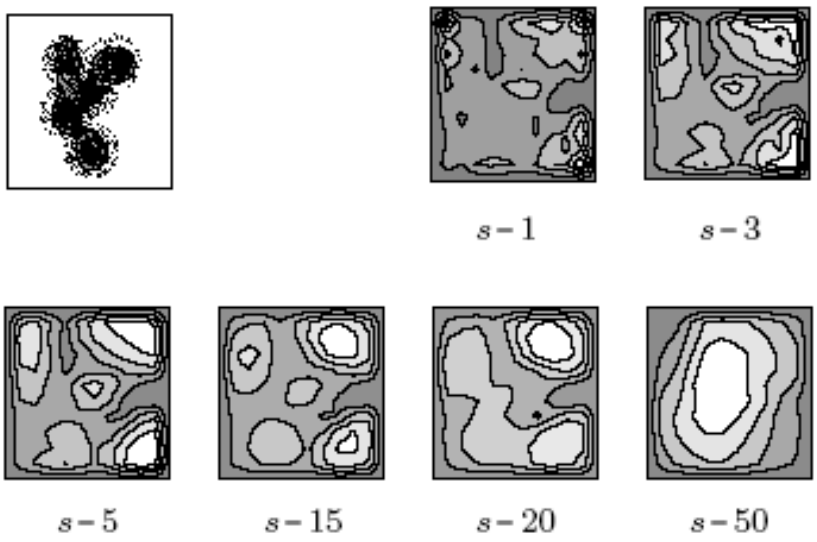


Figure 2.5: SDH visualization with different values of the smoothing parameter s[PRM02]. The Data set consists of 5000 samples, that are randomly drawn from a probability distribution, that is a mixture of 5 Gaussians. The SOM is of 10X10 units arranged on a rectangular grid.

more details are visualized but not the overall cluster topology. By increasing of the value of s, the general characteristics of the data become more visible.

Vector field visualization

A method of visualizing the cluster structure based on vectors is proposed in [PDR]. The vectors are drawn on top of the SOM lattice in the form of arrows having one vector per unit. The length and direction of the vectors are determined so that they give information about the cluster structure. The directions of arrows point to the location of cluster centers and the lengths visualize the location of the unit inside the cluster in term of being a boundary unit or not: long arrows indicates that the unit is located closely to the cluster boundary while short arrows means in general that the unit is located closely to the cluster center or between two clusters to which the unit have equal similarity. In particular, the length of the arrow demonstrates the ratio between the dissimilarity of the unit to the area the arrow is pointing to and the dissimilarity of the unit to the area the arrow is pointing from. This visualization is similar to the U-Matrix

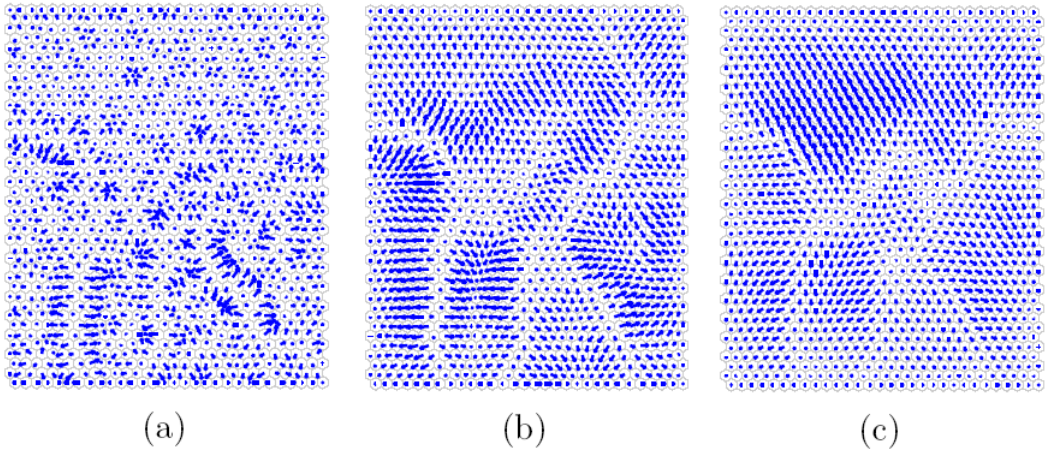


Figure 2.6: Gradient visualization with parameters:(a) $\sigma = 1$, (b) $\sigma = 5$, (c) $\sigma = 15$ [PDR]

visualization but it has the some other advantages especially that the cluster centers visualized by the arrows are computed with the consideration of global areas. The visualization provides an overview of the cluster structure of the data, the locations of cluster centers and the interpolating units, and the similarity of units to the surrounding areas. The influence of the neighborhood kernel on the visualization is controlled with a parameter σ. A large value of σ emphasizes the global cluster structure by enlarging the surrounding area that is taken into account while a smaller value emphasizes the fine details of clusters.

Figure 2.6 shows an example that illustrates this visualization. The example is taken from [PDR] and shows vector field visualizations with different values of σ of a 30X40 SOM that is trained with the Phonetic data set. Low values of σ lead to very granular visualizations, where only direct neighbors are taken into account for the computation of each arrow and thus only local gradients can be observed, as visualized in Figure 2.6 -(a) with $\sigma = 1$. By increasing this value, the clustering structure revealed shifts gradually from local towards global. Figure 2.6 -(b) provides a far better overview on the clustering structure, and individual regions can be distinguished with $\sigma = 5$. In Figure 2.6 -(c), the global structure is shown for $\sigma = 15$.

2.4.2 Visualization of SOM with Labeled data

All of the visualizations presented in the previous sections either only make use of the SOM units to visualize the data set or access the data vectors to make calculations that help by visualizing the data topology. In this subsection, we will present two visualization methods that assume the availability of labeled data and uses these (class) labels to produce a visualization, which shows the class topology of the data.

Graph-based class visualization

A different visualization method is proposed in [Aup03], which is based on graph analysis. This method assumes a labeled data set; that is the class label for each datum is known. This means in case of a trained SOM the class for each data vector is known and the problem is to extract information about the topology of the classes by making use of graph analysis, in particular Gabriel graph. Gabriel graph GG is a sub graph of Delaunay triangulation [For97] in which every edge has a forbidden region of a diameter with length equal to the edge length, Figure 2.7 - c; that is for every edge eij in GG the open ball with diameter $|eij|$ contains no other edges other than eij. This method defines three qualities of data, that can occur in the visualization, Figure 2.7 -a, which are:

1. isolated data units for which all neighbours through the graph have a class different from its own.

2. border for which there is at least one of its neighbours, which has the same class as its own.

3. normal for which all neighbours have the same class as its own.

After applying the algorithm described in [Aup03] visualizations like this in Figure 2.7 -G3 and G5 are obtained.

Pie-chart class visualization

In this section we will illustrate the pie chart visualization described in [RPM03]. Figure 2.8 shows an example of the pie chart visualization. In this visualization

each unit is represented with a pie chart, that is located in the position of the unit. A pie chart has a number of sectors equal to the number of classes assigned to the represented unit. Each one of these sectors represents one of the classes and has an arc length which is proportional to the contribution fraction of this class.

2.5 Summary

Exploring complex data sets, such as data sets with large vector dimensionality or/and large number of vector needs technics that help dealing with them, looking deeper inside them and understanding the relations inside them. The most important ones of these technics are clustering, projection and visualization.

Cluster analysis is the organization of a collection of patterns into clusters based on similarity among these patterns. There are types of clustering algorithms: Hierarchical algorithms, which can be divided into agglomerative and divisive algorithms, corresponding to bottom-up and top-down strategies, to build a hierarchical clustering tree. Agglomerative algorithms begin with each element in its own cluster and the clusters are merged together iteratively according to the similarity between them. The process continue until all elements are in one cluster. The second type of clustering algorithms is the Partitive algorithm, which identifies the partition by minimizing some error function.

Projection methods try to project the data on its underlying subspace, and thus help answering questions like which clusters exist? where are they?, which shape has the data cloud?, etc. Principally there are two types of projection methods: linear methods such as principal component analysis (PCA), and nonlinear such as multidimensional scaling (MDS).

Visualization methods can have some of many goals such as visualizing the overall topology of the data, showing the local details inside clusters, showing labels and class or helping to assign new data items. Two steps are needed for visualization: The first is vector quantization, i.e. reducing data set to a smaller one, that still represents the original data. The second is a vector projection to a low dimensionality. Visualization is the process of projection the real dimensions to visual dimensions like distance, color, position, etc. We differentiate between

methods for visualizing unlabeled data such as U-Matrix, P-Matrix, Vector Field Visualization and methods for visualizing labeled data, which have as a goal the visualization of class topology based on the labels assigned to data items.

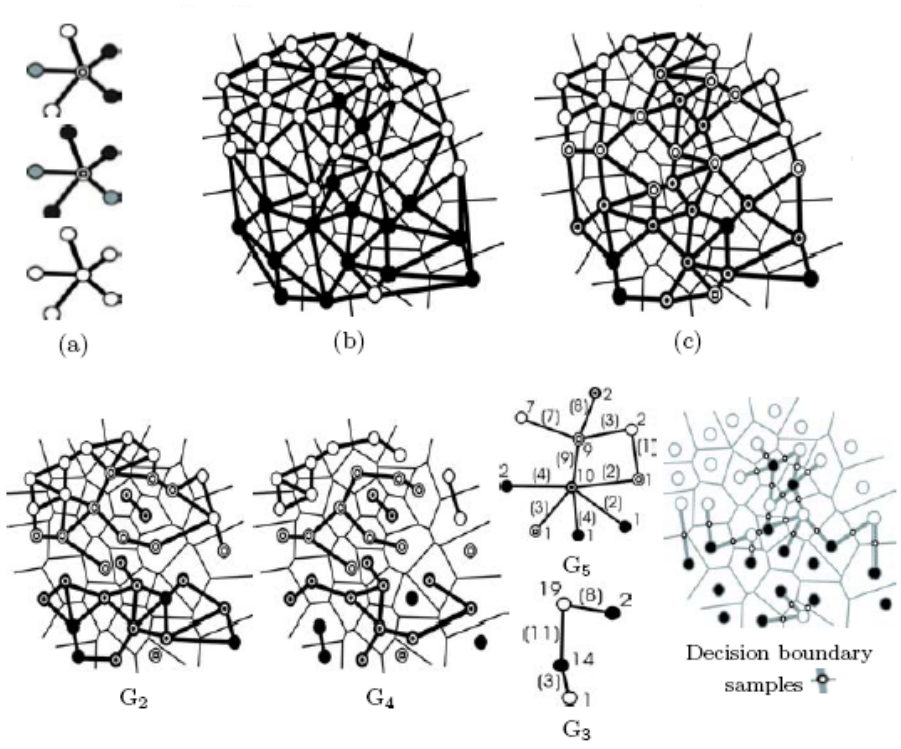


Figure 2.7: Gabriel Graph Visualization [Aup03] (a)Notation for the three data qualities defined in this method, from top to bottom: border, isolated, normal. (b)Voronoi diagram (thin lines) and Delaunay triangulation (bold lines) of the data (circles). (c)Gabriel graph (bold lines) of the data. (G2-G5)Number of nodes x (edges y) in (between) corresponding connected components of G2 and G4, are indicatd beside nodes (edges) of G3 and G5. The class graph G3 shows the topology of the classes and the way they are connected and the density of the connections between the different components, where two classes are drawn with their densities and topology; the class and quality graph G5 provides the same of information as in G3 in addition to the visualization of border and isolated components.

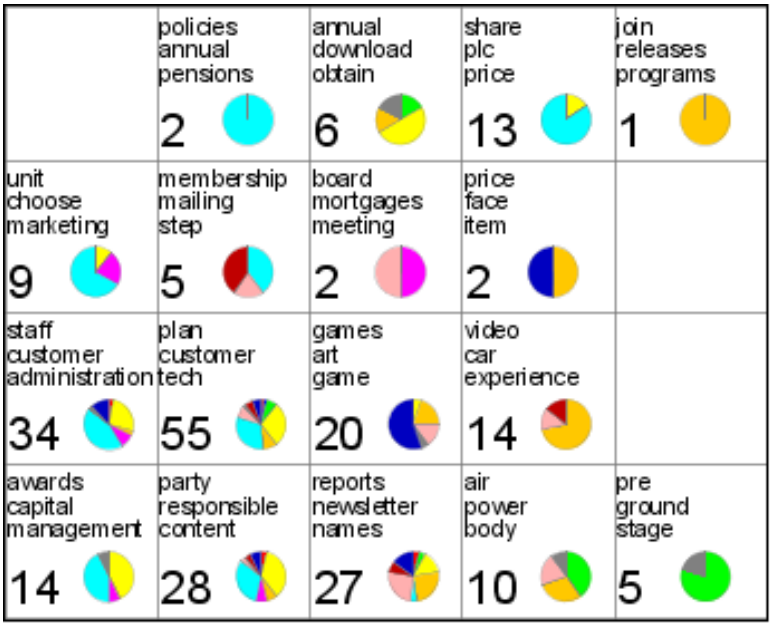


Figure 2.8: Pie chart visualization

Chapter 3

Methods

In this work a novel method for visualization of the SOM will be proposed. The aim of this method is to provide a visualization by coloring the SOM lattice. In particular, a class visualization based on class labels is provided. We will call this method *SOM Class Coloring* and we will use the shortcut SCC to denote it. It is clear that this method assumes (1)a trained SOM, and (2) a data set consisting of labeled data items. These labels are taken as the classes of the data set and used to color the SOM lattice. *SOM Class Coloring* produces a coloring on top of the SOM grid having the lattice of a trained SOM as a visualization ground. The purpose is to obtain a visualization which

1. reflect the class topology of the data set.

2. help assigning new data items and labeling them with one or more classes.

.

The first step in the *SOM Class Coloring* is assigning the labels to the SOM nodes; this is a straightforward step. In this work we assume that this assignment is given and we don't take this step in more details.

This chapter is structured as follows: In Section 3.1 the *SOM Toolbox* is described in more details, which provides the base configuration for the *SOM Class Coloring*. Section 3.2 describes the problem formally. In Section 3.3 we will describe one of the tries to achieve the coloring by letting color point flood in all directions. Section 3.4 introduces the use of graphs for coloring the SOM by

discussing some theoretical issues like voronoi diagrams and Delaunay triangulations which are important for our coloring method and we will then introduce the attractor functions which provide the key algorithms for achieving the smoothed class coloring.

3.1 SOM Toolbox

Although the trained SOM itself can be used to explore the data, there are information that cannot be visualized with the pure trained SOM; some information is always lost through dimension mapping which cannot hold the exact topology and all data relations. Many interfaces and tools were proposed which visualize the SOM and try to cover the drawback caused by lost information through mapping.

In this chapter the *SOM Toolbox* described in [RPM03] will be illustrated having the focus on the visualization aspects. It comprises visualization tools which use the SOM as a visualization start point, it provides then the possibility to make various visualizations on top of the SOM lattice. As input the tool takes a vector file with the raw data, a class info file with the involved class names, and template file with information about the data such as the dimensionality. The tool performs the SOM training process and provides a ground visualization of the trained SOM as a SOM grid with the units represented as pie charts, that show the class contribution. Depending on the details level set by the user, labels can also be displayed on the grid, see Figure 3.1.

As we can see the *SOM Toolbox* provides a basis on which we can build various visualizations. In the next sections we will propose *SOM Class Coloring* visualization and then describe how to extend the *SOM Toolbox* with it.

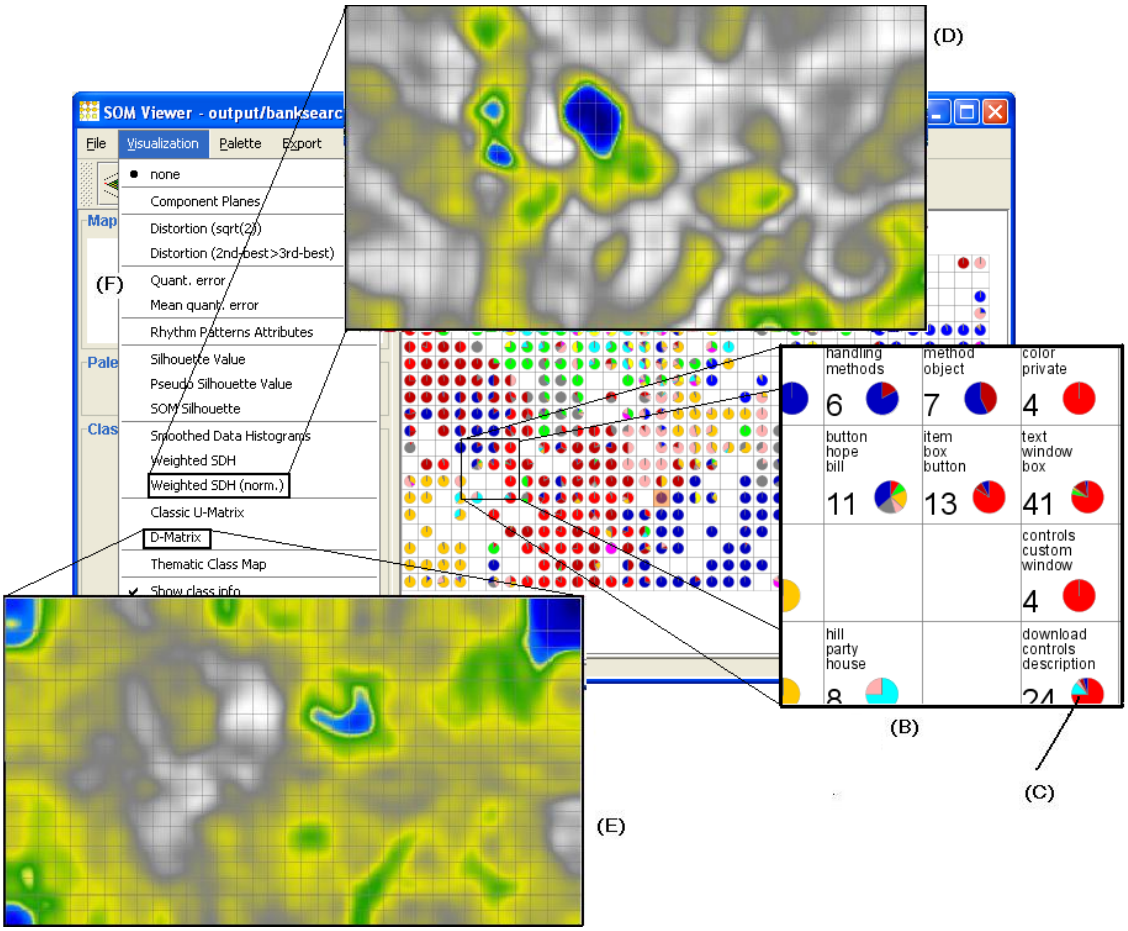


Figure 3.1: *SOM Toolbox* overview An illustration of some features of the *SOM Toolbox* with banksearch data (A)The SOM lattice with units represented as pie charts (B)Zoomed details view with labels and pie charts (C)Pie chart representing the unit position and the class contribution (D)Smoothed data histogram visualization as an example visualization (E)D-Matrix visualization as another example (F)There are other visualizations that can be selected from the visualization menu

3.2 Problem formulation

In this section we will give a formal description of the problem. Let $C = \{c_1, c_2, ..c_n | n \geq 0\}$ be the set of classes involved in the data set and $U = \{u_1, u_2, ...u_m | m \geq 0\}$ the set of units in the SOM. We assume that the assignment relation A is known which is a function $f : U \to R^n$ that assigns the classes of the data to the unit in which these data items are represented by, as well as the fractions of contribution for each class, that is $f(u) = \{a_1, a_2, ...a_n | a \in R\}$ where a_i is a real number giving the contribution of class c_i in the unit u; for example if we assume 5 classes and $f(u) = \{0.0, 0.25, 0.5, 0.25, 0.0\}$ that means that the following data classes and their contribution fractions are assigned to the unit u: 25% from class $c2$, 50% from class $c3$, 25% from class $c3$ and 0.0 (or no data) of the other classes.

We assume that the units are located on a SOM grid of the dimensions $p \times q$ where p is number of rows and q is the number of columns which implies a number of units $m = p \cdot q$.

The problem is to color the grid plane in such a way that the class topology of the data is visualized and that, according to this coloring, a new data item can be assigned to a class. Figure 3.2 illustrates this base configuration.

3.3 SOM coloring by color flooding

The first try to color the SOM was to let colored points flood, that is if we imagine the units as colored points and that the colors spread at the same time, we will get a colored map.

At this point we will try to define and determine the process of flooding the color; there are several approaches how the flooding of a set of color points can be simulated: the first one is that in each iteration a point grows to a ring more. The second is that in each iteration a point grows to one graphic grid (for example a pixel) more in this case the question arises which pixel: we can take the next one at the boundary or a randomly selected pixel from the boundary; this process continues as long as the reached areas are not occupied by other flooding points.

In the simplest case if each unit has one color (single class), things will go

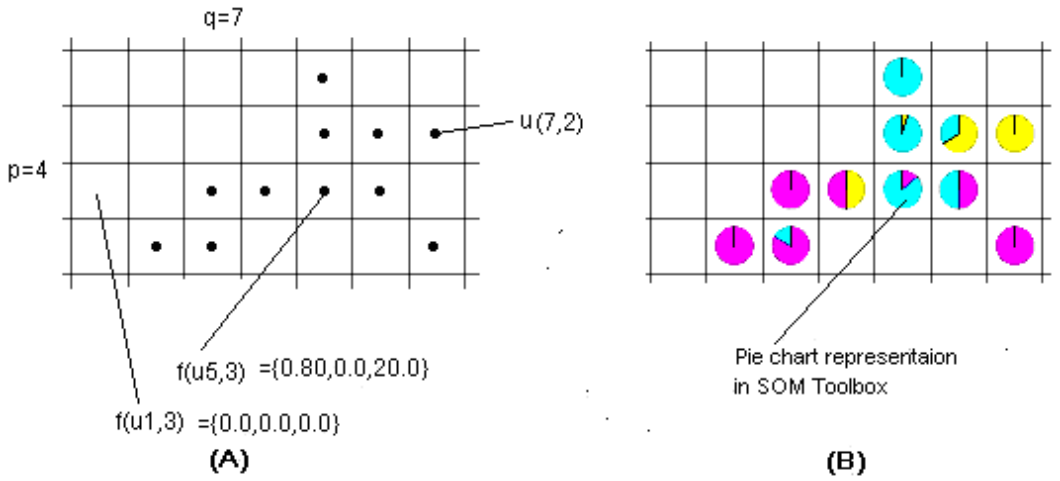


Figure 3.2: Base configuration of the *SOM Class Coloring* (A)The SOM units located on the grid with their class contributions, that is given by function f. (B)Representation of A as it is in *SOM Toolbox*, where f is represented with pie chart.

smoothly and we will get easily a coloring, that represents the class topology. Figure 3.3 illustrate the process of color flooding. As we can see, a reasonable coloring was produced by simple color flooding.

In fact this was too simple case: the main difficulty in this issue is that a unit can contain data of many classes, that is it can have many colors. In such cases a direct color flooding can not produce a coloring that meets our expectation.

To solve this problem we suggest the following concept: a unit with more than one class is substituted by a number of points equal to the number of classes, and these points are located closely to the original unit; then they are rotated round the original point so that they are arranged according to the classes at the neighboring units, that is we thereby aim to have neighboring classes as similar as possible as it is in Figure 3.4 illustrated. When all units are substituted, we let them flood as in the last example. We note here that this concept is not complete because it doesn't take into account the values of contribution of the classes, but it considers as if all classes were equally involved. That means if this concept works well for the first step, it must be then extended to consider the contribution values.

We implemented the suggested concept and tested it and we found, that it

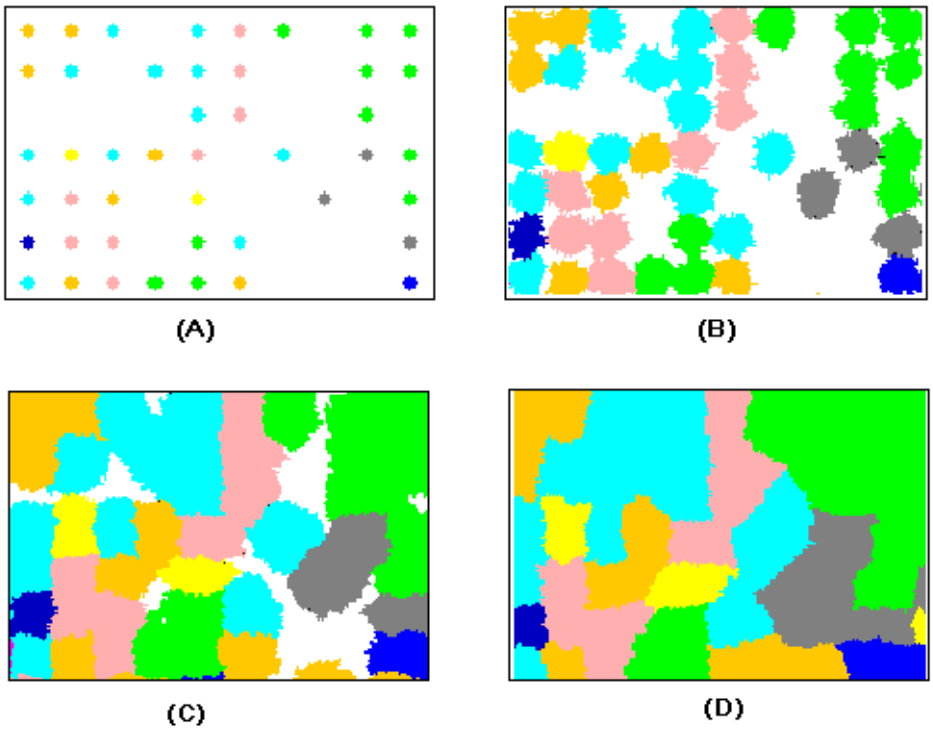


Figure 3.3: Coloring by Simple Flooding assuming one class per unit. (A)Colored points representing the dominant classes of the SOM units, (B to D)The process of color flooding animated in 3 steps.

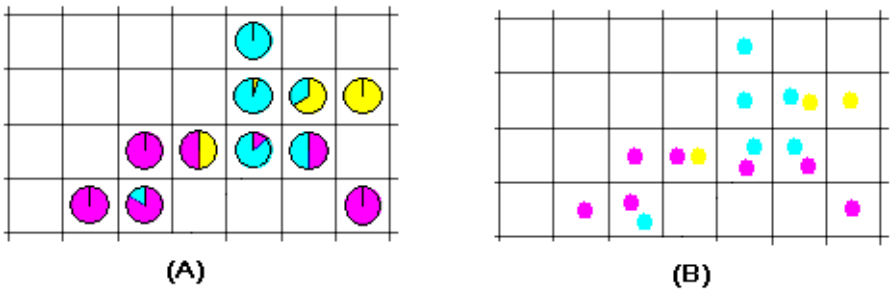


Figure 3.4: Unit substitution.(A) Multi class units represented by pie charts,(B)Substitution points that can be used by color flooding.

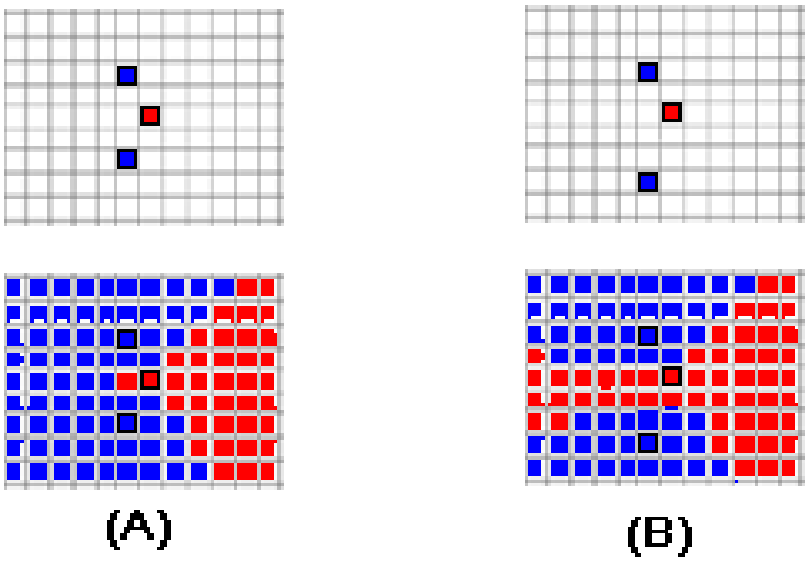


Figure 3.5: Instability effect in the color flooding. (A)While growing, the two blue points close the way and prevent the red point from growing to the left side. (B)After making a slight change in the position of one of the blue points, the red point can grow to the left side, which changes the end result dramatically.

works with critical drawbacks especially the low accuracy and stability: a slice change in the locations of the substitution points could result a dramatic change in the resulting coloring. The random positioning could produce a different result for every run. We analyzed the issue and found that it is caused by a starvation effect which occurs as the following: in one case the flow of an isolated color is stopped by the flow of the surrounding ones; in the other case if a slice change is done, the flow of the isolated color finds a way outwards and grows dramatically more eventually in a differen region. See Figure 3.5. This effect can also happen without making changes in the positions, but only by changing the order in which the pixels are occupied(for example by randomly selection). The drawbacks seemed for us to be in the nature of the concept and thus we stopped to search in this direction.

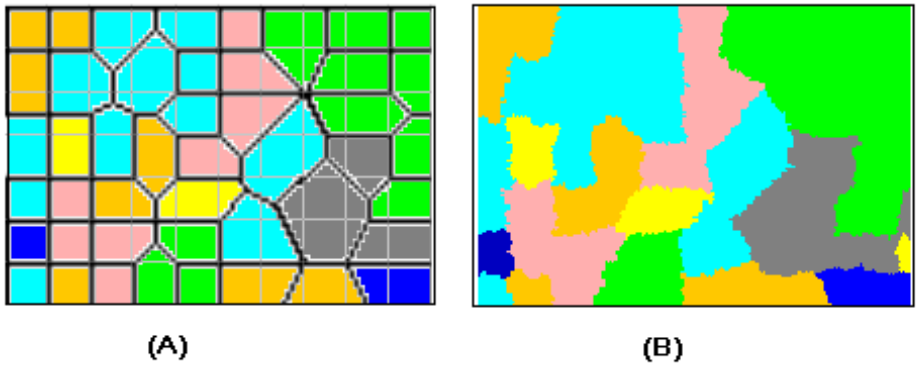


Figure 3.6: Areas in Voronoi diagram (A) are similar to these produced by color flooding (B).

3.4 Using graphs to color the SOM

Our next try was to use graphs segmentation for the visualization instead of color flooding. In this case the decision on Voronoi diagrams is straightforward, because it can produce a similar result like that of flooding the colors in Figure 3.3-(B). This similarity is clear in Figure 3.6. Voronoi diagrams are discussed in details in subsection 3.4.1

However we will face the same situation as in color flooding: After the voronoi areas are found, a solution must be found which considers units with many classes (colors) that have different ratios. Before we come to this issue, the Voronoi Diagrams are discussed in the following subsection.

3.4.1 Voronoi diagrams and Delaunay triangulations

Voronoi diagram [For97] of a set of points (n points here called sites) located on a plane is the diagram that partitions this plane into exactly n regions, so that each Voronoi region is assigned to a site and consists of all points on the plane that are closer to this site than to any other site. More formally: Given a set of sites $S = \{s_1, s_2, ..s_n\}$ on a plane, then the Voronoi region R of a site s_i is defined:

$R(s_i) = \{x : |x - s_i| <= |x - s_j|, \forall j \neq i\}$

where x is any point on the given plane, and the Voronoi diagram V of the site set S is defined:

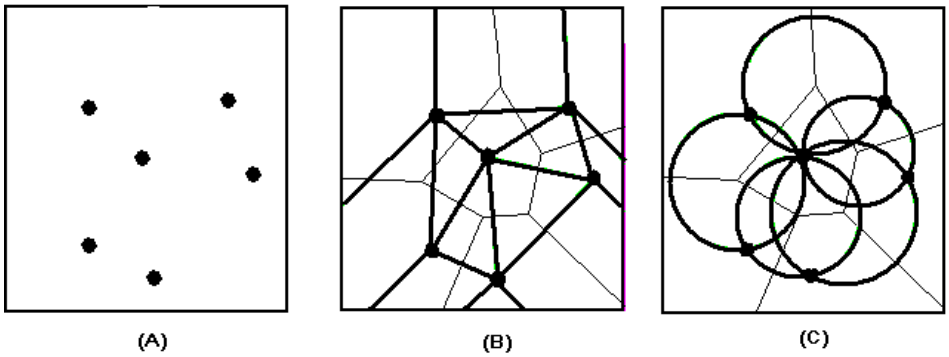


Figure 3.7: Relation between Voronoi diagram and Delaunay triangulation(DT). (A)a set of point (sites) in a 2-D space(B). (B)Voronoi faces (thin lines) and Delaunay triangulation (thick lines) of the sites in A. (C)The circumsphere of every triangle in DT contains no sites inside.

$V(S) = \{R(s_i), s_i \in S\}$

Voronoi diagrams are generally defined in $n - dimentional$ space. In this work we will consider only $2 - dimensional$ spaces because a two dimensional visualization underlies.

There is a close relation between Voronoi diagrams and Delaunay triangulation. Delaunay triangulation [For97] of a set of sites is the unique triangulation so that the circumsphere of every triangle contains no sites inside. voronoi diagram and Delaunay triangulation have a duality relation:

- Each vertex of Voronoi diagram is a center of a circumsphere of a triangle in the DT.
- Every line segment connecting two sites in two neighboring voronoi regions (two regions with a common edge) is an Edge in the DT.

See Figure 3.7. More formally:
$R(s_i), R(s_j) \in V(S)$ *and have a common edge* $e \Leftrightarrow \overline{s_i s_j}$ *is an edge in* $DT(S)$

There are many algorithms for computing the voronoi diagram; there are two main approaches: algorithms make use of the duality and compute the Delaunay triangulation such as the randomized-incremental algorithm [AS92] and Bowyer-Watson algorithm [Dev98]; other algorithms compute the voronoi diagram directly such as the sweepline algorithm [For86]. There are other algorithms that

use the divide-and-conquer algorithm such as the gift-wrapping algorithm [For97]. The complexity of the problem depends on the used algorithm; however the most optimal algorithms have a time complexity of $O(n \log n)$ such as the sweepline algorithm.

In this work we will describe two of these algorithms: the sweepline algorithm and the Bowyer-Watson, which was also implemented in the SOM Toolbox.

The sweepline algorithm

The sweepline algorithm was first introduced by Steven Fortune in [For86] and is often referred to as Fortunes Algorithm. This algorithm has complexity O(n log n), which is optimal for this problem. This algorithm is well-suited to streaming techniques; points are processed in sorted order and potential edges are maintained by a balanced search structure called the sweeping line. It is used in many application for solving problems such as finding Delaunay triangulation, voronoi diagrams and other problems like sorting, tree management, etc.

In this algorithm a sweep line crosses the space and produces events when passing the underlying sites. These events control the computing see Figure 3.8. Actually the sweep line splits the problem domain into two regions, an explored region and an unexplored region. It applies an ordering to the problem because we reason about the explored area based on what we have seen so far and ignore the unexplored area, that is only the points crossing the sweep line affect the current computation since all other points are too far away. In the problem of finding the voronoi diagram see Figure 3.8, the sweepline moves vertically from the top of the plane to its bottom, a parabola is maintained for every site: the parabola arises when the sweepline passes the site and grows while the line sweeping down, so that the base line defines the parabola for the corresponding site. The voronoi diagram is then given by the intersection points of the parabolas. Actually, not all the parabolas are computing while the beach line sweeps, because this would be very costly. Instead, only instances are calculated when the beach line changes topology for example when the sweepline passes a new site. More Information about this algorithm are available in [For86].

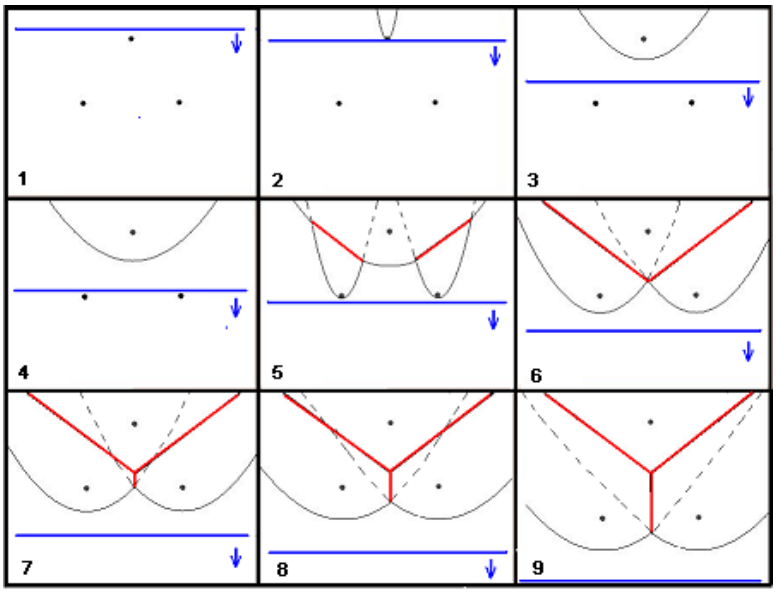


Figure 3.8: Sweepline algorithm. The slides 1-9 animate the algorithm while finding the Voronoi diagram of the three sites in slide 1. In slide 2 a parabola for the upper site arises because the sweepline passes this site. In the next two slides the parabola grows while the sweepline moves. In slide 5 two parabolas for the two lower points arise because the sweepline passes these points. In the rest slides all parabolas grow while the line sweeping. The intersections of the parabolas form the edges of the Voronoi diagram as in slide 9 (red lines).

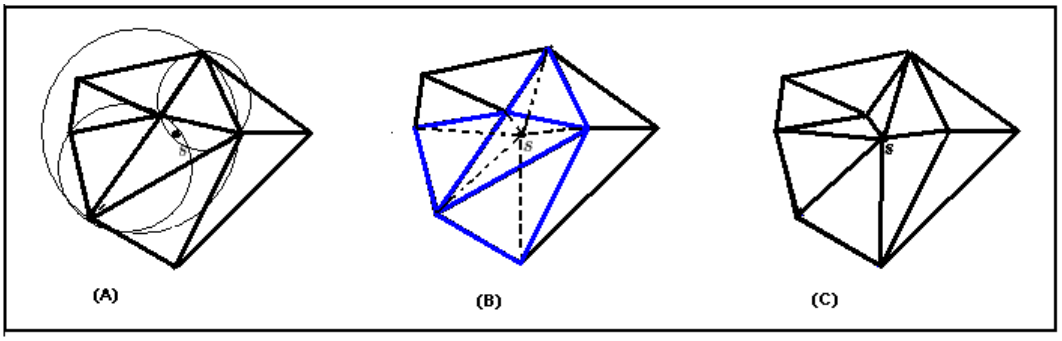


Figure 3.9: Bowyer-Watson algorithm when adding new site. (A)The faces to be deleted are all faces whose circumsphere or itself contains the new added site. (B)Faces to delete are colored, and new faces are marked in dashed lines. (C)The resulting Delaunay triangulation after insertion and update.

The randomized-incremental algorithm (Bowyer-Watson algorithm)

The randomized-incremental (RE) [For86] algorithm in general adds the sites sequentially to the Delaunay triangulation updating the triangulation after each addition; for this update three steps are needed, as in Figure 3.9: the first step is discovering all triangles that are affected by the addition, which are the triangles whose circumspheres contains the added site; the second step is deleting all triangles discovered; the third step is the rebuilding of the empty region as a result of the deletion taking into account the new site, so that every new triangle has the new site as a vertex. The efficiency of the algorithm depends on the efficiency of the used algorithm for discovering faces to be deleted.

3.4.2 Voronoi cell coloring

In the case of single-class nodes, each voronoi cell can be assigned the according class color. As already said in previous chapters the difficulty of our problem is that a SOM unit could have to many classes assigned. If the units were assigned to single classes, then coloring the voronoi regions with the corresponding colors should be the optimal solution of the problem.

In this section we will at first describe some of the solutions for this problem, which we tried but have failed because of some reasons. Then we will introduce in the next subsections two methods that solve the problem and meet our

expectations.

We started by a solution that is based on node substitution similar to the substitution described in Figure 3.3 and Figure 3.4,where each node is substituted with n sites (substitution set) where:

1. n is the number of classes assigned to the node and the sites.
2. each site is assigned a class.
3. All sites in the substitution set are located almost in the same point of the node and
4. The sites are arranged by rotation about the node center, so that colors assigned to sites in a region as similar as possible to colors in the neighboring regions.

Suppose V is the Voronoi diagram before the substitution and VS is the Voronoi diagram for the substitution sites.

Because the substitution site are located (almost) in the same place as the unit itself, then we expect that the substitution (splitting of the units) makes only local changes in the voronoi regions of V in the meaning that the region boundaries of V exists also in VS and each of them is divided into further partitions. Furthermore, because the neighboring sites are taken into account in the arrangement, we expect to have continuous color regions, that go over the region boundaries of V to the other regions. See Figure 3.10

This method has several problems:

1. The class sharing value is not reflected, because the areas of the subregions depend on the positions of the sites and the shape of the region. If these areas are to be controlled, the positions of the substitution sites must be moved but this requires that the whole Voronoi diagram to be changed, in the meaning that the region boundaries of V are also changed. For example in Figure 3.10-B, the region R consists of two classes c1 and c2 and thus divided into two areas; if we imagine, that class c1 contributes with only 1%, then we must move the node very close to the neighbor region to get an area of the 1% of the region area. But moving the node from its position

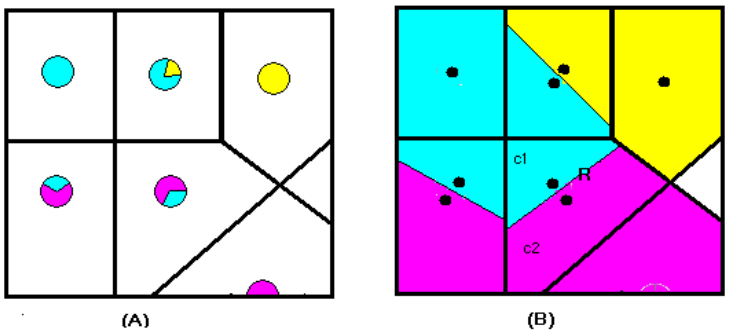


Figure 3.10: Region substitution.

affects the borders of the Voronoi cell itself. This means, that the whole voronoi diagram must be changed.

2. Only moving the sites is not sufficient to meet the contribution grads (sharing fraction), also the rotation about the node is to be controlled. But this rotation could produce a conflict with the aim to tune the own classes with the neighboring classes, as mentioned previously. In other words it could be impossible to consider both of having right areas and right directions for many classes by only positioning the node, because the produced areas depend also on the shape of the Voronoi cell.

3.4.3 Angular segmentation of Voronoi cells

We gave up the substitution and tried another method, namely to divide the voronoi region into sectors similar to a pie chart. Each of them is defined with an angle. The angles are to be calculated depending on the sharing value of the class and on the neighboring classes as depicted in Figure 3.11. The method solves the drawback in the substitution method which causes a global topology change if the sites are moved. But there is another problem or shortage with this solution: we consider the case in Figure 3.11, for input like in (A) the described method would produce a coloring like in (B). If we let a class have more than one sector, then the resulting coloring could be like (C). Actually, in this case the distribution of the classes seems like a red region, that spread over three Voronoi

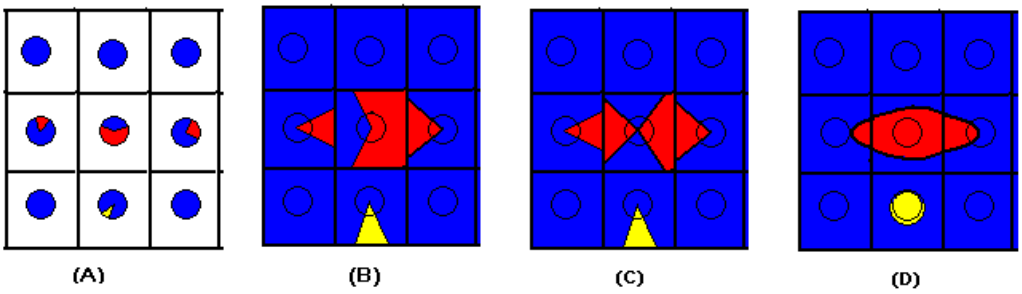


Figure 3.11: Sector partitioning.

cells. However what we would like to have is something like the coloring in (D). Also in case of an isolated class, we would like to have it as an isolated area in the middle of the Voronoi cell and not as a sector. Such a smoothed coloring can not be reached any using the sectoring method.

So we stopped our investing in this direction and searched for other methods. We found two methods, that were satisfying: the first is assigning every grid unit (pixel) to a class separately with the help of a distance algorithm, that takes into account the sharing value and the neighborhood and the second method is by coloring the different classes on a chessboard like grid with the areas according to their relative frequencies. Both of the two methods are described in details in the following subsections.

3.4.4 Smooth partitioning of Voronoi cells

In this section we will introduce a method for producing a smoothed partitioning of the voronoi regions into class (color) areas.

At first let us define some notations to help expressing the problem and our algorithms to solve them.

- $R = \{r_1, r_2, ..r_n | n \geq 0\}$ is the set of voronoi regions which is drawn on top of the SOM lattice as described in Section 3.4.1 with the units as faces.
- $C = \{c_1, c_2, ..c_m | m \geq 0\}$ is the set of all classes that exist in the data set.
- $N(r_i) = \{r_{i_1}, r_{i_2}, ...\}$ is the set of the neighbor regions of region r_i. These are regions that share r_i with an edge.

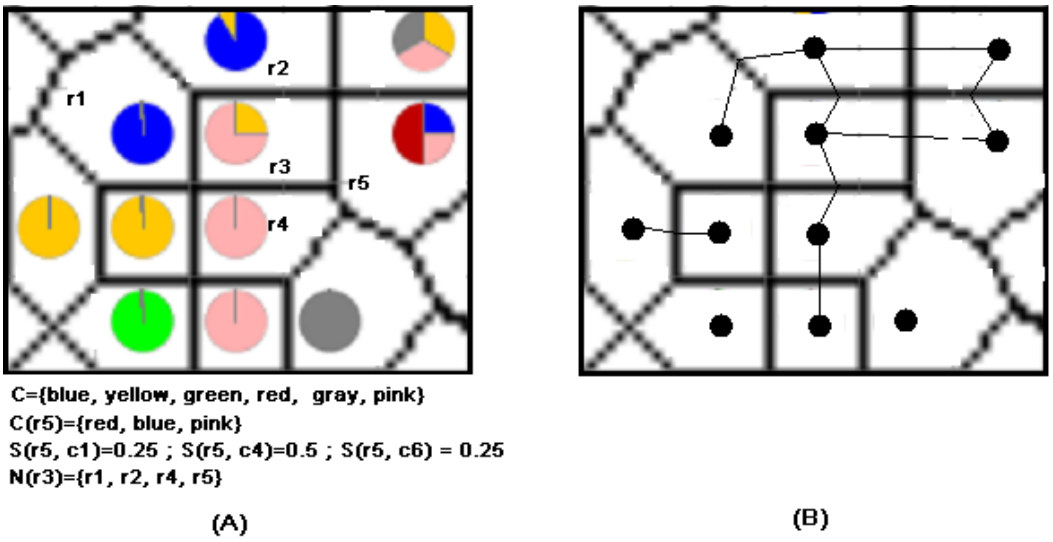


Figure 3.12: (A)Notations used int this section(Note that we use the colors as class names for simplifying). (B)Connection lines between regions that have same classes.

- $C(r_i) = \{c_{i_1}, c_{i_2}, ...\}$ is the set of classes share in the unit located at the face of r_i.
- $S(r_i, c_j) \in R^+$ is the value of the sharing of the class c_j in the region r_i.
- Furthermore we define the term *connection line* which is an imaginary line that connects the site of a region r_i with the middle of the edge of a neighboring region r_n if there is a class c where $c \in C(r_i)$ *and* $c \in C(r_n)$. In other words this line denotes that a region has a class that also the neighboring region, to which the line is connected, has, as shown in Figure 3.12 (B)

A key issue in this method is a function for assigning pixels to classes according to a distance function we will call it *attractor function* because it produces an effect which is similar to the magnet effect, this function is applied on a *region*, a *line segment*, a *class* and a *number n* that denote the number of pixels to be assigned.The function works as follows:

1. At first all pixels (or a predefined number of pixel blocks in case lower granularity representation is desired for performance improvement) in the

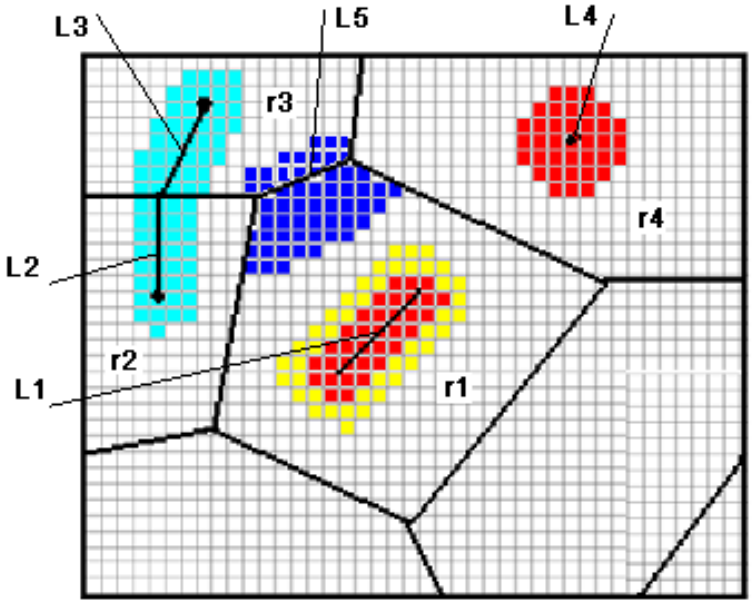


Figure 3.13: The work of attractor function: In region r1, at first the 34 grids, that are most close to L1, are colored in red then the next nearest 45 grids are colored in yellow. The 41 grids most nearest to L5 are colored in blue, also 14 grids from r4 that are nearest to L5 from the other side, are colored in blue. that gives the effect of an area extending over two regions In regions r2 and r3 the same thing is done with L2 and L3, because L2 and L3 have a common point the resulting colored area appears as a thin area, that extends over two regions In r4 the line L4 has the length 0 (is equivalent to a point), thus the resulting colored area has a circular shape.

region are sorted according to their distance to the *line segment*. The distance here is defined to be the distance to the closest point that lies on the line segment(Note that this distance is different from the commonly known distance which measures the length of the normal).

2. After all pixels are sorted, the first n pixels (the nearest ones to the line segment), that are not yet assigned, are assigned to the given class. Figure 3.13 shows this functionality.

The effect of this function is that line segments (we will call them attractors) attracts the pixels of a specific color toward them. Depending on the position of this line segments in the Voronoi cell, the length, and direction of the segment, different attractors are obtained, we list some them:

- Edge of the Voronoi cell: applying the function on an edge of the region lets the color go to the border, now if the function is also applied to the same edge but from the other side (in the neighboring region), then the resulting area reflects a cluster that extends over two regions. see the attractor L5 in Figure 3.13 applied in r1 and r3; this can be applied if two neighboring regions share the same class.

- Line segment connecting the site with the middle of an edge: This attractor is useful to produce a colored area that spreads over many Voronoi cells. See L2 and L3 in Figure 3.13

- Line segment inside the Voronoi cell: produced a pseudo rectangular areas inside the cell. See L1 in Figure 3.13

- Point attractor: It is a line of length 0. It produces a circular area inside the region and is suitable for coloring isolated classes. See L4 in Figure 3.13.

In the remainder of this section we will describe the algorithm for the *attractor* function and then we will describe some rules for finding the $reference\ lines$ depending on the classes that are present in a region and in it neighboring regions, so that the resulting coloring reflects the class topology of the data.

Distance Function

The underlying distance function measures the distance between a point (grid center) and a line segment. In particular it measures the distance d from a point P to the closest point on the line segment $\overline{P_0P_1}$.
Only three cases that can occur:

1. If P lies on the line segment, then $d = 0$

2. If the perpendicular line from P on the line $\overrightarrow{P_0P_1}$ intersects the line segment $\overline{P_0P_1}$, then the searched distance is the normal distance.
   $d = |\overrightarrow{P \perp P_0P_1}|$

3. When P lies at one of the sides of the segment line, then the searched distance is the shorter of the distances between the point and the segment end points; that is $d = min\{|\overline{PP_0}|, |\overline{PP_1}|\}$

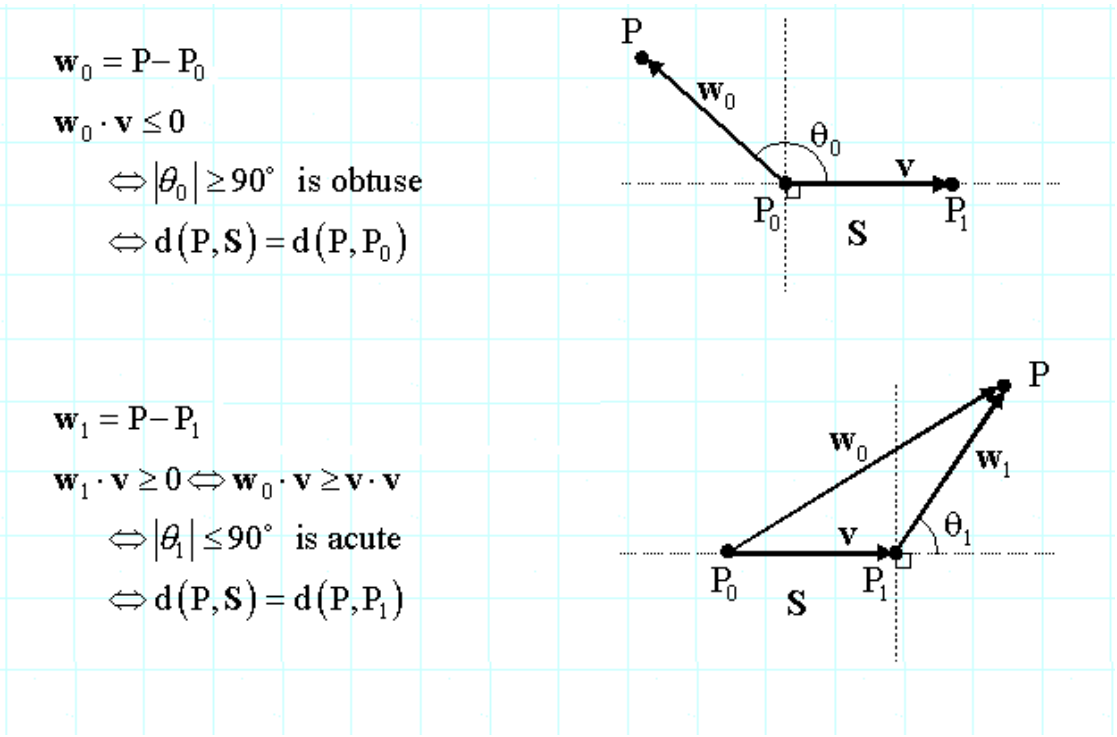


Figure 3.14: Distance between a point and a line segment[Sof].

An easy and sufficient way to find the distance without measuring unnecessary distances is to consider the angles between the segment P_0P_1 and the vectors P_0P and P_1P from the segment endpoints to P. See Figure 3.13. If either of these angles is 90^o , then the corresponding endpoint is the perpendicular base $P(b)$. Otherwise, if the angle is not $90°$, then the base lies to one side or the other of the endpoint according to whether the angle is acute or obtuse. These conditions are easily tested by computing the dot product of the vectors involved and testing whether it is positive, negative, or zero. The result determines if the distance should be computed to one of the points $P0$ or $P1$, or as the perpendicular distance to the line itself [Sof]. See Figure 3.14.

Note that the two tests can be done just using the two dot products $W_0 \cdot V$ and $V \cdot V$ which are also the numerator and denominator of the formula to find the parametric base of the perpendicular from P to the extended line L of the segment S Figure 3.14. This leads to the following algorithm as shown in the pseudocode:

$distance(P, \overline{P_0P_1})\{$
$\quad \overrightarrow{V} = P1 - P0$
$\quad \overrightarrow{W} = P - P0$
$\quad if((c1 = \overrightarrow{W} \cdot \overrightarrow{V}) <= 0)$
$\quad\quad return\ distance(P, P0)$
$\quad if((c2 = \overrightarrow{V} \cdot \overrightarrow{V}) <= c1)$
$\quad\quad return\, distance(P, P1)$
$\quad b = c1/c2$
$\quad Pb = P0 + bv$
$\quad return\ distance(P, Pb)$
$\}$

Coloring algorithm

The underlying algorithm for coloring assumes that there is a SOM lattice with units labeled with classes and and their contribution fractions. A voronoi diagram is first found with the units as sites. Then the resulting voronoi regions are colored each separately taking into account all of the neighbor regions. The coloring consists of the following steps:

- Finding a dominant class for each region and coloring the region with its color. The dominant class is not the class with largest fraction, but the class having the most number of neighbor regions, which have the same class. If the number is equal for several classes, then the dominant class is selected randomly from these.

- For the remaining classes, line segments are found depending on the classes in the region and in the neighborhood. The attractor function is then applied on these line segments, thereby overwriting the dominant class color.

The algorithm for the first step is straight forward: a simple iteration through the neighbor regions and counting the occurrence of the underlying class. If this is done for all classes in the region, then the one with the most hits is selected.

In the second step, the main issue is how to find the best suitable line segments for applying the attractor function, so that the resulting coloring reflects the data

class topology as faithfully as possible. In the rest of this subsection, we will describe and improve an algorithm for solving this problem.

We consider four cases, that can occur concerning the relation between the class c being processed in the region r with the sharing fraction s and between the classes c_i in the neighboring regions r_n. We also assume that the region r contains p pixels. The following attractor types can be applied:

1. The class c is isolated: there is no neighboring region $r \in r_n$ that contains data of the class c. This is the simplest case because we do not need to consider a cluster that extends to other regions for this class. Thus we use the point attractor by adding a segment with length 0 (a point) in the site location and assign $p * s$ pixels for the class c by applying the attractor function for this class and this line segment. See Figure 3.15 -A

2. There is only one neighboring region r_n that contains the class c: in this case we try to attract the colored pixels at the region border close to r_n. Thus we add a segment line from the site location to the middle point of the common edge e (the edge between r and r_n) and apply the attractor function on this segment with $p * s$ pixels. See Figure 3.15 -B

3. There are two neighbor regions r_1 and r_2 that have the class c and that are themselves neighbors to each others. This implies that r,r_1,r_2 all shares a vertex v. In this case we use the point attractor by add a line segment of length 0 to the point v (the common vertex). We do that hoping that when all of the three regions concentrate the color at this point, then we have a colored cloud extending over the three regions. See Figure 3.15 -C

4. There are more than one neighbor region $\{r_1, .., r_n | n > 1\}$ that have the class c but none of them is a neighbor of the other: we treat each of these regions similar as in case 2 with the difference that the number of pixels assigned is $(p * s)/n$ that is we add n line segments, one for each region from the site to the middle point of the corresponding common edge. See Figure 3.15 -D

Border smoothing by weighting the line segments

There is an undesired effect, that occurs if two neighboring areas in two neigh-

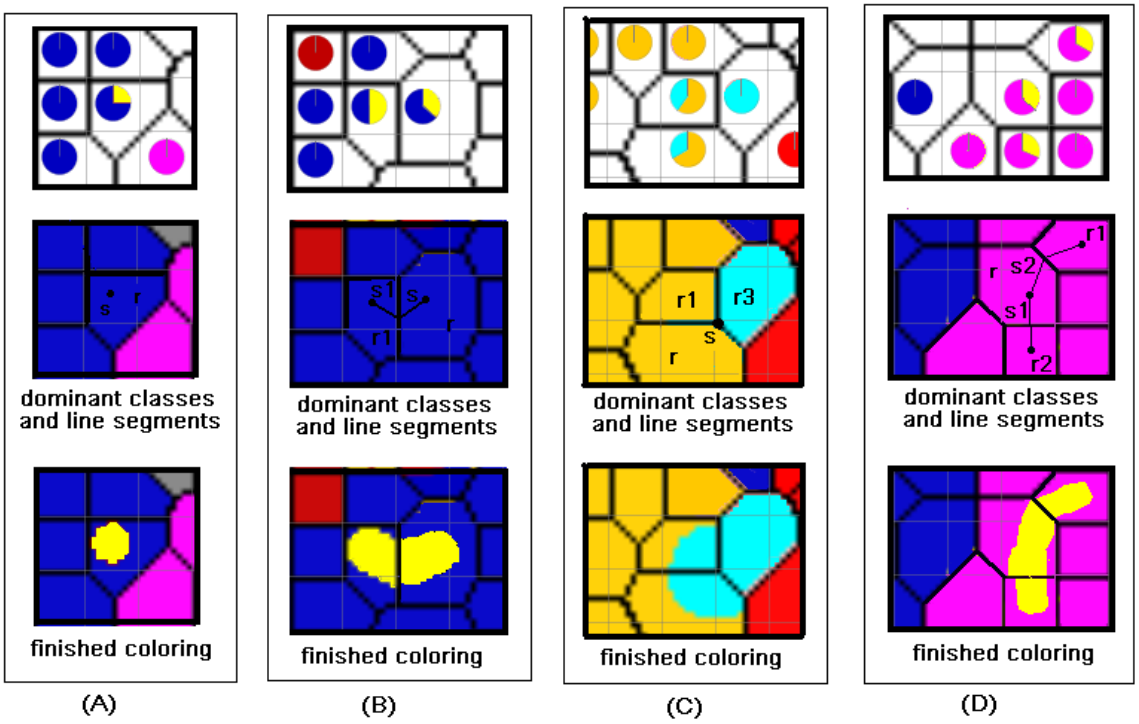


Figure 3.15: Finding the lines segments (A)At first each region is colored with the dominant class color. The class red is isolated because no neighbor region have this class; aline segment s with length 0 is added and the attractor function is applied. In the finished coloring we see the class red as a circular area within the region. (B)The class yellow in the region r remains after painting the dominant colors. There is one neighbor, that have the same class which is r1, thus we add a segment s from the site to the middle of the common edge. The same thing happens when painting $r1$, from the view of $r1$ there is a neighbor region r that have the same class yellow, thus adding the line segment $s1$. After painting r and $r1$ we get continuous yellow area extending over the two regions. (C)In this case the two neighbor regions $r1$ and $r2$ are self neighbors and shares r with the vertex at s because all of the three regions have the class cyan, we add the line segment s with length 0 at the common vertex, so that the resulting class cluster is continuous for all of the three regions as it is seen in the finished coloring. (D)Because r have two neighbors of $r1$ and $r2$, that have the same class yellow, but they are not self neighbors, we add two segments $s1$ and $s2$ each one from the site location to the middle of the edge of the corresponding region. After finishing, we have a yellow stripe, that connect $r1$ and $r2$ crossing r.

boring regions have the same color but they do not have smooth bordering. In particular, this occurs if a class has different contributions in the two neighboring regions or if the regions are of different areas, so that the the border lines at the common edge have a zigzag form. See Figure 3.16 -A. To solve this problem we use a *weighted comparator* for the sorting process needed in the attractor algorithm described in 3.4.4. Recall that the pixels are sorted according their distances to a line segment and then they are assigned to the class in this order. The weighted comparator uses a weighted distance metric for finding the the distance to the line segment. That is, the line segment is assigned two weights for the two ends and the weights are taken in account in measuring the distance: If p is the point to be measured and $\overline{p_1p_2}$ is the line segment then the weighted distance is:

$$d_w(p, p_1, p_2) = d(p, p_1, p_2) + |\overline{pp_1}|(1/w_1)^2 + |\overline{pp_2}|(1/w_2)^2 \tag{3.1}$$

where $d(p, p_1, p_2)$ the normal (not weighted) distance and w_1, w_2 the weights for the line segment ends p_1 and p_2 respectively. The weighted distance results the effect that pixels tends to be concentrated more around the end point with the larger weight. Now if we have two neighboring regions having the same class like Figure 3.16 -A, where the contribution fractions for the class c are f_1 for region r_1 and f_2 for region r_2, then we assign the points of the line segments connecting the two regions weights as follows: the face points take the contribution value as a weight $w1 = f_1$, $w_2 = f_2$, and the common point is assigned a weight of the value $\frac{w_1+w_2}{2}$. See Figure 3.16 -B. This implies that the pixel concentration on both sides of the common edge is the same for both regions regardless of the values of the contribution fractions. This weighted distance produces a smoothed bordering like in 3.16 -C.

The minimum visible class parameter

There are Nodes that have classes assigned with a low contribution. Some of these are outliers. If one is interested to have a visualization of a certain abstraction level for example if only the main class clusters are important, the such classes could make an undesired effect, because the are unimportant details. For this reason we defined a parameter, the *minimum visible class*, which can be set by the user to values from 0 to 100%. This parameter defines the minimum

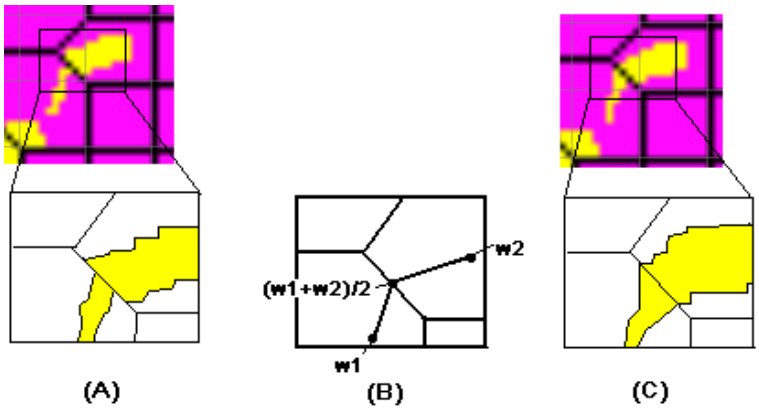


Figure 3.16: Smoothing of the border by using weighted line segments. (A)The border at the common edge has a zigzag form because the areas are not equal. (B)The line segments are weighted by using the contribution value as weights for the face points and the average for the common point. (C) Smooth partitioning as a result of the weighted sorting.

contribution, that a class must have in order to be painted. Setting the parameter to 0 will cause that all details are painted; while setting the parameter to 100% will cause that only the dominant classes are painted. setting the value to any other value will cause that only these classes are painted, that have a contribution equal or greater than this parameter. Note that the effect of this parameter is applied only on classes other than the dominant class, because the dominant class is painted in all cases as a background, which forms a base for painting the other classes. Figure 3.17 shows a comparison between three visualizations, with the parameter set to 0 in the first and to 50% in the second, and to 100% in the third.

Chessboard visualization

In this section we describe another alternative for visualizing the regions with multiclass contribution. In this visualization every region r is divided into grids. For every class c with contribution f in r, a number $m = n * f$ grids is calculated, where n is the total number of the grids in the region. Now the pixels are are assigned to the classes by using a uniform distribution function. when each pixel is then colored with the corresponding class color we get a visualization like this in Figure 3.18

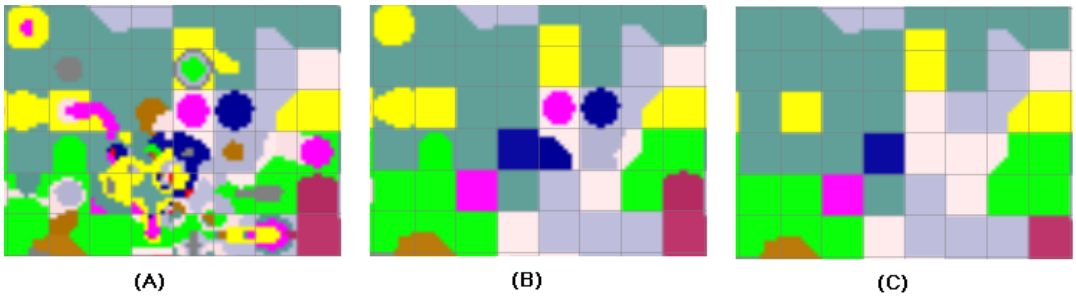


Figure 3.17: The minimum visible class parameter. (A) Visualization with parameter set to 0. (B) The same data visualized with the parameter set to 50% and (C) set the parameter set to 100%.

3.5 Summary

To visualize a trained SOM with labeled data, we propose a method for coloring the SOM lattice according to the class labels. We assume that every unit in the trained SOM has one ore more classes assigned to it and every class is assigned a fraction which gives the contribution of the class in the unit. The Coloring method consists of the following steps:

- Partitioning the SOM lattice with a Voronoi diagram having the units as sites. That is every unit is the site of a Voronoi cell.
- Using the attractor function to partition the Voronoi cells, each one separately taking into account the classes in the unit and in the neighborhood.

The attractor function is an algorithm, that attracts the colored pixels to a line segment. To partition a Voronoi Cell with the attractor function, we select line segments with suitable lengths, directions, start and end points, taking into account the classes in the unit and the classes in the neighboring units. Finally we apply the attractor function to these line segments and the corresponding classes.

Visible classes can be filtered according their contribution fractions with the help of the *minimum visible class* parameter, which can be set interactively by the user.

There is an alternative visualization which color the Voronoi cells with a chess board like coloring: The Voronoi cell is divided into squires, which are colored

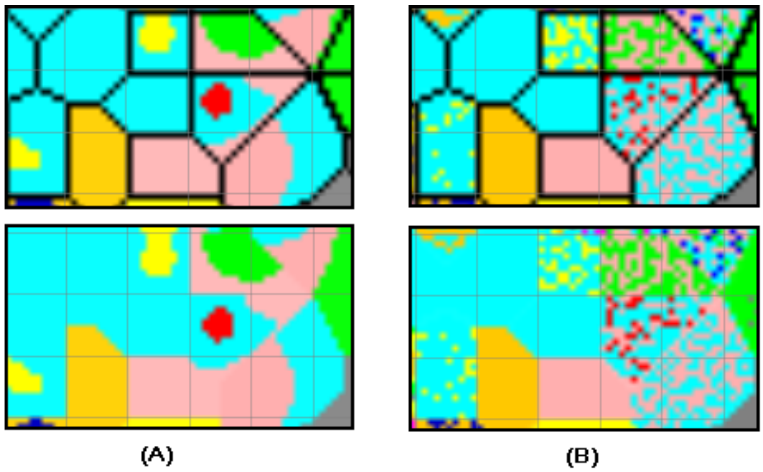


Figure 3.18: Chessboard Visualization vs. smooth partition visualization. (A)the smooth partition visualization, (B)Chessboard visualization. In each case with and without voronoi borders.

according a uniform distribution function taking into account the contribution fractions for each class.

Chapter 4

Experements and evaluation

In this chapter we will test and evaluate our work with three data sets, selected from three different practical areas: the first one the iris data set, which is a simple data set consisting of 150 sample of flowers. The second data set is the banksearch data set, a standard data set for testing clustering and classification method in the web fields; it consists of 11000 web documents. The third one is an audio data set: it consists of audio samples taken from the broadcast of 8 radio station.

4.1 Iris data

The Iris benchmark database is a data set with 150 random samples of flowers with 4 attributes (sepal and petal length and width, of Iris plants). There are three classes labels: Iris Setosa (St), Versicolor (Vs) and Virginica (Vg). Exactly 50 data are labeled with each of the three classes. A SOM with the following configuration was trained with this data set: 35 X 25 SOM grid, 10000 iterations, 0.7 learn rate, random seed of the value 7, no normalization was performed on the data. Figure 4.1 -A shows the pie chart visualization of the trained SOM and Figure 4.1 -B the class coloring visualization.

The iris data is a simple data set in relation of class distribution: as it is clear from the visualization there are no SOM nodes with multiple classes, each node has a single class assigned. Even so we can see the benefit of the SOM coloring when it is compared with the pure pie chart visualization: with the help of the

SOM coloring we can directly see the class topology of the data with sharp class cluster boundaries. In fact the main class clusters can also be seen in the pie chart visualization, but the absence of exact borders make it difficult to decide for data points that are close to the cluster borders and those points that lie near outliers or noise nodes. See the regions marked with circles in Figure 4.1.

The main benefit of the coloring in case of such simple data is the ability to label new data: after we obtain such a coloring from a labeled data set, we can assign a new unlabeled data by finding the best matching unit for this data and then find in which colored area this unit is located. Note that the best matching unit can be eventually a new node, if so, we can see the benefit in comparison with the pure pie chart visualization, where we must estimate and choose one of the nodes in the neighborhood to assign the class.

4.2 Text data

In this section we will use the Banksearch data set described in [SC02] to test and evaluate our visualization method. Banksearch data set was designed as a standard data set for general use in web document clustering and similar experiments and in particular for research in the field of unsupervised clustering of web documents, so that clustering methods can be tested with a common data set, and thus be easily and objectively compared. It was aimed to use a relatively large number of categories, but with sufficient of each category to allow a wide variety of sensible experiments, each using clearly defined subsets of the data set. The banksearch data set consists of 4 main categories (Banking & Finance, Programming languages, Science and Sport), which are further divided into a total of 11 subcategories (themes). These Categories, their themes and identification Ids are listed in Table 4.1. For each one of the 11 themes, the data set contains 1000 documents,which means a total number of 11000 documents.

A SOM with the following configuration was trained with this data set: 35 X 25 SOM grid, 10000 iterations, 0.7 learn rate, random seed of the value 7, no normalization was performed on the data. Figure 4.2 shows an overview of the class visualization of the trained SOM: (A) is the pie chart visualization and (B) is the smoothed class coloring of the trained SOM.

Table 4.1: Categories and their associated themes in the banksearch dataset.

Dataset Id	Dataset Category	Associated Theme
A	Commercial Banks	Banking & Finance
B	Building Societies	Banking & Finance
C	Insurance Agencies	Banking & Finance
D	Java	Programming languages
E	C/C++	Programming Languages
F	Visual Basic	Programming languages
G	Astronomy	Science
H	Biology	Science
I	Soccer	Sport
J	Motor Sport	Sport
K	Sport	Sport

by comparing the pie chart visualization with the smoothed coloring in Figure 4.2 we see that the class clusters are easier to recognize in the smooth coloring, especially those clusters formed by nodes with multiple classes. With pure pie chart visualization, it is difficult to recognize such class clusters. Smooth coloring provides sharp boundaries between class clusters. These boundaries are helpful in the process of assigning new data.

Figure 4.3shows the class coloring of the same data with different values of the parameter minimum visible class. In Figure 4.3 -(A) only the dominant classes are visualized; This is achieved by sitting the parameter minimum visible class to 100%. Recall that the coloring algorithm find the dominant class for each Voronoi cell and paint the class color as a background for the corresponding cell. Then the algorithm paints the rest of the classes. By setting a large value for the parameter minimum visible class, we prevent the algorithm from painting classes other than the dominant class. The dominant class visualization shows the general class topology by visualizing the main classes and ignoring the details and outliers. In Figure 4.3 -(B) the parameter is set to 50%, only few details are seen. In Figure 4.3-(C) the parameter is set to 0% and this causes the no classes are filtered and thus all details are visualized.

In Figure 4.4 the the border of the voronoi cells are visible to help analyze the

coloring. Furthermore 6 samples from different areas were zoomed. The pie chart visualization of each sample is placed beside it to help comparing and analyzing the sample. Samples of significant meaning were selected; some of them show strength and other shows drawbacks of the class coloring.

The sample in Figure 4.4 -(A) shows a drawback in the coloring algorithm, namely finding the dominant class: The Voronoi cell marked with a circle was not optimal partitioned: we expect that the blue area in the marked cell is connected to the blue cluster in the neighborhood. Actually there is an isolated blue area in this cell. after analyzing the sample we found that this happens if there are two classes that are equally dominant. In our case they are blue and brown. The randomly decision to one of the two classes results in such sub-optimal partitioning.

The sample in Figure 4.4 -(B) shows an area with a heavy interleaved class distribution: As it is shown in the pie chart visualization, the nodes in the sample have up to 7 classes assigned. Partitioning such areas is not meaningful especially if the cells are small and close to each other. The resulting coloring looks like chessboard visualization.

The sample in Figure 4.4 -(C) also shows a drawback concerning the areas marked with circles and colored in light gray. Three neighboring Voronoi cells have the same class, namely the class colored in light gray; it is expected to have a continuous gray area extending over the three cells. Actually we see four gray areas, two in the median cell and one in each of the two other cells. We analyze the case and found that this is caused by the class colored in green, which has two special properties: isolated and has a large fraction in the cell. The algorithm always assigns isolated classes first of all. That is the most of the area is colored in green which means there is no enough place to paint a gray stripe, which is drawn through the cell and connect the two other cell. If we change the algorithm to paint the isolated classes at the end, then we will face the problem, isolated cells could be split int two or more parts, which is also not optimal.

The sample in Figure 4.4 -(D) shows an optimal partitioning in our opinion: The class colored in brown form a continuous cluster that extends over many Voronoi cells. The cluster borders are smoothed without zigzag boundaries. Isolated classes are also painted as expected. If there are more than one isolated

class in one Voronoi cell, they are painted as rings in the middle of the Voronoi cell. In fact this good quality holds for most of the cases.

The sample in Figure 4.4 -(E) shows a typical example of isolated classes. Isolated classes are painted as a circle in the middle of the voronoi cell. If there are more than one isolated class, then the they are painted as rings about the circle. The order of painting (which class to which rings) is randomly selected.

The sample in Figure 4.4 -(F) shows the effect of the weighting to produce smoothed borders without zigzag boundaries. The yellow area marked with a circle has smoothed boundaries, i.e. at the points, that join the yellow areas in the three cells cells.

Figure 4.5 shows the chessboard class coloring of the banksearch data set. This visualization is meaningful especially in the areas, where the classes are interleaved, because in this case the smoothed partitioning makes no sense.

4.3 Audio data

In this subsection we will test and evaluate our visualization method by using the data set presented in [LR06], where a method for profiling radio stations is described. The aim of this method is to use SOM to give listeners the possibility to easy select the radio stations they like from the overwhelming number of radio stations, that exist online and over the air. Radio station maps are created, which visualize the profiles of radio stations so that it is possible to directly pick a specific program type instead of having to search for a suitable radio station.

In this data set, audio output of 8 radio stations was analyzed: features from the audio signal received from each radio station were extracted, then a two-dimensional SOM was trained with this data. Rhythm Histograms and Statistical Spectrum Descriptors were used for profiling radio stations. Data was sampled with a rate of one feature vector for every 6 seconds received from a radio station broadcast. For more information see [LR06].

Figure 4.6 shows the class coloring of the the data set. There are 8 classes, which represent the radio stations. The coloring partitions the SOM-lattice into 8 partitions, which are painted on top the SOM lattice. The nodes are organized according the similarity between feature vectors, which represent the genre of

the broadcast signals; that means that near areas on the Map represents similar genres regardless of the color(radio station).

To help testing and analyzing the coloring, we manually divided the lattice into regions labeled with the most important categories of genres, such as *speech*, *classic*, *pop*, *schlager*, etc. These labels was taken from [LR06]. Each region represents a main genre and can be further divided into subregions that represents different genres that are similar; for example speech can be divided into news, reports, advertisement, etc.

At first sight, we can see that there are some large areas painted in one (main) color forming class clusters; also, there are other areas that are painted in many colors and in some cases they are heavy interweaved. Class clusters can be interpreted as radio stations that are specialized or have the focus on audio broadcast of certain genres, an example for this type could be the radio station Oe1 and Bgld: For some one who is familiar with these radio stations, this is meaningful, because radio station Oe1 is a radio station which *classic* music as one of the main broadcas categories, whereas the other analyzed stations seldom broadcast classic music. The same thing holds also for the genre *schlager* and the radio station Bgld: Bgld Radio station broadcast very often *schlager* music.

Interweaved classes can be interpreted as radio stations, that broadcast similar or almost the same genres; an example could be the radio stations OE3 and TIROL, which are mostly represented by the area top/left in Figure 4.6 -B. Also it can interpreted as radio stations that broadcast genres in a wide spectrum and has no focus, such as the radio station ONE.

The area represents the genre *Pop* music (upper part of the figure) is very interleaved. This is meaningful because because *Pop* music is very popular and is broadcast by almost all radio stations. The same thing holds exactly for the area *Speech* especially those regions representing *news* genre, which is also expected, because all radio stations broadcast news.

In the areas with heavily interleaved classes we face the problem of ambiguousness: if the classes are uniformly (or nearly uniformly) distributed, the coloring algorithm fails to find a definite order to color the areas; namely to find the dominant class, which is painted as a background for the voronoi region, on which the other classes are painted. The selection of the dominant class depends on

the class distribution in the neighborhood. In case of heavily interleaved areas and, the dominant class may be selected randomly. This means that more than one coloring are possible. Figure 4.6 -E shows such a case. On the contrary Figure 4.6 -F shows areas with classes, that are not very interleaved. These areas have a clear class structure, because the dominant classes and coloring order is algorithmically clear defined. In fact the ambiguousness in the coloring of heavy interleaved classes can be ignored: we can not assume that the trained SOM holds and reflects the data exactly, but only the general topology and details up to a certain grad are hold; this means that the coloring algorithm deals with a class distribution that meet the reality with a certain accuracy level, which varies depending on many factors; in such a situation we can ignore the error and the ambiguousness that result from the heavy interleaved class distribution.

Figure 4.7 shows the class coloring of the data set with different values of the parameter minimum visible class. As we see in the figure the visible details decrease as the parameter value increase. After comparing the visualizations Figure 4.7-(A) an Figure 4.7-(D) we can see a serious problem: if we look at the areas marked with circles in (A), we see a dominant colors which are brown in the upper area and red in lower one. After setting the parameter to 100% the elementary classes should be eliminated and only the dominant class should be visible. If we look at the visualization in (D) we see that the corresponding areas (areas marked in circles) have a different classes than these dominant classes we see with our eyes; namely blue/cyan in the upper area and green in the lower one. This means that there is an error in the coloring algorithm. We analyzed the problem and found the error; it was namely in the algorithm, that find the dominant class: the algorithm finds the dominant class as follows: for each class in the actual unit, it counts the regions in the neighborhood which contain this class, then it selects the class with the maximum number of such regions. The algorithm does not take into account the contributions for these classes. This means that if most of the surrounding regions have a class then it is considered the dominant class regardless the contribution fraction of this class. So the marked area in (A) right/buttom have the class green in most of the cells but in a low contribution, nevertheless it is considered as a dominant class and this is the error.

We corrected the algorithm to consider the weighted occurrence instead of only

the occurrence of the class in the surrounding regions in finding the dominant class; That is, the dominant class is the class for which the sum of the fractions in the surrounding neighborhood is the maximum.

The same test after the correction produced a satisfying result, which matches the expectation and agree with the dominant class seen by the human eye. The result after correction is illustrated in Figure 4.8.

Also, we tested the banksearch data set after the correction and we found that some effects, which we considered as sub-optimal, disappeared. For example the effect in Figure 4.4-(C). The class coloring of the banksearch data set after the correction of the error is illustrated in Figure 4.9

Figure 4.10 shows the chessboard visualization of the radio search data set. In the visualization we can see the main clusters and the class topology and clouds of color mixture in the areas with interleaved class distribution. This coloring is meaningful because if you concentrates only at the full painted areas you see the dominant classes and thus the general class topology. Whereas if you concentrate at the color mixture you see the detailed class distribution.

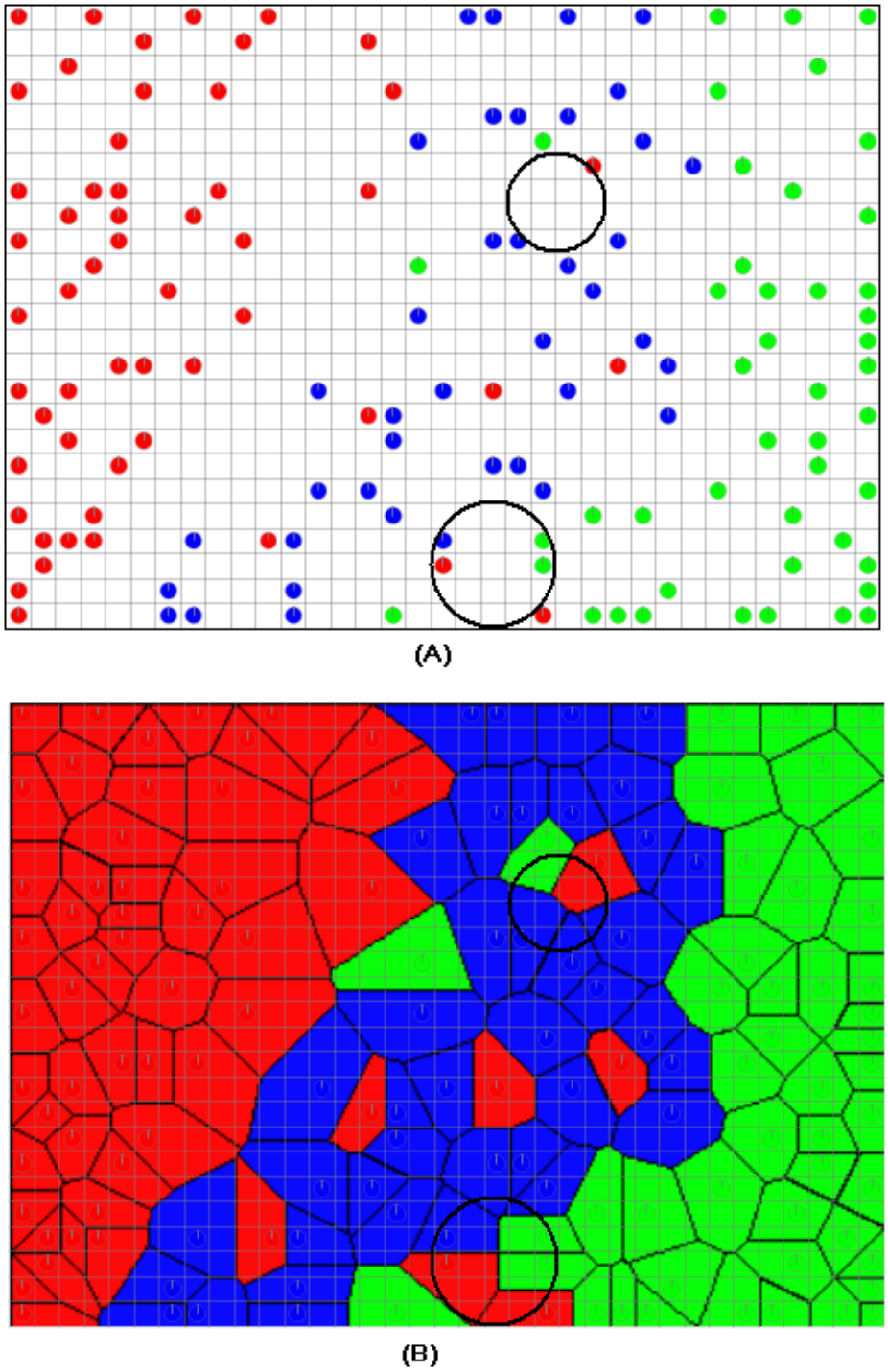


Figure 4.1: Class coloring of the iris data. The areas marked with circles are examples of regions, where it is difficult to assign new data without using the coloring.

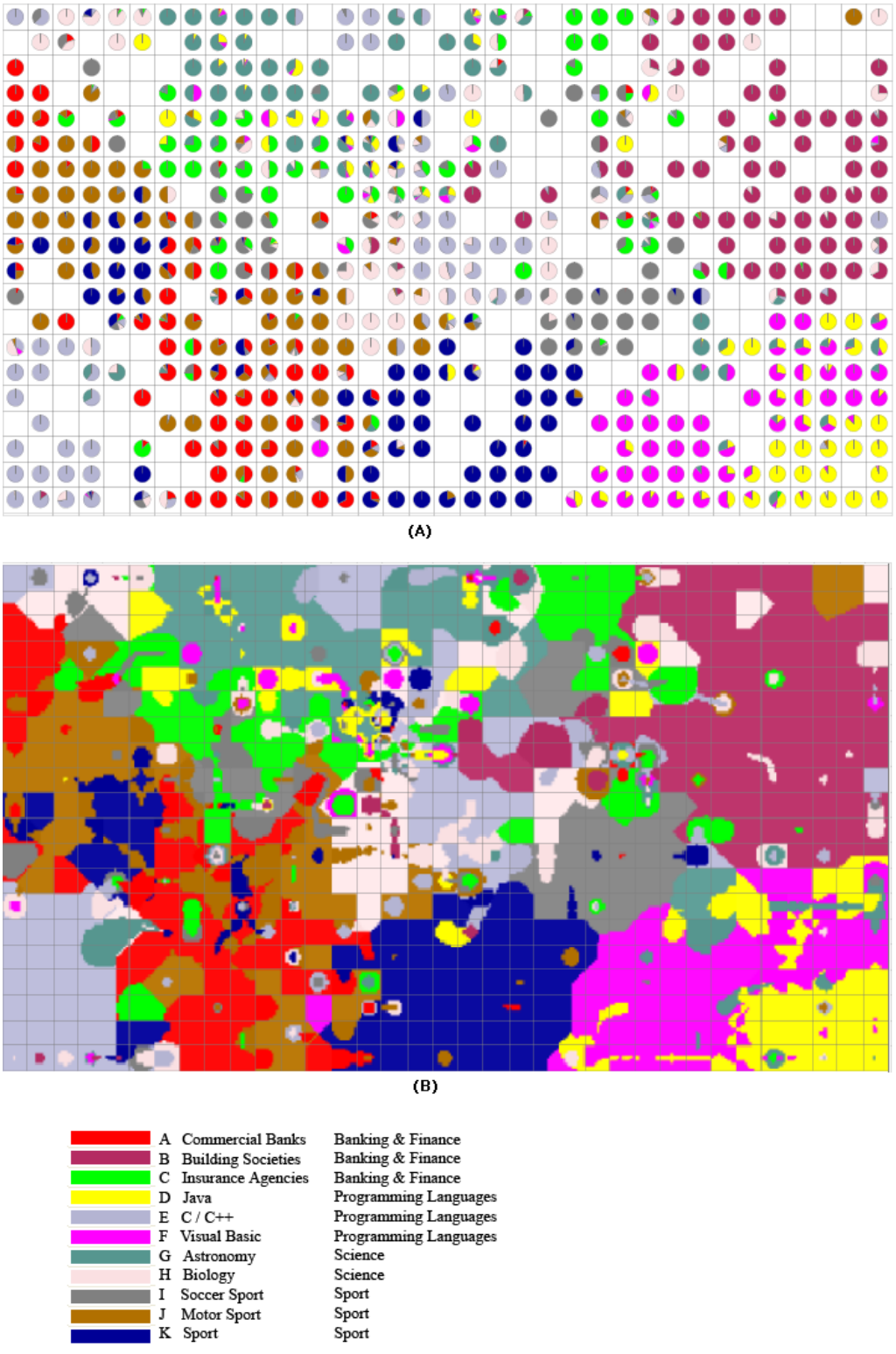


Figure 4.2: Class visualization overview of the banksearch data set. (A)Pie chart visualization. (B)Smoothed class coloring of the same data set. For the categories and themes see Table 4.1.

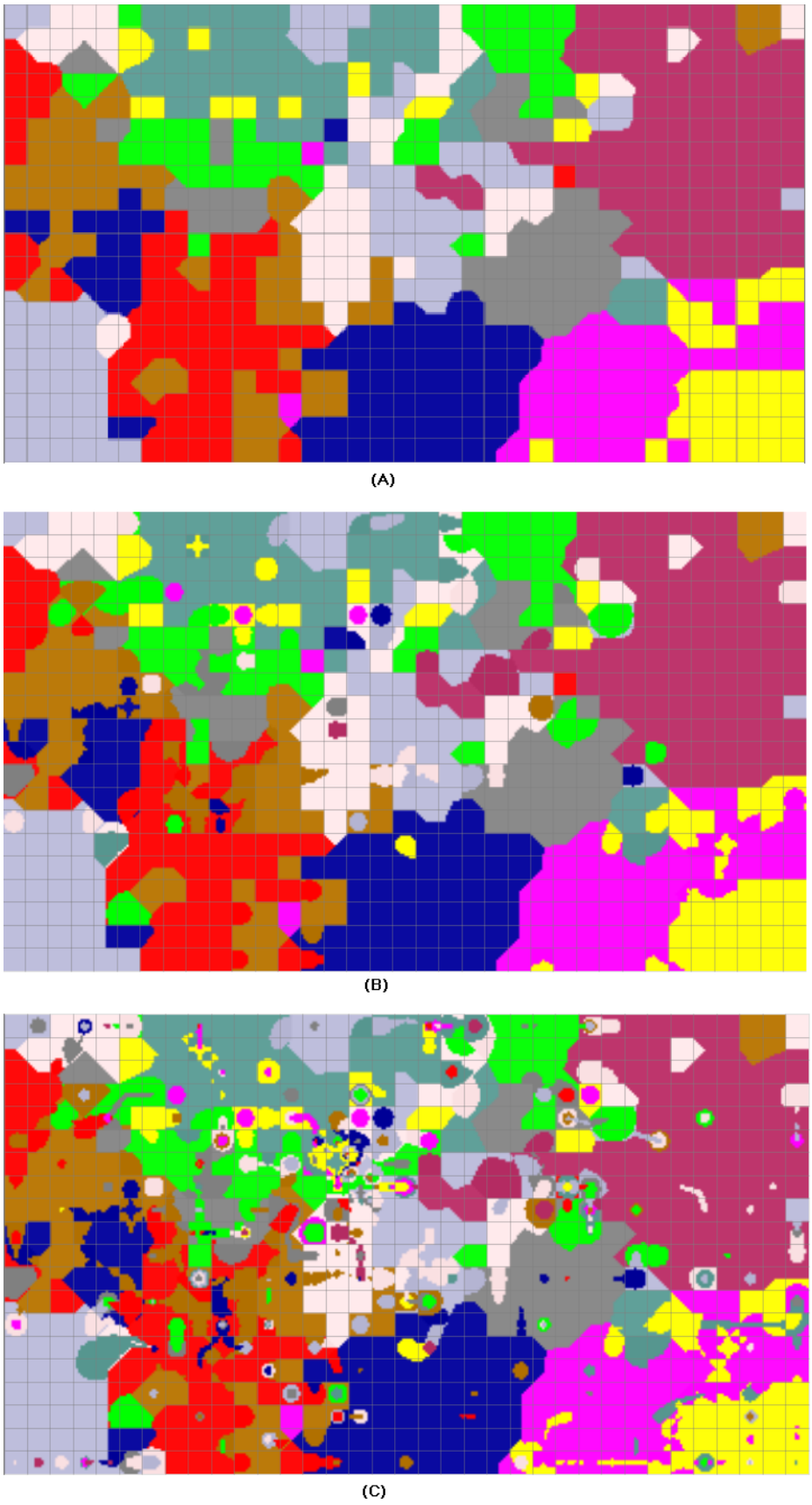


Figure 4.3: Class Coloring with different values of the parameter minimum visible class. (A)the parameter is set to 100%. (B)the parameter is set to 50%. (C)the parameter is set to 0%.

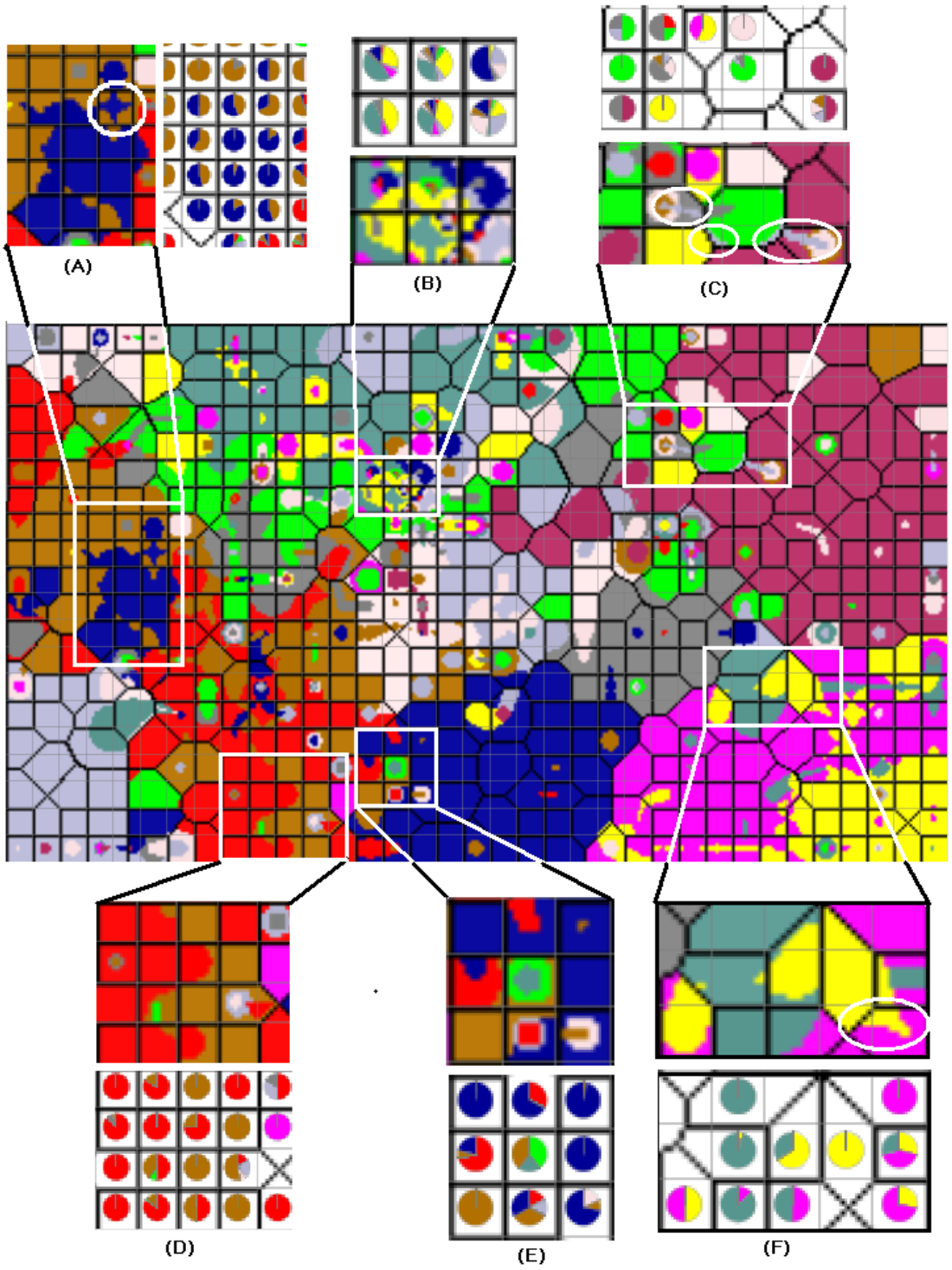


Figure 4.4: Class coloring with Voronoi cell border. (A) to (F) are significant samples showing some strengths and drawbacks of the visualization.

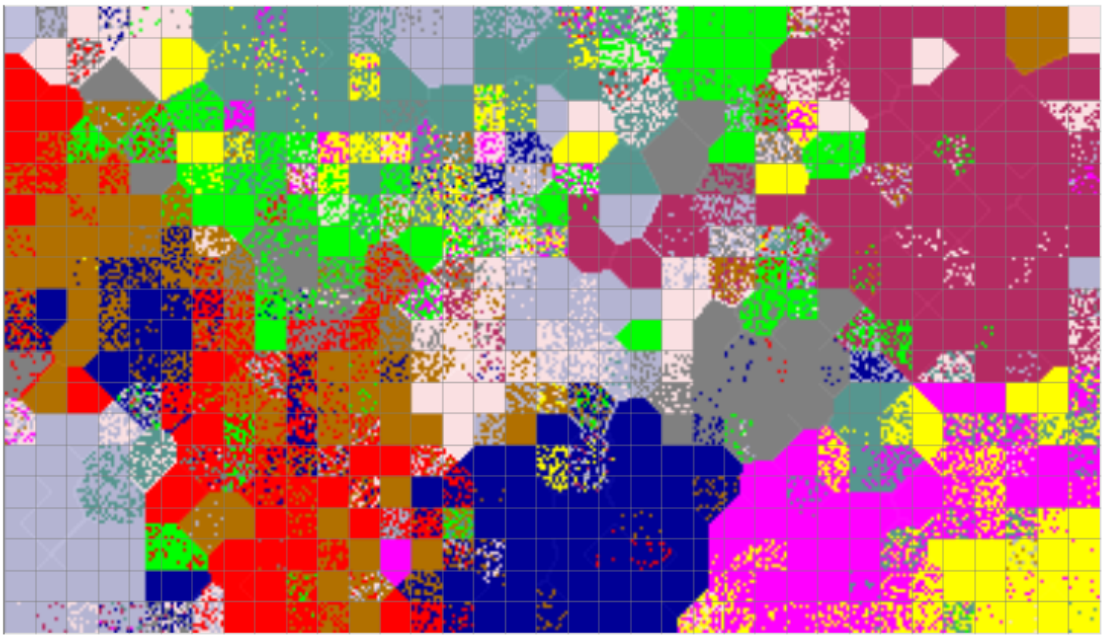


Figure 4.5: Chessboard visualization of the banksearch dataset.

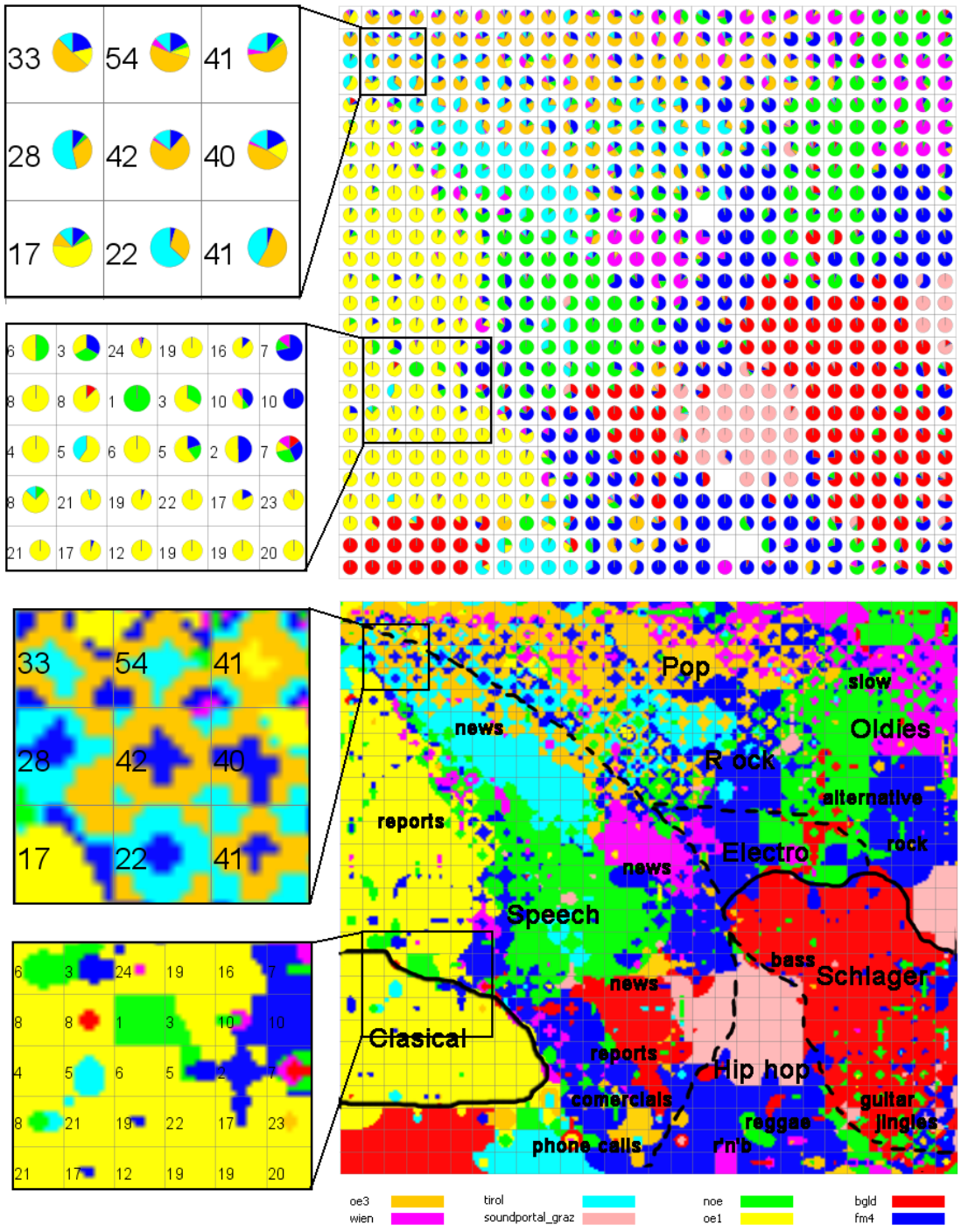


Figure 4.6: Radio station data set. (A) Pie chart visualization. (B) Class coloring of the data set. The text labels describe the main categories of the genres. See [LR06].

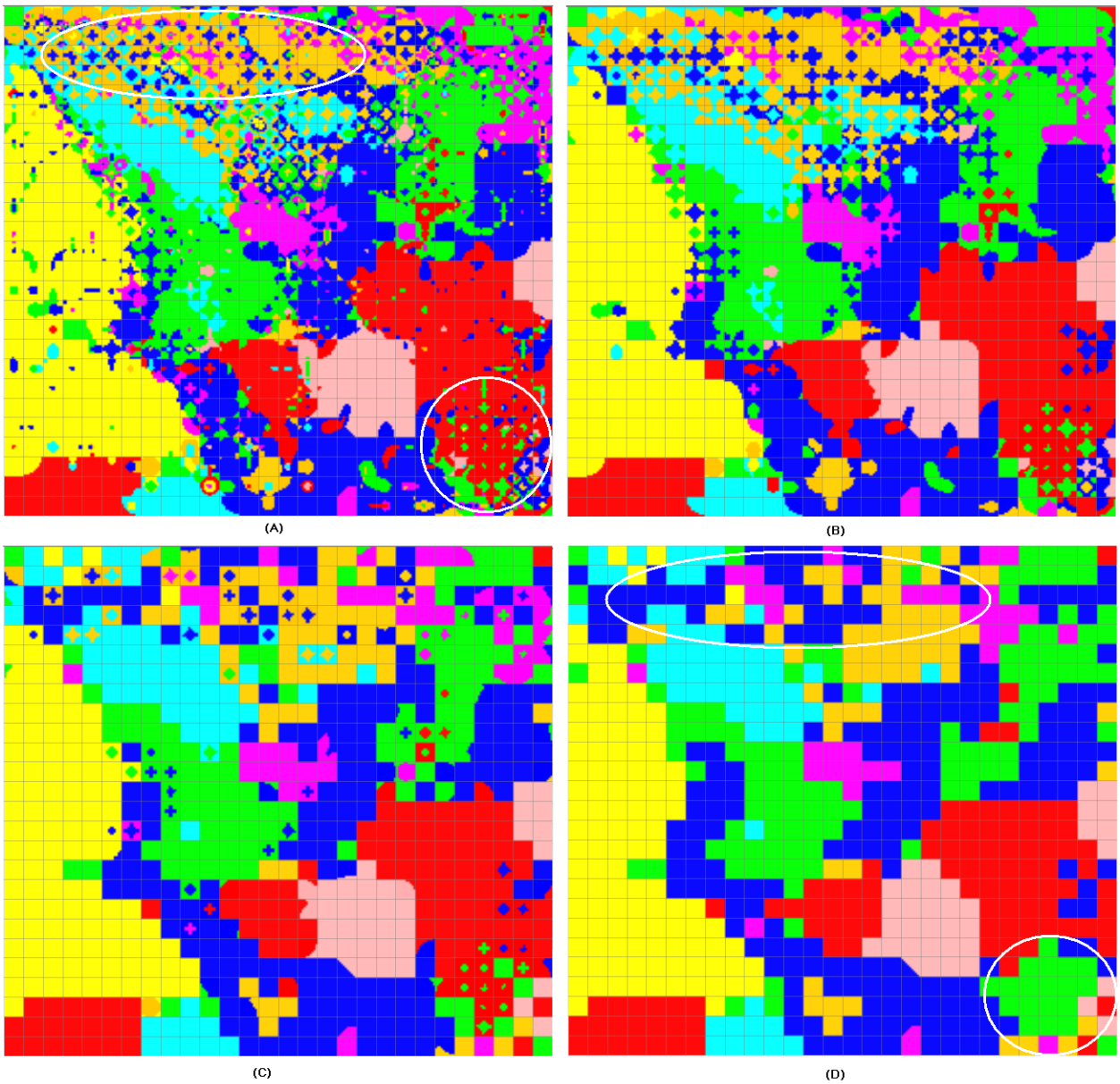


Figure 4.7: Class coloring of the radio search data set. The parameter minimum visible class is set to different values (A) 0% (B) 30% (C) 60% (D)100%. The dominant classes in (C) do not agree with these seen by the human eye in (A).

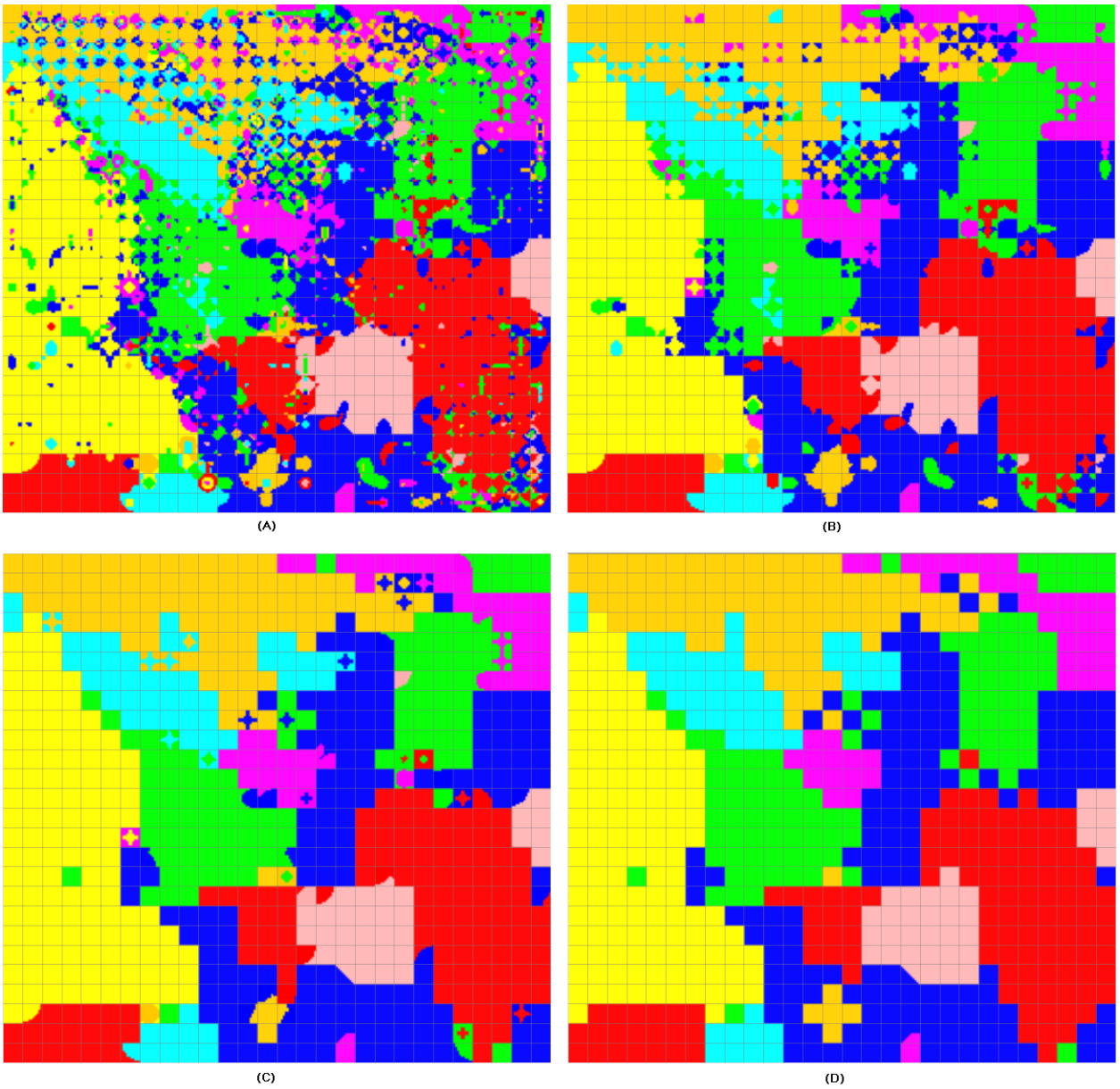


Figure 4.8: Class coloring of the radio search data set after the correction of the algorithm. The parameter minimum visible class is set to different values (A) 0% (B) 30% (C) 60% (D)100%. The dominant classes in (C) agree with these seen by the human eye in (A).

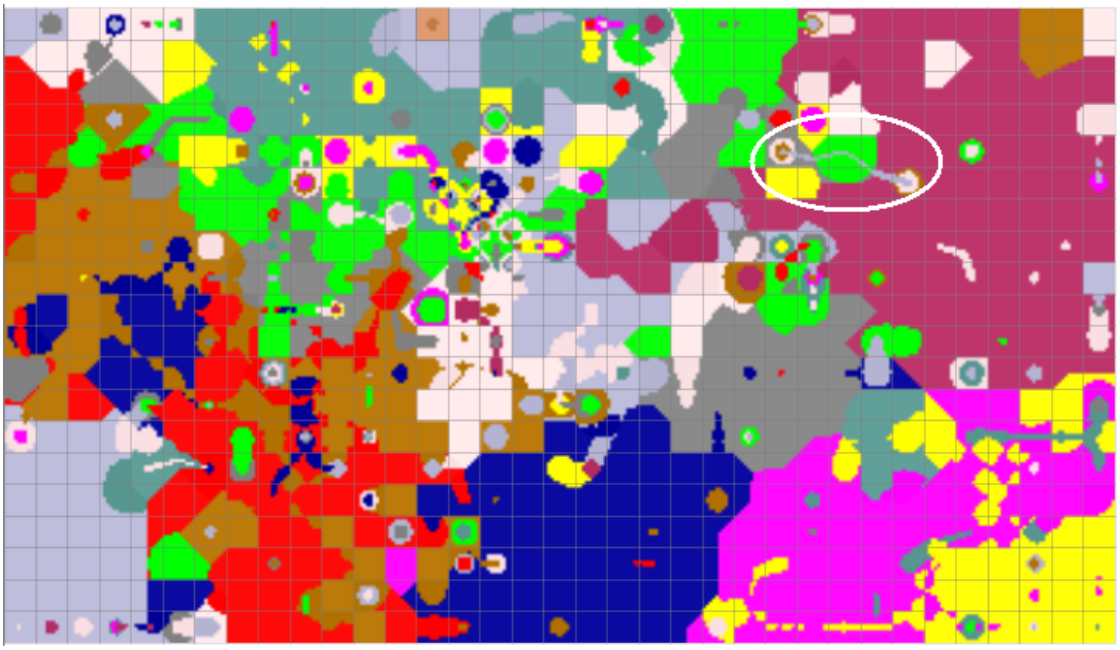


Figure 4.9: Class coloring of the banksearch data set after the correction of the algorithm. Some of the drawbacks that was found in the least section have disappeared after the correction. For example the area marked with a circle is to be compared with Figure 4.4-(C).

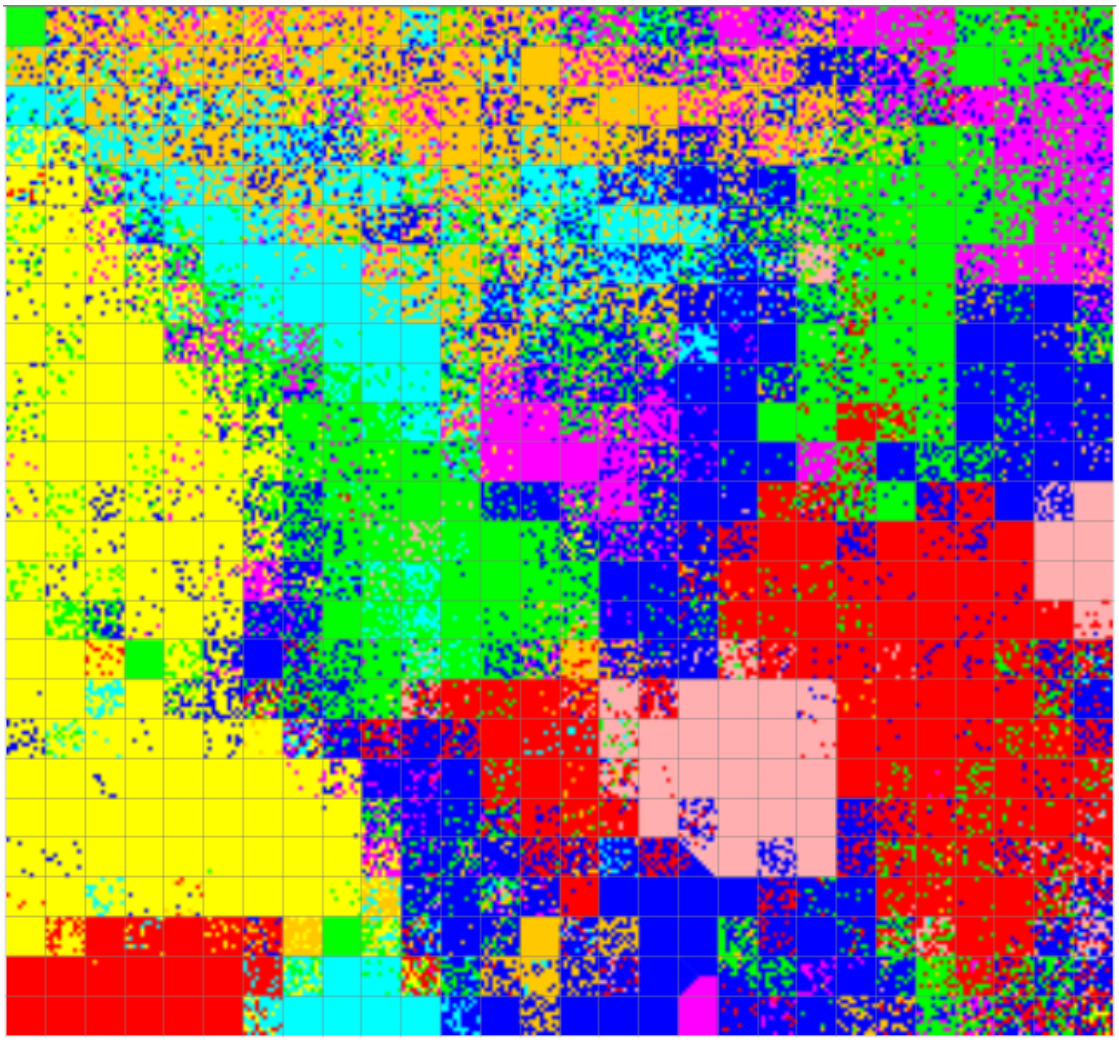


Figure 4.10: Chessboard visualization of the radio search data.

Chapter 5

Conclusions

Self-Organizing Map is a strong and useful tool for analyzing large and/or complex data sets especially those of high dimensionality. If the SOM is designed to be a tool for human use, then a SOM visualization is useful or sometimes necessary. SOM Visualizations are divided into two main categories: Visualization methods for unlabeled data, which are either visualizations that show the map in relation to the data set such as hit histograms or visualizations that are derived from the model vectors which aim at showing cluster structure and boundaries such as U-Matrix. The second category includes visualizations that assume the availability of labeled data and uses these (class) labels to produce a visualization, which shows the class topology of the data such as the pie chart visualization.

In this work a novel visualization method for visualization of labeled data is proposed namely the SOM class coloring method. This method aims to produce a colored partitioning of the SOM lattice depending on the class distribution in the units. The coloring process has two main steps: The first step is partitioning the SOM lattice by finding the Voronoi diagram having the unit positions as sites. The second step is partitioning each Voronoi cell separately depending on the classes in the corresponding unit and the classes in the neighboring cells. To achieve the second step, attractor functions are used: The attractor function is an algorithm, that attracts the colored pixels to a line segment. For partitioning a Voronoi cell with the attractor function, line segments with suitable lengths, directions, start and end points are selected, taking into account the classes in the unit and the classes in the neighboring units; finally the attractor function is

applied to these line segments and the corresponding classes.

An alternative method for coloring the cells is the chess board coloring, in which the Voronoi cell is divided into squires, which are colored according a uniform distribution function taking into account the contribution fractions for each class.

The class coloring method was tested with three different data sets, namely the iris data set, banksearch data set and an audio data set consisting of the broadcast of eight radio stations. The most of test showed satisfying results. Some drawbacks were found such as the problem of ambiguousness; that is more than one coloring is possible. The problem occurs if the class distribution is very interleaved.

Bibliography

[AJ99] P.J. Flynn Anil Jain, M.N. Murty. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, September 1999.

[AS92] Franz Aurenhammer and Otfried Schwarzkopf. A simple on-line randomized incremental algorithm for computing higher order voronoi diagrams. In *Annual Symposium on Computational Geometry, Proceedings of the seventh annual symposium on Computational geometry*, pages 363–381. ACM Press, New York, NY, USA, 1992.

[Aup03] Michael Aupetit. High-dimensional labeled data analysis with gabriel graphs. In *Proceedings Intl. European Symp. on Artificial Neural Networks (ESANN'03)*, Bruges, Belgium, 2003. Dside publications.

[Dev98] Olivier Devillers. Improved incremental randomized delaunay triangulation. In *Proceedings 14th Annu. ACM Sympos. Computer Geometry*, pages 106–115, 1998.

[For86] Steven Fortune. A sweepline algorithm for voronoi diagrams. In *SCG 86: Proceedings of the second annual symposium on Computational geometry*, pages 313–322, New York, NY, USA, 1986. ACM Press, New York, NY, USA.

[For97] Steven Fortune. Voronoi diagrams and delaunay triangulations. pages 377–388, 1997.

[Kas97] Samuel Kaski. Data exploration using self-organizing maps. *Acta Polytechnica Scandinavica, Mathematics, Computing and Management in Engineering Series No. 82*, March 1997.

[Koh97] Teuvo Kohonen. Exploration of very large databases by self-organizing maps. In *Proceedings of ICNN'97, International Conference on Neural Networks*, pages PL1–PL6. IEEE Service Center, Piscataway, NJ, 1997.

[LR06] Thomas Lidy and Andreas Rauber. Visually profiling radio stations. In *Proceedings of the 7th International Conference on Music Information Retrieval (ISMIR 2006)*, 10 2006.

[PDR] Georg Pölzlbauer, Michael Dittenbach, and Andreas Rauber. Advanced visualization of self-organizingmaps with vector fields.

[PRM02] Elias Pampalk, Andreas Rauber, and Dieter Merkl. Using smoothed data histograms for cluster visualization in self-organizing maps. In *Proceedings of the International Conference on Artifical Neural Networks (ICANN'02)*. Springer Lecture Notes in Computer Science, Madrid, Spain, 2002.

[RPM03] Andreas Rauber, Elias Pampalk, and Dieter Merkl. The som-enhanced jukebox organization and visualization of music collections based on perceptual models. *New Music Research*, 32(2):193–210, June 2003.

[Sam69] John Sammon. A nonlinear mapping for data structure analysis. In *IEEE Trans. Computer*, volume c-18, pages 401–409, May 1969.

[SC02] Mark Sinka and David Corne. A large benchmark dataset for web document clustering. In *Soft Computing Systems: Design, Management and Applications, Volume 87 of Frontiers in Artificial Intelligence and Applications*, pages 881–890, 2002.

[Sof] Softsurfer.com. Geometry algorithms. about lines and distance of a point to a line (2d 3d). See http://softsurfer.com/Archive/algorithm_0102/algorithm_0102.htm.

[Ult03a] Alfred Ultsch. Maps for the visualization of high dimensional data spaces. In *Proceedings Workshop on Self organizing Maps(WSOM'03), Kyushu*, Japan, 2003.

[Ult03b] Alfred Ultsch. Pareto density estimation: Probability density estimation for knowledge discovery. In *Proceedings Conf. Soc. for Information and Classification*, Cottbus, Germany, 2003.

[Ult05] Alfred Ultsch. Esom-maps: tools for clustering, visualization, and classification with emergent som. In *Technical Report 46, Dept. of Mathematics and Computer Science*, D-35032 Marburg, Germany, March 2005.

[VA00] Juha Vesanto and E. Alhoniemi. Clustering of the self-organizing map. *IEEE-NN*, 11(3):586, May 2000.

[Ves99] Juha Vesanto. SOM-based data visualization methods. *Intelligent-Data-Analysis*, 3:111–26, 1999.

[Yin03] Hujun Yin. Nonlinear multidimensional data projection and visualisation. In *Lecture Notes in Computer Science 2690*, Manchester, M60 1QD, UK, 2003.