Text Mining with Adaptive Neural Networks

eingereicht von:

Rudolf Mayer

Diplomarbeit

zur Erlangung des akademischen Grades
Magister rerum socialium oeconomicarumque
Magister der Sozial- und Wirtschaftswissenschaften
(Mag. rer. soc. oec.)

Fakultät für Wirtschaftswissenschaften und Informatik,
Universität Wien
Fakultät für Technische Naturwissenschaften und Informatik,
Technische Universität Wien

Studienrichtung: Wirtschaftsinformatik

Begutachter:
Univ.Doz. Dipl.-Ing. Dr.techn. Andreas Rauber

Wien, im *Februar 2004*

Abstract

Analysing high-dimensional data is a task where software tools can reasonably assist the data analyst, by visualising, and thereby uncovering, the inherent structure and topology of the data collection. Especially the kinds of tools that can produce results autonomously, i.e. **unsupervised** tools, are a goal; here, **neural network models** may be one solution. In the category of unsupervised neural network models, the ones based on the principles of the *Self-Organizing Map* have become quite popular.

We have tested the applicability of adaptive unsupervised neural network models, specifically of a model which was proposed just recently, the *Adaptive Hierarchical Incremental Grid Growing*, for free-text data in the *domain of tourism*: we have utilised the model to create a structured and hierarchically organised view of *Austrian hotels.* To be able to give a good analysis of the model's strength and weaknesses, we have furthermore compared the results with two other models, the standard Self-Organizing Map respectively the *Growing Grid*, and the *Growing Hierarchical Self-Organizing Map.*

Kurzdarstellung

Die Analyse hoch dimensionaler Daten ist eine Aufgabe, bei der softwaretechnische Hilfsmittel den Datananalysten durch Erzeugung einer Visualisierung der Struktur und Topologie der Datensammlung unterstützen können. Besonders eine Software, die diese Visualisierung autonom, d.h. **unüberwacht**, erzeugen kann, ist wünschenswert, und **neuronale Netzwerke** sind eine Lösungsmöglichkeit. Modelle, die auf den Prinzipien der *Self-Organizing Map* (Selbstorganisierende Merkmalskarte) beruhen, erfreuen sich größerer Beliebtheit.

Wir haben die Anwendbarkeit von adaptiven, unüberwachten neuronalen Netzwerken, und hier insbesondere eines erst kürzlich vorgestellten neuen Models, der *Adaptive Hierarchical Incremental Grid Growing* auf die Analyse mit Hinblick auf unstrukturierten Text aus dem Anwendungsbereich *Tourismus Informationssysteme*, untersucht. Das Model der AHIGG wurde zur Erstellung einer strukturierten, hierarchisch gegliederten Visualisierung von *Hotelbeschreibungen* verwendet. Um die Ergebnisse vergleich- und besser analysierbar zu machen, wurden ähnliche Visualisierungen mit einer Self-Organizing Map und einer *Growing Hierarchical Self-Organizing Map* erstellt.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the past few years, an increasing flood of digital information and data, from various different sources, became available to a wider range of users, ever more readily and easily. This trend is, besides others, perfectly supported by the rapid development of the Internet, particularly the *World Wide Web*: for the year 2002, there were about 8,712,000 unique web sites; out of these, 35% or 3,080,000 web sites were publicly available. Though the growth of the number of web sites is not as fast as it was in the late 90s, there was yet a significant growth of 20% between 2000 and 2002 [1]. Furthermore, so-called *Digital Libraries* offer massive document collections as well.

The quantity of data available increases rapidly, but that does not necessarily lead to the same quantity of information becoming retrievable: it gets very difficult for users to find the relevant piece of information they want in this flood of data. Therefore, methods supporting users with *organising*, *exploring* and *searching* features in collections of text data are needed.

Classical methods for searching documents by keywords exist; these methods may be enhanced with proximity search functions as well as keyword combinations according to Boole's algebra. Especially the keyword combination is currently widely used and well known to a large number of users, as it is used, for example, in so-called "search-engines" in the World Wide Web, as well as for libraries, etcetera. Other methods rely on document similarity measures based on a vector representation of the text documents. However, the visualisation of the results, most of the time being lists ordered by the keyword-relevance, has deficiencies in aiding the user to actually retrieve the desired documents.

A different approach is the *explorative* search, when there is a lack of exact keywords to guide the search process towards relevant information. This may

[1]all statistics taken from the "OCLC Online Computer Library Center, Inc.", http://wcp.oclc.org/stats/size.html, accessed on December 12th, 2002

be supported by structuring the data to visualise clusters, and by organising the data into hierarchies, which the users can "browse through" to find the information they are searching for. This approach is advantageous in that it is familiar to users: it is how libraries and bookshops are organised.

There are different approaches for organising data; among the most widely used, *cluster analysis* as a statistical method, based on pairwise similarities between documents, has to be mentioned. Hierarchical, tree-like structures can be generated to describe similarities between the documents, and to provide explorative searching. A discussion on these techniques can be found e.g. in [Sal89].

In the recent years, however, increasing attention has (again) been focused on neural network models, mainly because they became available at reasonable costs, with increasing computational speed. Neural network models had their origin in attempts to model the functioning of the human brain; the first models were already introduced in the 1940s. Neural networks may be utilised to model non-deterministic learning processes. Their architecture in general consists of a number of *neurons* as processing units, which are connected by *synapses*; neurons are normally *activated* by receiving input activation from their neighbours. This architecture can be modelled using a *graph*, where the neurons are represented by *nodes*, and the synapses by the *edges* between the nodes.

Three categories can be distinguished for the way neural networks *learn* [Spe96]:

- Supervised learning: Here, the result from the network for a specific input is compared to the desired output, and the error between these two is minimised in subsequent learning iterations. This process can be compared to human learning, under the supervision of a teacher; the teacher gives feedback, and the pupil tries to minimise the error. The *Backpropagation Network* is a well known example for this kind of neural networks.

- Unsupervised learning: Here, the neural network is organising itself in subsequent learning iterations. Changes to the neurons (which reflect the learning process) are determined by a so-called *learning rate.*

- Reinforcement learning: this is a hybrid of supervised and unsupervised learning; the network is only notified whether the result is good or bad.

For clustering problems, especially unsupervised models are well suited, as changes in the document collections will appear quite often [Mer97] (for example, new documents on the world wide web or in digital libraries are added every single day in enormous quantities); in a supervised environment, one would have to revise the class structure frequently. Among unsupervised models, *Adaptive Resonance Theory* and the *Self-Organizing Map* have to be mentioned [Spe96]. Although the functioning of the Self-Organizing Map has not yet been theoretically proven [Bay95], this model, and variants of it, has enjoyed increasing popularity in the last few years.

In this thesis, we will focus on how to use neural network models to structure data and to organise it in hierarchies, and therefore allow explorative search as well as (shown, for example, in [KKL$^+$00]) a better visualisation of the results from a keyword-based search. Among the quite many different models proposed in the literature, we chose a rather new model, namely the *Adaptive Hierarchical Incremental Grid Growing* (AHIGG), first presented in [He01]. This model will be implemented and tested for usefulness with an example of real-world data. However, to give the reader a background on neural network models, as well as to argue and justify why the model of the AHIGG has been chosen over the many alternatives, we will start by describing and discussing neural network models that have inspired the development of the AHIGG. We will point out their weaknesses for our purpose, i.e. structuring and organisation into hierarchies of a large collection of unstructured text documents, in Chapter 2.

More specifically, after defining some basic characteristics and prerequisites that neural network models should fulfill for our purpose, we will take a closer look at the model that forms the basis and has inspired the development of many similar neural network models: the *Self-Organizing Map* (SOM), as first presented in [Koh82] and described in detail for example in [Koh90], in Section 2.1. This model allows a mapping from high-dimensional data to a lower dimensional representation, usually a two-dimensional grid. This two-dimensional representation is an easily human-readable one, thereby providing the user with a good visualisation of the complex input space, allowing to recognise the structure in the input space.

Though the SOM is a well analysed and widely used tool for Information Retrieval and Text Mining purposes, it lacks some features desired in our application: it is in general necessary to have an a-priori knowledge about the data to choose the right parameters for the two-dimensional map; furthermore, in spite of considerable speed-ups due to algorithmic short-cuts allowing the SOM to scale up to very large data sets [KKL$^+$00], computational complexity is still significant, and besides, the SOM lacks mechanisms for direct cluster visualisation and for organisation in hierarchies.

The *Growing Grid* [Fri95a] and the *Incremental Grid Growing* [BM93] will then be presented in Sections 2.2 and 2.3, respectively, as models addressing the problem of the need to have an a-priori knowledge of the structure of the data by replacing a grid with a fixed, predefined size (which is hard to determine) by a dynamically growing one. The Incremental Grid Growing uses a more flexible approach in developing the mapping by dropping the strictly rectangular structure, using an adaptive architecture instead; moreover, it proposes a way to visualise clusters directly in the architecture of the map by introducing the concept of *connections* between the elements of the map. Clusters become visible as a group of interconnected elements that are separated by the other elements forming different clusters, in the way that they have no connection between them. However, these models still lack mechanisms for creating a hierarchical structure.

Then, the *Growing Self-Organizing Map*, proposed in [AHS00], will be presented in Section 2.4 as a model having similar characteristics in the growing algorithm, also not limited to a completely filled rectangular shape. In addition, the authors propose a way of creating hierarchies, though the proposed approach is a manual one, by interactively applying the model again on subsets of the input data.

Then, two models allowing hierarchical structures to be created automatically will be presented in Sections 2.5 and 2.6. The first one, the *Hierarchical Feature Map*, proposed in [Mii90], builds upon using the Self-Organizing Map in its hierarchical layers. Therefore, the single maps on the different hierarchical layers suffer from the same shortcomings as the Self-Organizing Map itself, though the computational speed may be increased dramatically, resulting from breaking down the problem with the use of smaller maps. Furthermore, the Hierarchical Feature Map always builds a fully balanced hierarchical tree (i.e., all possible paths down the hierarchical layers have the same depth), a representation that will only fit into a uniformly distributed data collection - an assumption that cannot be made for real-world data. That is where the *Growing Hierarchical Self-Organizing Map* (GHSOM), presented e.g. in [RMD02], combining the principles of the architecture of the Growing Grid to generate a (not necessarily balanced) dynamically growing hierarchical representation of the input data, adds. Being based partially on the Growing Grid, some of the shortcomings also apply to the GHSOM.

Therefore, the *Adaptive Hierarchical Incremental Grid Growing* (AHIGG) was proposed in [He01], building on the Growing Hierarchical Self-Organizing Map, but rather using the Incremental Grid Growing for the individual maps instead of the Growing Grid. This new model of the AHIGG will be described and discussed in detail in Chapter 3.

Finally, the AHIGG has been implemented, and will be tested and discussed with respect to its usability and usefulness for document classification in Chapter 4. The AHIGG will be tested on three data sets: first, we will apply two *demo data sets*, the well known *animals*, containing 16 data items with 13 features, and t*zoo*, consisting of 100 data items with 20 distinct features, demo data sets, as an intuitive way to show that the AHIGG produces useful mappings. Using such a demo data set has the advantage that the clusters and hierarchies developed by the AHIGG may be easily evaluated, because the semantic, in this case the different classes of animals (mammals, birds, etcetera) are well known [Mer97].

Then, we will use data from the domain of tourist information services, a collection of free-text hotel descriptions, to test the AHIGG on a large, real-world collection of unstructured data. An approach to facilitate information retrieval in tourist information systems with free-text queries has been carried out e.g. in [DMB02]; however, this was a more conventional approach by transforming the free-text queries into SQL statements, which were carried out in turn on structured data. We, however, want to rely only on the unstructured, free-text data.

In order to use the neural network models presented in this thesis for creating a mapping from a high-dimensional input space to a two-dimensional grid, the various text documents have to be represented as histograms of its words, i.e. the free-text data has to be described in high-dimensional vectors of features [Mer97]. How this can be achieved will be dealt with in detail in the corresponding Section 4.4.1. The results from this application will be presented and a comparison to the Self-Organizing Map respectively the Growing Grid, and the Growing Hierarchical Self-Organizing Map will be carried out. The chapter will finish with a discussion of the results.

Chapter 5 will give some indications for future work, while a conclusion of our work will be presented in Chapter 6.

Chapter 2

Neural network models

We want to identify neural network models giving a topology-preserving mapping from a high-dimensional input space to a two-dimensional output space, keeping spatially close vectors in the input space V spatially close in the target space A. The model should create a visualisation of the inherent complex structure of the input data, allowing to give the user an insight into it. To support explorative search, our model would need to provide mechanisms to organise the mapping in a hierarchical structure; further, clusters should be easily visible and recognisable. The whole process should be automated, and no a-priori knowledge of the input data set should be necessary to get satisfying results. This can be formulated by defining the following prerequisites:

1. The mapping should be topology-preserving [Fri92]:
   - Input patterns that are spatially close in the input space V, i.e. are similar to each other, should also be mapped spatially close in the output space A.
   - Elements which are spatially close in A should have similar input patterns mapped on them.
   - Areas of a high density in V should be represented by a corresponding number of elements in A.
2. The model should (automatically) create a *hierarchical* mapping, corresponding to the hierarchies found in the input data.
3. The model should provide a (direct) way for visualising *clusters* in the input data.
4. The model should not require any *a-priori* knowledge of the structure of the input data, i.e. the grid sizes and hierarchical layers should be determined automatically.

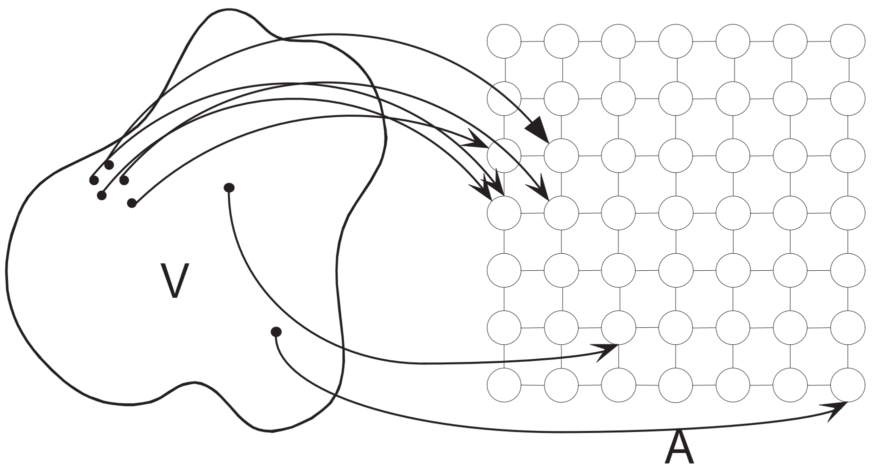


Figure 2.1: **Topology-preserving mapping**: Spatially close elements in V are spatially close in A as well.

5. The structure of the two-dimensional representation should not be limited to a completely filled rectangular shape (in other words, the representation should fit the input space, not the other way around).

The first prerequisite is illustrated in Figure 2.1: input patterns that are spatially close to each other in the input space will be spatially close to each other in the mapping. The high-density area in V is represented by correspondingly many spatially close elements in A.

All the models in this chapter fulfil these first three prerequisites; however, to the best of our knowledge, so far only the model described in Chapter 3, the *Adaptive Hierarchical Incremental Grid Growing*, fulfils ***all*** of them.

The rest of this chapter is organised as follows: first, we investigate the model which constitutes the "ancestor" of all the subsequently presented models: the *Self-Organizing Map* (Section 2.1). Building on this model, architectures that do not require (or require reduced) a-priori knowledge of the input data space are presented: the *Growing Grid* (Section 2.2), the *Incremental Grid Growing* (IGG, Section 2.3), as well as the *Growing Self-Organizing Map* (GSOM, Section 2.4). While the IGG and the GSOM also fulfil prerequisite 5, the IGG also allows direct cluster visualisation.

Then, models that allow the organisation of the map in hierarchies are presented: the *Hierarchical Feature Map* (Section 2.5), and the *Growing Hierarchical Self-Organizing Map* (Section 2.6), with the latter also fulfilling prerequisite 4 by determining grid sizes and hierarchies dynamically.

Also, models not using a two-dimensional output space, or rectangular grid structure (and therefore not providing an easy, direct visualisation), the Growing Cell Structures and the Neural Gas, will be examined in Section 2.7, as they propose some interesting mechanism for the growth process, as well as the connectivity and cluster visualisation.

2.1 The Self-Organizing Map

The Self-Organizing Map (SOM), as first presented in [Koh82] and described in detail e.g. in [Koh90], is a well known and widely used neural network model; it defines many principles that other models presented later in this chapter build on. As documented in the *SOM Bibliography* ([KKK98], [OKK03]), the SOM has been widely discussed and examined, and is, amongst other applications, widely used for visualisation of high-dimensional data, as it generates topology-preserving and dimensionality reducing maps. In general, the SOM consists of a two-dimensional regular grid of nodes, which is an easy, "human readable" representation. Each node on the grid is associated with a model of some observation, which is computed by the SOM algorithm so that they optimally describe the domain of observations. The SOM organises the models in a way that similar models are closer to each other than more dissimilar ones. In doing so, the SOM also provides a kind of clustering of the input data, a desired feature in Information Retrieval and Text Mining applications. However, cluster boundaries are not detected and visualised directly.

2.1.1 The Architecture

The SOM usually consists of a (rectangular or hexagonal) two-dimensional grid, with each node of the grid being associated with a model. These models m_i have the form of a so-called model vector $m_i = [m_{i1}, m_{i2}, ...m_{in}]^T \in \Re^n$ of the same dimension as the input vectors $x_i = [x_{i1}, x_{i2}, ...x_{in}]^T \in \Re^n$. To each of the nodes of the SOM, a number of input vectors x_i are assigned to during the training process, with similar vectors being mapped to the same node.

2.1.2 The Training Algorithm

The training algorithm consists of the following basic steps:

- Initialisation of the network

- A number of iterations of
  - Presenting input patterns and finding the best matching node
  - Adapting the model vectors of the best matching node and a certain number of neighbouring nodes.

These steps will now be described in detail.

Initialisation

For initialising the nodes in the grid, a common approach is to assign each node with a randomly generated model vector. Alternatively, randomly chosen vectors from the input data set could be taken.

Best matching node

A vector of the collection of input patterns is randomly selected, and *presented* to the SOM: we find the model (i.e. the node's model vector) which is most similar to a presented input vector x. As a measure for the similarity, the Euclidian distance between the model vector m_i and the input vector x can be taken. The node c which has the smallest Euclidian distance to the input vector, called the *winner*, is selected as the best matching node according to

$$c(x,t) = arg \min_i \{\|x(t) - m_i(t)\|\}. \tag{2.1}$$

Model vector adaptation

To improve the quality of the SOM, after each vector presentation, some model vectors of the SOM are adapted towards the input vector x in the *learning process.* The value of the new model vector is determined by its current value and two other factors, the *learning rate* α as well as the *neighbourhood function* h_{ci}, and can be computed according to

$$m_i(t+1) = m_i(t) + \alpha(t) \cdot h_{ci}(t)[x(t) - m_i(t)]. \tag{2.2}$$

The learning rate α, $0 < \alpha(t) < 1$, determines how much a vector is adapted, and should be a time-decreasing function, in other words, vectors should be adapted more in the beginning of the learning process, with this adaptation decreasing towards the end. The neighbourhood function is typically designed to be symmetric around the winning node; its task is to impose a spatial structure on the amount of model vector adaptation [Mer97].

For the neighbourhood function, there are mainly two different approaches to be found in the literature. The simpler one is by defining a *neighbourhood set* N_c, centred around the best matching node c. Nodes which lie inside this neighbourhood set are (all to the same degree) adapted according to the learning rate, while nodes outside the neighbourhood set are left as they are. Therefore, the neighbourhood function can be written as

$$h_{ci}(t) = \begin{cases} 1, \forall i \in N_c(t) \\ 0, \forall i \notin N_c(t) \end{cases} \tag{2.3}$$

This neighbourhood function is illustrated in Figure 2.2 (a), with the black-coloured nodes, lying within the neighbourhood set, going to be adapted, i.e. for these nodes, the neighbourhood function takes value 1. It is of advantage to have a time-variable width or radius of Nc, with Nc being very wide at the beginning of the training process, shrinking monotonically with time.

The second, more widely used approach for a neighbourhood function is the use of a Gaussian function. With r_i and r_c denoting the coordinates of the nodes i and c in the two-dimensional output space $\Re^2$, respectively, a proper form for h_{ci} might be

$$h_{ci}(t) = e^{\frac{\|r_i - r_c\|^2}{2 \cdot \sigma(t)}} \tag{2.4}$$

Analogous to the first approach for the neighbourhood, this function is decreasing with time t by the usage of a monotonically decreasing function $\sigma(t)$. Contrary to the simpler approach, using the Gaussian function will adapt the nodes' model vectors differently depending on their "distance" from the winning node c in the output space A. This is expressed by the term $\|r_i - r_c\|$, specifying e.g. the Euclidean distance. This behaviour is a desired one: nodes close to the winning node should be adapted more than nodes further away, as closer nodes mean closer relation. This neighbourhood function is illustrated in Figure 2.2 (b), where the black node is the winner, and the grey-shaded nodes are going to be adapted (the darker the node, the more it will be adapted).

Further information and a more detailed description of the Self-Organizing Map can be found for example in [Koh90].

2.1.3 Examples of SOM Applications

The Self-Organizing Map has been widely used for many different kinds of applications, e.g. for pattern recognition, image encoding, similarity recognition, etc. Some examples may be found in [Bay95] and [Spe96].

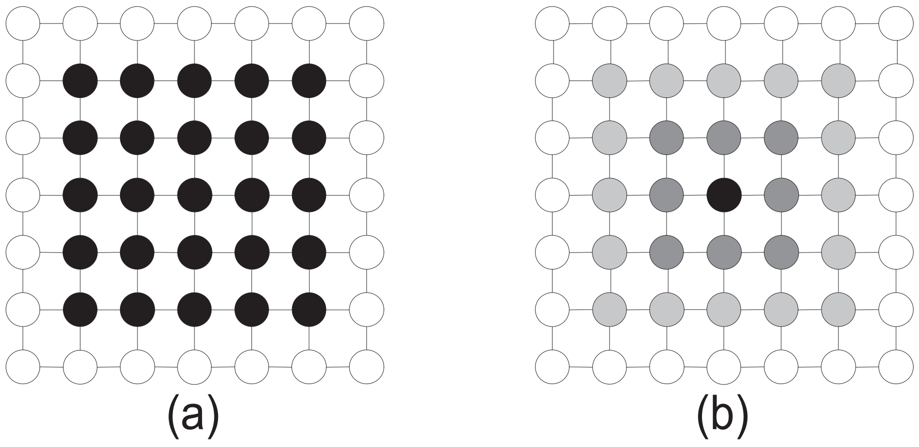


Figure 2.2: **Two variants for the neighbourhood adaptation**: In (a), all nodes will be adapted to the same degree, in (b) the adaptation strength is dependent on the distance from the winner.

Various projects dedicated to the organisation and visualisation of high-dimensional document collections have also been carried out. Among these, the two *WEBSOM* and the *SOMLib* projects are particulary interesting, as they provide (at least to some extent) methods for exploration of the data; they are described below.

WEBSOM

In the *WEBSOM* [Koh97] project, the usefulness of the Self-Organizing Map for the organisation and exploration of massive document collections was shown. A document collection of a total of 1,124,134 documents from 85 different *Usenet newsgroups* were organised in a map of $204 \times 510 = 104{,}040$ nodes, with 315 inputs each. In the final learning phase, the map was trained with 1,000,000 cycles, using techniques to speed-up the original SOM algorithm (some of these improvements are presented in Section 2.1.6). The WEBSOM was computed on general-purpose computers in tolerable time (eight to nine weeks).

The WEBSOM project showed successfully that SOMs can be used for large document collections; further, it provides a user interface for navigating through, and thereby exploring, the data, offering two different levels of "zoom" before displaying the list of documents represented by the selected area of the map[1]. Besides that, the WEBSOM project initially used an

[1]The WEBSOM project is accessible on the World Wide Web at

interesting way to form *word category histograms* of the documents by using another SOM for this purpose. For more details on this, please refer to [Koh97].

WEBSOM 2

In the *WEBSOM 2* [KKL$^+$00] project, an even bigger document collection than in the first WEBSOM project was organised: 6,840,568 patent abstracts were mapped as 500-dimensional document vectors on a SOM of 1,002,240 nodes. The algorithm was, similarly to the WEBSOM project, an improved version of the original SOM algorithm, utilising among others the techniques presented in Section 2.1.6; it took around six weeks to compute the final SOM.

The visualisation of the WEBSOM 2 is highly similar to the WEBSOM, offering mechanisms for submitting keyword searches as well as content addressable search queries (the user submits a longer text, e.g. another document, to search for similar content) and for exploring the data.

SOMLib

The *SOMLib* project [RM99a], [RM99b] is, as the name implies, a system for creating (digital) libraries based on the SOM. It tries to provide the users with an environment familiar to them from conventional libraries, i.e. categorising of the documents available; further, the user can interactively *explore* the library.

The application builds on SOMs, using an architecture of two layers of SOMs. First, there are individual SOMs for subsets of (independent) document collections. These maps are rather small, therefore allowing a satisfyingly fast application with the SOM algorithm; the maps are trained using the standard SOM algorithm. Secondly, there is another layer of a SOM that functions by *integrating* the smaller maps. This SOM, however, is not trained using the sum of all the sub-maps' input vectors, but rather by using the maps' model vectors as input vectors. Therefore, on this SOM, we will not find the documents directly, but rather the nodes representing these documents in another SOM. This approach is aimed for modelling distributed libraries, as only rarely will all the documents be located in one and the same place. By its architecture, the SOMLib allows SOMs representing document collections to be integrated into one common representation.

http://websom.hut.fi/websom/ (accessed on January 7th, 2004), including a demo-application of data-exploration

The SOMLib package further utilises the *LabelSOM* technique (see Section 2.1.5), and a component called *libViewer* to metaphorically visualise the resulting library [RB99].

2.1.4 Alternative Cluster Visualisation

The SOM has become a popular tool for organising high-dimensional data; therefore, a lot of research has been done to find improvements for the visualisation of clusters. Among the many approaches, two approaches proposed in [MR97] on cluster visualisation in standard Self-Organizing Maps are presented here; these methods are add-ons to the standard SOM, therefore preserving its robustness. The first approach mirrors the movement of a node's model vector in a virtual two-dimensional output space, while the second determines the degree of connectivity of neighbouring nodes, utilising the distances between their model vectors. These methods will be described in detail now.

Adaptive Coordinates

The basic learning rule of the SOM is extended to capture the movement of the various model vectors within the (virtual) two-dimensional output space: after each iteration, the *Adaptive Coordinates* (AC) of all but the winning nodes are moved towards the position of the winning node in the output space. Thus, clustering the nodes around the winning node resembles the clustering of the model vectors around the input signal presented. After the convergence of the training process, the clusters can be visualised using the adaptive coordinates of the nodes to plot them in the virtual output space. This is illustrated in Figure 2.3: While (a) shows the standard representation of a SOM, the same SOM is visualised in Adaptive Coordinates representation in (b). Clusters become clearly visible by similar nodes being grouped closely to each other.

For more details on the Adaptive Coordinates, refer to [MR97].

Cluster Connections

Cluster Connections (CC) is a technique based on post-processing information contained in the model vectors of a trained SOM: the distances between model vectors of neighbouring nodes are used to determine three types of relationships between nodes: *Highly similar nodes*, *medium similar nodes* (nodes that still belong to the same cluster), as well as *dissimilar nodes*, i.e. nodes with a cluster boundary going right in between them. A visualisation

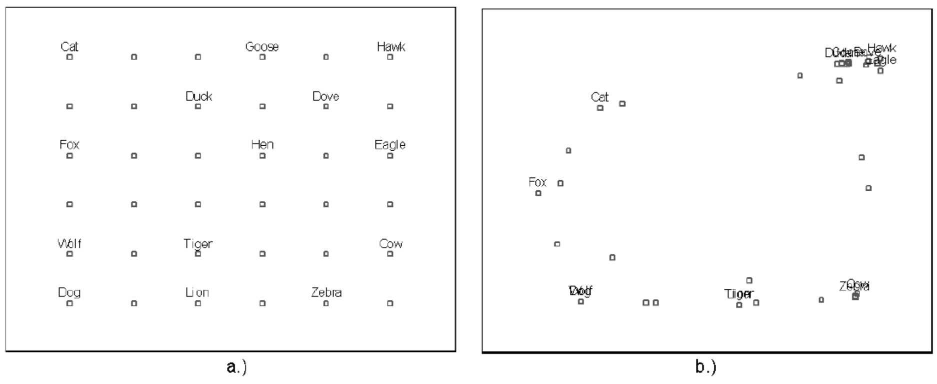


Figure 2.3: **A** 6 × 6 **Self-Organizing Map** in (a) standard representation and (b) *Adaptive Coordinates* representation

example with the same trained SOM as in Figure 2.3 (a) is presented in Figure 2.4: Here, clusters become visible by the non-existence of a connection; furthermore the degree of similarity between nodes is indicated by the darkness of the connection symbols.

For more details on the Cluster Connections, please refer to [MR97].

2.1.5 Labelling a SOM

Though using the techniques presented in Section 2.1.4, or similar ones, clusters in the SOM become visible and intra cluster relations information is extracted, still the relevance of attributes of the input patterns for a cluster, i.e. the characteristics of this cluster, are not extracted. For the majority of SOMs so far, the labelling was a manual process; this approach is suitable for small maps, where there is a knowledge about the input data. However, for large maps and unknown data characteristics, it is unfeasible. Also approaches to label the SOM with the label of the input data are only feasible when the label of the input patterns actually carry some information about their characteristics. In the WEBSOM project (see Section 2.1.3) for instance, the name of the use-groups was used for labelling. However, in many cases, information like this will not be available.

One method to automatically assign the nodes in a SOM with labels that give some descriptive information about the clusters is the *LabelSOM* method [RM01]. It builds on the fact that the most descriptive attributes for a set of input data are the ones shared by all data on a specific node. If a majority of input patterns mapped on a particular node exhibit a highly

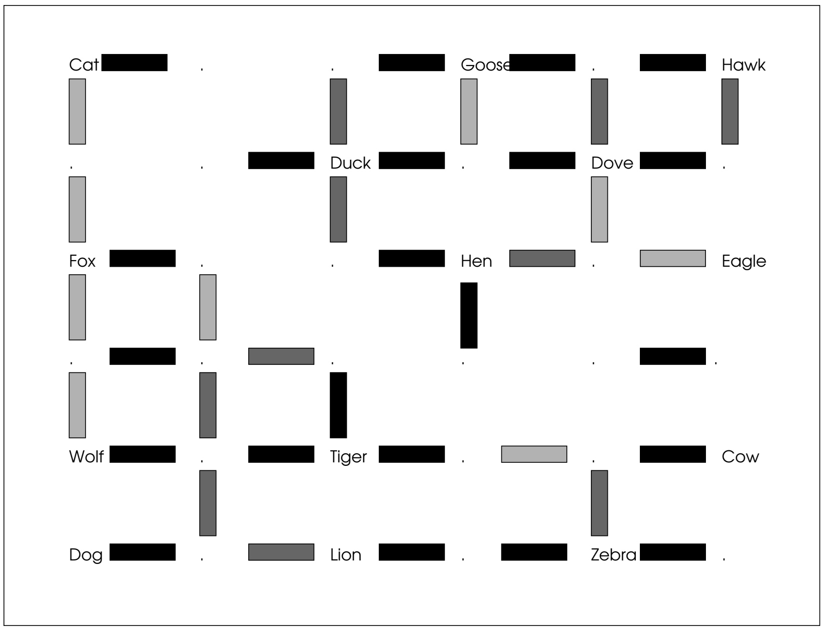


Figure 2.4: **A** 6×6 **SOM in Cluster Connections representation**: Cluster boundaries as well as the degree of similarity become visible.

similar input vector value for a particular feature, the same will apply for the corresponding model vectors; therefore, those model vector elements that show largely the same value for all input patterns mapped on a node may serve as a descriptor for that very node. Determining these elements is based on computing the quantisation error q_i for each vector element k of a node i according to:

$$q_{i_k} = \frac{1}{\|C_i\|} \times \sum_{x_j \in C_i} \|(m_{i_k} - x_{j_k}\|, \tag{2.5}$$

where C_i is the set of input patterns $x_j \in \Re^n$ mapped on node i, whose model vector is m_i. The smaller the value, the higher the similarity between all input vectors; a value of 0 would denote all nodes sharing the same value. [RM01] proposes to either select as many attributes as a predefined number, or to use a *preciseness threshold*, selecting attributes with a quantisation error below that value.

It is to noted that with this approach, labels that either describe the shared *presence* or the *absence* of a specific attribute in a cluster are determined; however, in text mining applications, we want to describe a cluster of documents by its present features only, in other words, we do not want to describe documents by saying what they are *not* about. Therefore, besides requiring a low quantisation error, we also demand an attribute to have high model vector value, which indicates high importance: it is suggested to define another threshold indicating the *minimum importance*, and to select only those attributes with a model vector value above this threshold.

2.1.6 SOM Algorithm Improvements

In recent years, a lot of improvements to speed-up the original algorithm of the Self-Organizing Map have been proposed; out of these, some of the suggestions from the *WEBSOM* [Koh97] and *WEBSOM 2* [KKL+00] projects will be presented here.

Rapid construction of large maps

The idea behind this technique is to estimate *good* initial values for the model vectors of very large maps on the basis of asymptotic values of the model vectors of a (much) smaller map. In other words, a smaller SOM is trained with the same input patterns, and then interpolated to the larger map. This is based on the idea that with good initial values, the training process can be run with much fewer iterations and much smaller neighbourhood range and adaptation strength. Thus, the process is much faster in total. This technique was applied in the WEBSOM 2 project, where a small 435 nodes map was, in four successive stages, enlarged 2,304 times, to develop a large map of 1,002,240 nodes.

Rapid fine-tuning of large maps

- Addressing old winners: It is assumed that in the middle of the training process, when the SOM is smoothly ordered, though yet not asymptotically stable, the model vectors are not changed much during one iteration. When the same input pattern is presented to the SOM some time later, the new winner is found at, or in the vicinity of, the old winner. This can be utilised as follows: To each input pattern, a pointer to the old winner is stored. When an input pattern is presented again, the node referenced will be searched first, and then a local search in the neighbourhood will suffice to find this new winner. As a precaution, when the new winner is at the edge of the neighbourhood, this will be only a preliminary best match, around which a new neighbourhood is built and searched. A regular full search for the winner can be performed intermittently to ensure globally best matches. This mechanism is, according to [KKL+00] a significantly faster operation than

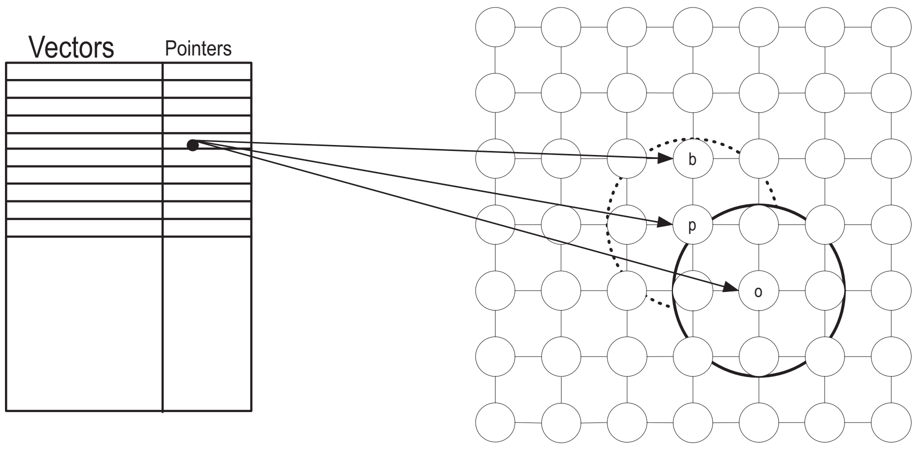


Figure 2.5: **Fast winner search** in the Self-Organizing Map algorithm

the exhaustive winner search over the whole SOM.

A possible scenario is depicted in Figure 2.5: For the selected input, node o is stored as the old winner. In o's neighbourhood, p is found as the best matching node; however, p is located at the edge of the neighbourhood, and will therefore be regarded as a preliminary best match only. Therefore, a new neighbourhood is built (and searched) around this node. A node b is found as the final best match in the new neighbourhood.

- Batch map principle of the SOM: The incremental algorithm of the SOM, as defined by Equation 2.1 and 2.2, can often be replaced by a significantly faster batch computation version, which moreover does not require the specification of any learning rate $\alpha(t)$. This technique is based on the assumption that the SOM will converge to some ordered state, and therefore, the expectation values of $m_i(t+1)$ and $m_i(t)$ for $t \rightarrow \infty$ are required to be equal, i.e. in the stationary state,

$$\Delta m_i = m_i(t) \cdot h_{ci}(t)[x - m_i^*] = 0 \tag{2.6}$$

holds true. For simplicity, $h_{ci}(t)$ is regarded as being time-invariant, and with a finite number (batch) of x(t), Equation 2.6 is written as

$$m_i^* = \frac{\sum_t h_{ci} \cdot x(t)}{\sum_t h_{ci}}. \tag{2.7}$$

However, this is not an explicit solution for m_i^*, as on the right, h_{ci} still depends on $x(t)$ and all m_i^*. Thus, starting with an approximation for the m_i^*, Equation 2.1 is utilised to find the indexes c(x) for all the x(t). Then, Equation 2.7 and 2.1 are applied iteratively, and after a few cycles, stable solutions for the m_i^* are found. For a more detailed discussion of the batch map principle of the SOM, refer to [KKL$^+$00] or [Koh98].

2.1.7 Discussion

The Self-Organizing Map fulfills the first prerequisite defined for our application as defined in Section 2: With the neighbourhood adaptation, similar input patterns are guaranteed to be spatially close on the grid.

The SOM neural network model has numerous advantages: It provides a robust algorithm for a mapping from high-dimensional data to a lower-dimensional (usually a two-dimensional) grid while preserving the topology of the input space. The developed map is an easy human-readable visualisation of the complex input data. The Self-Organizing Map has been proved to be suited for applications in the Information Retrieval and Data Mining area; some examples of different applications of the SOM can be found in Section 2.1.3.

Still, there are also a number of disadvantages. The architecture of the SOM does not allow *clusters* to be easily detected, as it is for example possible in the *Incremental Grid Growing*, presented in Section 2.3. However, a wealth of methods has been developed to provide better cluster visualisation of the SOM; two of these methods, presented in [MR97], are described in Section 2.1.4. Also, methods like the *LabelSOM* [RM01] (see Section 2.1.5) allow easier interpretation of the mapping.

Another problem lies in the inflexible architecture of the SOM. With a fixed size of the grid, one needs to know what size might be optimal for mapping a given input data set, which requires some knowledge of the input data. This knowledge will, in many cases, not be present, and gaining it through experiments for each single case is not desirable. Therefore, a model that automatically, during runtime, decides about its architecture, depending on the input patterns, is desired. Finally, the architecture of the SOM does not allow any hierarchical structuring of the input patterns.

Further, the SOM poses some constraints on the size of the document collection that can feasibly be displayed on a single two-dimensional map [RDM00].

Implementation issues

Implementation of the architecture and the training algorithm is relatively simple, and well known. Also, various publicly available implementations have been released, e.g. the public-domain software package *SOM_PAK*, described in [KHKL96][2].

However, for a *large* set of input patterns, a large grid of nodes will be necessary for an adequate mapping, resulting in an increased computational effort. Therefore, the SOM does not seem to be well suited for organising large data sets in a short time. Yet, a lot of improvements of the original algorithm have been proposed; some of them are described in detail in Section 2.1.6.

Conclusion

Though all applications presented in Section 2.1.3 show the usefulness of the SOM for text mining purposes, the computational speed, and especially the lack of organisation into hierarchies does not make it attractive for our purposes; we will rather use a hierarchical neural network model.

2.2 Growing Grid

The *Growing Grid*, as presented in [Fri95a], is a self-organising network generated by a growth process. The application range of the model is the same as that of the *Self-Organizing Map* presented in Section 2.1, and can therefore also be used for data visualisation. The Growing Grid overcomes one of the shortcomings of the SOM: the need to predefine the size of the grid, by growing dynamically in size.

2.2.1 The Architecture

Like the SOM, the Growing Grid is a network consisting of a rectangular, $k \times m$ grid A of nodes; like in the SOM, each node has a model vector. Additionally, however, each node has a *resource* variable τ_c, which is used to gather statistical information to decide where to expand the grid.

[2]This software package is available at http://www.cis.hut.fi/research/som_lvq_pak.shtml (accessed on December 10th, 2003)

2.2.2 The Training Algorithm

The training algorithm of the Growing Grid incorporates the two basic steps of the SOM algorithm, i.e. presenting input patterns and finding the best matching node, as well as adapting the model vectors of the best matching nodes and a certain number of neighbouring nodes. Additionally, the grid is expanded when certain conditions are fulfilled, with the learning process being terminated when a stopping criterion comes true. For initialisation, the resource values t_c are set to 0.

Best matching node and model vector adaptation

Analogous to the SOM, for each input pattern x, the node c with the most similar model vector is determined according to Equation 2.1.

Adaptation of the model vectors towards x is done on the winning node c, as well as on its neighbouring nodes. Rather than using the Euclidean distance as described in Section 2.1.2, the measure known as *city-block distance* or L_1-*norm* is used to calculate the distance d between two nodes $c_1 = a_{i_1 j_1}, c_2 = a_{i_2 j_2}$ in the output space A:

$$d(c_1, c_2) = \|(i_1 - i_2\| + \|j_1 - j_2\|. \tag{2.8}$$

With α_0 being a constant learning rate and σ a constant neighbourhood-width parameter, model vectors in the Growing Grid are adapted according to

$$m_i(t+1) = m_i(t) + \alpha_0 \cdot e^{-\frac{d^2(c,i)}{2 \cdot \sigma^2}} \cdot (x - m_i)(\forall i \in A), \tag{2.9}$$

with the exponential term defining a Gaussian neighbourhood adaptation as in Equation 2.4.

With each adaptation step, the resource variable τ_c of the best matching node is increased by one, thereby acting as a winning counter.

Grid expansion

A parameter λ_g determines the number of adaptation steps by $k \times m \times \lambda_g$ for a network of the size of $k \times m$ nodes. In other words, the parameter indicates how many adaptation steps per node are done on average before expanding the grid.

After $k \times m \times \lambda_g$ adaptation steps, the node e with the maximum resource value, i.e. the node which has the highest number of best matchings, is determined (this node is often called the *error node*). With the aim of distributing the input patterns more evenly over all nodes, new rows or columns are inserted in the vicinity of this node. Assuming that a neighbour f of e, where

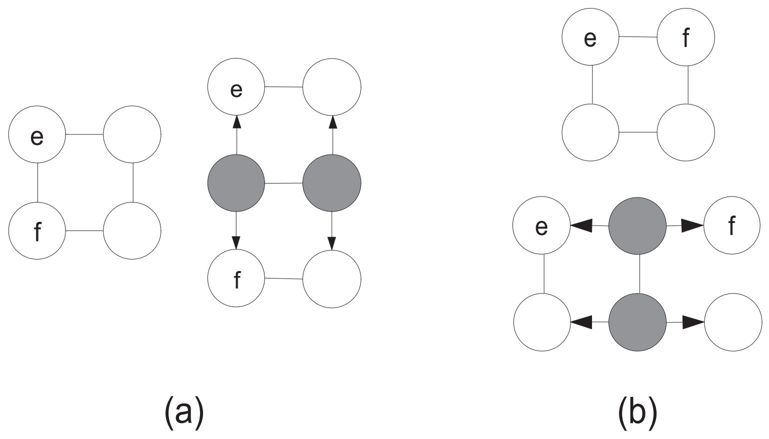


Figure 2.6: **Inserting new nodes in the Growing Grid**: Insertion of a row (a) or column (b) between the nodes e and f.

f has the most different model vector to e (indicating a direction with high variance), the resolution of the grid should be increased in that direction; this is done by inserting a new row or column, respectively, between f and e.

The above can be formally described as follows: with N_e denoting the set of neighbouring nodes directly connected with e, according to

$$N_e = \{c \in A | d(e, c) = 1\}, \tag{2.10}$$

the neighbouring node to e with the most different model vector fulfils

$$\|m_f - m_e\| \geq \|m_i - m_e\| (\forall i \in N_e). \tag{2.11}$$

It is assumed that e and f are neighbours in the i-th row of A, particulary with $e = a_{ij}$ and $f = a_{ij+1}$. Then, a new column j' (having k nodes) is inserted between the columns j and $j + 1$. Initial values for the model vectors of the new nodes are derived by interpolating from model vectors of their neighbours, thereby, as intended, increasing the density of model vectors in the vicinity of m_e:

$$m_{rj'} = \frac{1}{2} \cdot (m_{rj} + m_{rj+1}) (\forall r | 1 \leq r \leq k). \tag{2.12}$$

By this, the number of columns increases by one: $m = m + 1$.

The case that e and f share the same column can be treated in complete analogy. These two variants of inserting new nodes are described in Figure

2.6, with (a) showing the case of inserting a new row between the error node e and its most different neighbour f; in (b), a new column is added.

After growing the grid as described above, all resource variables $\tau_i, \forall i \in A$ are reset to zero, and a new round of adaptation is started, until the stopping criterion, as described below, is fulfilled.

Stopping criterion

As the network's size is not predefined, it has to be decided during runtime when the growing process should be terminated. In the simplest case, a maximum number of nodes allowed on the grid can be defined, halting the growth process as soon as this maximum number is reached or surpassed.

However, an indirect stopping criterion might be desirable, as for a direct one (for example, the maximum number of nodes) similar criticism as with the predefined size of the Self-Organizing Map (see Section 2.1.7) might apply (though the number of nodes seems to be a less complex parameter as the actual grid-size, and thereby an improvement). An indirect stopping criterion might be e.g. when the fraction $\frac{\tau_i}{m \times k}$ falls below a boundary for each node $i \in A$, i.e. when at the most a certain fraction of the input patterns has been mapped to each node. However, attention has to be drawn to designing a stopping criterion that will eventually be fulfilled, or to combine it with a maximum number of nodes allowed.

Fine-tuning of vector positions

The constant learning rate chosen for the adaptation during the learning process prevents the convergence of the model vectors to near-optimal positions. However, once the growth process is finished, the model vectors can be fine-tuned with a number of adaptation steps, using a decreasing learning rate.

With t' denoting the time in the fine-tuning phase, and λ_f determining how many adaptation steps per node are performed on average, $t'_{max} = k \times m \times \lambda_f$ adaptation steps are performed according to Equation 2.9, with a time-dependent learning rate $\alpha(t')$, according to

$$\alpha(t') = \alpha_0 (\frac{\alpha_1}{\alpha_0})^{\frac{t'}{t'_{max}}}. \tag{2.13}$$

After the fine-tuning phase, the training process is finished, and the network has reached its final structure.

2.2.3 Discussion

The Growing Grid builds on the principles of the Self-Organizing Map, having a two-dimensional rectangular network structure and a similar, but slightly enhanced architecture and training algorithm; the size of the grid is not predefined, but the best matching size for the given input patterns is automatically determined during the training process. The grid is expanded in areas with a high density of input patterns, thereby improving the representation quality in these areas.

The Growing Grid deals with one of the major shortcomings of the SOM, where one had to have some a-priori information about the collection of input pattern to be able to predefine a fitting grid size. With the grid growing in size, a well defined stopping criterion can fulfil the task for nearly any collection of input patterns.

The usefulness of the Growing Grid for data visualisation has been shown for example in [Fri96] for a demo data set, and in [Mer98a] for software libraries.

On the other hand, the Growing Grid's expansion process always inserts complete rows or columns respectively, therefore also increasing the density of model vectors in areas of the map where nodes may represent their input space adequately. This problem cannot be solved with an architecture where the grid's rows and columns are completely filled, but for example with an approach as in the Incremental Grid Growing (Section 2.3).

Furthermore, the Growing Grid, as the SOM, does not provide any mechanism for directly visualising the clusters (as it is incorporated in the Incremental Grid Growing, presented in Section 2.3) or for creating hierarchies (as the models going to be presented in sections 2.5 and 2.6).

2.3 Incremental Grid Growing

The *Incremental Grid Growing* (IGG), as described in [BM93], tries to deal with some shortcomings of the Self-Organizing Map: it addresses the problem of predefining the grid size by an algorithm that automatically chooses a grid size and shape fitting to the input patterns during run-time. Furthermore, it proposes a solution how to visualise clusters by using the concept of *connections* between nodes; clusters become visible by areas of connected nodes, being separated from other areas simply by not having a connection to them. Plus, the IGG proposes how to deal with discontinuities in the input space, which in the standard SOM architecture may lead to having nodes in areas where input probability is 0; this is solved by having the grid not fully filled with nodes, but having some grid positions left empty.

To develop an accurate representation of the topology, the algorithm must prevent erroneous encoding of the topology, which (at least complete prevention) is impossible without an a-priori knowledge of the input space. Alternatively, the algorithm could recognise and correct such errors, though fully organising a map and then modifying it so that unwanted structures are removed requires too much computational effort. Therefore, an incremental growing algorithm combining both approaches, with heuristics how to remove connections and for adding new nodes, was chosen.

2.3.1 The Architecture

The Incremental Grid Growing uses a two-dimensional grid of nodes as the SOM does, but drops the idea of having this grid fully connected to visualise cluster boundaries, as well as the idea of having nodes placed on all grid-positions. In other words, the grid of the IGG (likely) becomes sparse. The grid starts from a few nodes, and is expanded during the training process.

2.3.2 The Training Algorithm

After initialisation, the algorithm is run in a number of iterations, each consisting of the following steps:

- Presenting vectors and adapting the grid to the input patterns, using a slightly modified self-organising process of the SOM.
- Expanding the network by adding nodes to those areas (at the perimeter of the grid) that represent their corresponding input space inadequately.
- Deleting or adding connections between nodes by examining their model vectors.

The new network structure is re-organised; the process continues until a stopping criterion, in the form of a maximum number of nodes, is fulfilled. The steps of the algorithm will be described in detail below:

Initialisation

Initially, the network consists of four connected nodes in a 2×2 grid. The model vectors of these nodes are chosen randomly from the input, as well as the *error values*, as described below, are set to 0.

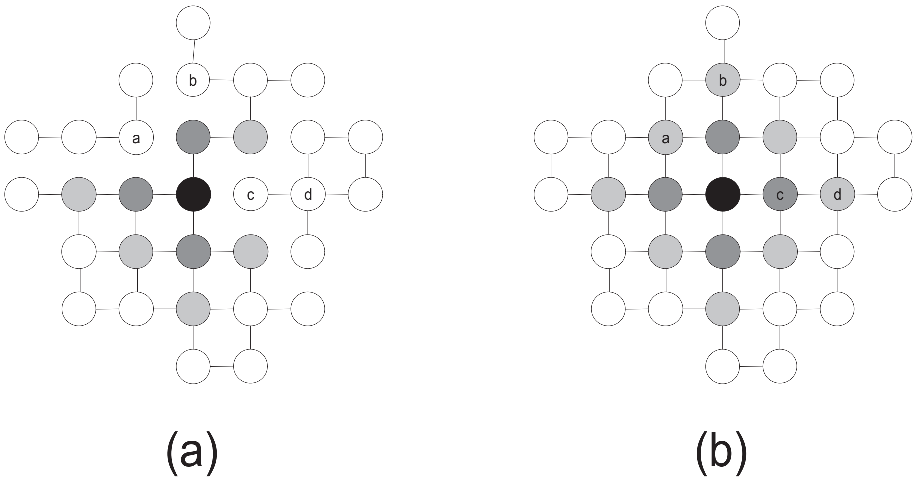


Figure 2.7: **Neighbourhood in the Incremental Grid Growing**: connectivity as the decisive criterion for neighbourhood adaptation of the nodes *a* -*d*.

Best matching node and model vector adaptation

Analogous to the SOM, input patterns are presented to the network, the best matching node is chosen, and the nodes' model vector are adapted towards the presented input pattern - the network develops a two-dimensional mapping of the high dimensional data (see Section 2.1.2 for details on the SOM training algorithm).

Note, though, that there is one important difference for the adaptation of neighbouring nodes: in this model, whether a node will be adapted or not does not only depend on its distance to the winning node, but also on whether there is a connection between these two nodes. Also, the distance is not measured as a distance between the positions on the grid, but as the length of any existing shortest path between these two nodes. This is illustrated in Figure 2.7: Both (a) and (b) represent the same network, except for the connections. In (a), the node *b* is not adapted, due to the length of its *shortest path* to the winning node being outside the neighbourhood range. The nodes *a, c* and *d* are **not** adapted, because they have no connection to the winning node (marked in black); in (b), however, these nodes have a connection to the winner (respectively, a length of a shortest connection to the winner within the neighbourhood range), and therefore are adapted.

Additionally, so-called *boundary nodes*, with a boundary node defined as any node in the grid that has at least one directly neighbouring position not

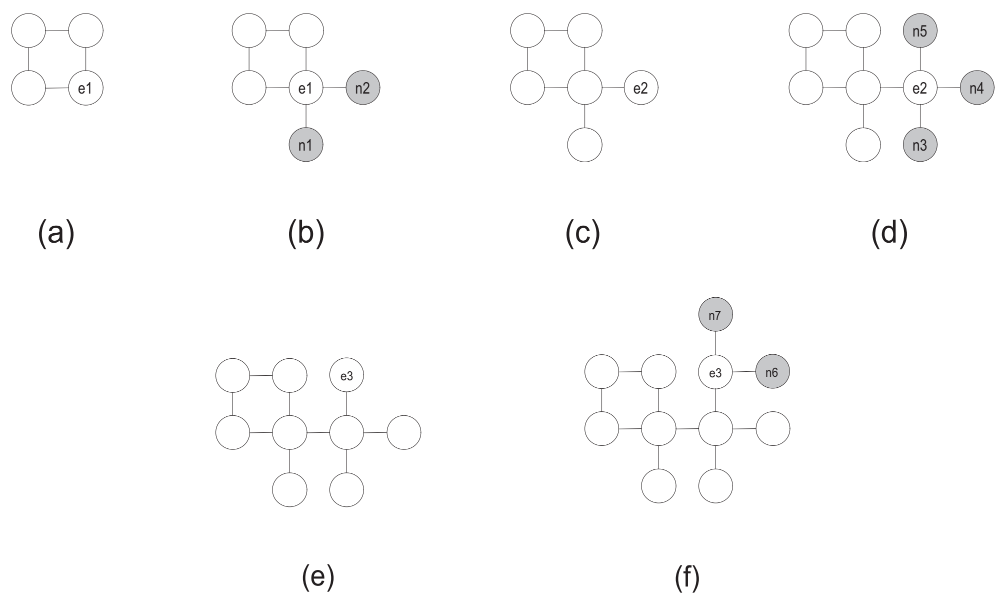


Figure 2.8: **Expanding the grid in the Incremental Grid Growing**: growing new nodes around the error nodes.

yet occupied by a node, store a special *error value* E. Each time a boundary node is selected as a winning node during the input pattern presentation, this error value E is increased by the square of the distance between the node's model vector m and the input pattern x according to

$$E(t) = E(t-1) + \sum_k (x_k - m_k)^2, \tag{2.14}$$

where k is the k-th component of the feature vector.

Grid expansion

At the end of each training cycle (a training cycle consists of a number of iterations of adaptation), the node with the highest cumulative error, the *error node*, determined by the highest value of E among all boundary nodes, is said to be the node representing the area of the input space most inadequately. Therefore, the grid is expanded around this node, by adding new nodes in all as yet unoccupied grid locations in its direct neighbourhood. The new nodes are initialised being directly connected to the error node.

In Figure 2.8, a scenario of the growth process is illustrated: The IGG starts with an initial 2×2 grid in (a). After the first training cycle, the node

$e1$ is selected as the error node, and new nodes ($n1$ and $n2$) are added in open neighbouring positions of $e1$ (b). After the expansion, another training cycle is carried out, and $e2$ is selected as the new error node (c), and new nodes $n3$, $n4$ and $n5$ are added in (d). After a third cycle, with $e3$ as the error node (e), and new nodes $n6$ and $n7$ added, the grid has grown almost three times in size.

For the initial model vector, two cases are distinguished for a newly added node:

- The node has, besides the error node, other already existing direct neighbours (which are not necessarily, and not possibly, connected to the new node). Then, the value for the model vector is initialised as the average value of all neighbouring nodes' model vectors according to
$$m_{new,k} = \frac{1}{\sum_{i \in N} m_{i,k}} \tag{2.15}$$
with $m_{new,k}$ being the k-th component of the new node's model vector, and N being the set of the n neighbouring nodes. In Figure 2.8 (d), this case applies for the nodes $n3$ and $n5$.

- The newly added node has no other direct neighbours. In this case, the new node's model vector is initialised to the error node's model vector; the error node's model vector is modified to the average of its existing neighbouring nodes' and the new node's model vectors, according to
$$m_{e,k} = \frac{1}{n_e + 1} \times (m_{new,k} + \sum_{i \in N_e} m_{i,k}) \tag{2.16}$$
with $m_{e,k}$ being the k-th component of the model vector of the error node, and N_e the set of n_e existing neighbours of the error node.

  This case is illustrated for the nodes $n1$ and $n2$ in Figure 2.8 (a).

The training process is terminated when a stopping criterion, given by a predetermined maximum number of nodes, is fulfilled.

Initially, new nodes are connected to the rest of the network structure only through the error node. This might change in subsequent iterations, though, as described below.

Examining connections

During the self-organising process, nodes may develop model vectors that are close to the model vectors of nodes they are not connected with, whereas on

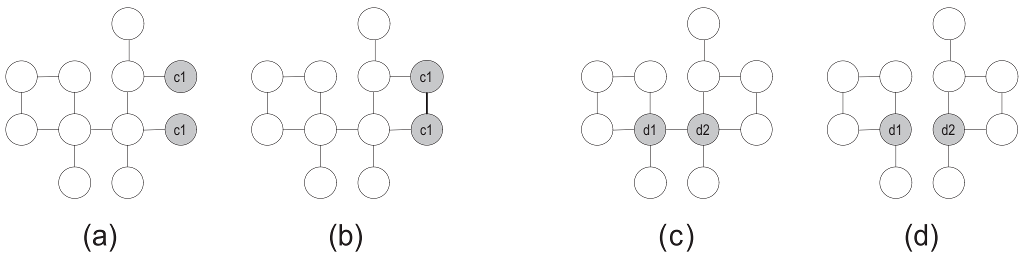


Figure 2.9: **Connectivity in the Incremental Grid Growing**: in (a), two similar nodes establish a connection, whereas in (b), two dissimilar nodes are disconnected.

the other hand, a node's model vector might become significantly different to a model vector of a neighbouring node with which it is connected. In both cases, it would be desirable to change the connection-state between these two nodes. For this, a "connect" and a "disconnect" threshold parameter is defined. After each training cycle, the grid is examined by comparing each pair of two directly neighbouring nodes by calculating the Euclidean distance between their model vectors. The following cases can be distinguished:

- If there is no connection between the nodes so far, but the distance is below the connect-threshold, a new connection is added. This is illustrated in Figure 2.9: the two nodes $c1$ and $c2$ have developed similar model vectors (a), and therefore a new connection is established between them (b).
- If there is already a connection between the nodes, and the distance between the two nodes' model vectors is above the disconnect-threshold (in other words, the two nodes have become too different), the connection is deleted. This is depicted in Figure 2.9 (c), where the two nodes $d1$ and $d2$ are identified to be too dissimilar, and therefore the connection between them is detected (d). Note that by this, two separate clusters have been formed.
- In all other cases, the state of the connection remains the same.

2.3.3 Discussion

The Incremental Grid Growing addresses and (partly) solves two major shortcomings of the Self-Organizing Map: the need to have an a-priori knowledge about the input data to be able to specify the grid-size parameters is reduced, as defining the maximum number of nodes requires less information about the input space as the need to define also the width and height of the grid as in the SOM, but is still not completely solved. A computational measure would be desirable.

The inability to visualise clusters with the SOM however is solved with the concept of connections. Connections in the IGG present the input distribution - if it forms a connected area, it will be rarely necessary to delete connections from the grid, on the other hand, if there are distinct clusters in the input patterns, (and if the thresholds are selected properly), these clusters will develop into separate clusters in the mapping as well. Once a cluster is separated from the rest of the network, it will continue to develop separately (as neighbourhood adaptation will be done only on, and resulting from, nodes within the very same cluster). These clusters may represent categories or sub-sets in the data; capturing these structures mapped in the two-dimensional map can assist in the interpretation of the input data. However, selecting *absolute* values for the thresholds, given an unknown data set, might be hardly possible. This issue is addressed in the *Adaptive Hierarchical Incremental Grid Growing* in Section 3.2.2. Further, inter-cluster relations are not visible anymore. This, however, could be solved by applying a visualisation technique like the Cluster Connections as described in Section 2.1.4.

Growing the network only at its perimeter allows it to develop an arbitrary topology, while expanding only around areas that represent the input space inadequately encourages the map to develop only topological structures that are actually present in the input data.

Though the IGG seems to provide solutions to some of the shortcomings in the SOM, it has, to our best knowledge, been rarely used in applications with real world data. In [BM95], the IGG has been tested upon data from the *Webster online thesaurus*; in [KM98], the IGG has been compared to the Growing Grid in matters of pattern recognition, with the authors stating that it produces slightly better results, which might be due to the target-oriented strategy of adding nodes at those edges of the grid where the quantisation error is at its largest.

On the other hand, the Incremental Grid Growing model does not provide any mechanism for creating hierarchies. Therefore, we will investigate hierarchical models in Sections 2.5 and 2.6.

Implementation issues

For the computational effort, arguments analogous to the ones presented in the sections about the Growing Self-Organizing Map (2.4.4) would apply: the computational effort is decreased due to the incremental growth process, resulting in a smaller grid for most of the training process. However, this has not yet been proved in experiments.

2.4 Growing Self-Organizing Map

The Growing Self-Organizing Map (GSOM), as presented in [AHS00], is a model similar to the Incremental Grid Growing presented in Section 2.3. The same as the IGG, the network grid starts small in size, growing during an incremental process, with the growth being only at the perimeter of the grid, and (likely) leaving the grid sparse in some areas.

There are important differences to the IGG though, mainly with the GSOM not having the concept of *connections* (i.e., the grid is fully connected, each node has a connection to its direct neighbours). Further, the initialisation of newly created nodes follows a more sophisticated rule. The stopping criterion for the growth process again is a user-defined parameter, but, instead of using the number of maximum nodes allowed as in the IGG, the user specifies a more abstract "spread factor" (described in detail in Section 2.4.3), which will determine the degree of the map's spreading, and thereby the size of the network.

2.4.1 The Architecture

The GSOM uses a rectangular two-dimensional grid for its mapping. The grid is, however, not necessarily (and not likely) completely filled with nodes - during the training process, empty grid positions may emerge. All nodes, however, are fully connected to each other.

2.4.2 The Training Algorithm

The GSOM algorithm builds on the basic SOM concepts extended by some steps, and can be structured in three phases: Initialisation, growing, and smoothing phase. The algorithm will be described in detail below.

Initialisation

As with the GG and IGG, the network is initialised with four nodes, mainly because it allows all starting nodes to be *boundary nodes* (as defined in Section 2.3.2), thus each node has the possibility to grow in its own direction at the beginning. The model vectors are initialised with random values from the input vector space.

A variable E_{max}, tracking the highest accumulated error value E of a node in the network, is initialised to 0. The *growth threshold gt*, which will guide the growth process of the map, will be calculated from the user-defined *spread factor*.

Best matching node and model vector adaptation

Analogous to the SOM, input patterns are presented, and the best matching node is found as the node having the least Euclidean distance to the presented input vector. Adaptation of nodes also follows the principle of the SOM, using a time-decreasing learning rate and neighbourhood range. The winner's *error value E* will be increased by the difference between the input vector and the model vector similar to Equation 2.14 of the Incremental Grid Growing.

However, the neighbourhood is smaller compared to the SOM, to have a more localised model vector adaptation. Also, the learning rate holds an interesting new concept, which is justified by the incremental growth process: at the very beginning of the training process, the grid is small (four nodes in general), allowing the same nodes to be selected as winner for *very* different input vectors. This will cause the model vectors of the same set of nodes to fluctuate in completely different directions. As a solution, a new learning rate reduction rule, taking into account the number of current nodes in the network is proposed: $\alpha(t)$ is the learning rate at time t, according to:

$$\alpha(t+1) = \beta \times \psi(n) \times \alpha(t), \tag{2.17}$$

where $0 < \beta < 1$ is a constant value, causing Equation 2.17 to converge towards 0 for $t \to \infty$, and $\psi(n)$ is a function of n, the number of nodes in the network; this function is used to manipulate the learning rate in a way that it is smaller at the beginning (when there are fewer nodes), and gradually taking higher values as the network expands and the number of nodes increases. As a simple formula, $\psi(n) = 1 - \frac{R}{n(t)}$, with the authors using $R = 3.8$ in their experiments, is proposed.

Grid expansion

This phase is also similar to the corresponding one in the IGG presented in Section 2.3.2 - the growth takes place only at the perimeter of the network. A difference is found, though, on how the growth is initiated, which is the case when any node has a higher error value than the growth threshold, i.e. when the condition $E_{max} > gt$ is fulfilled. All nodes for which the error value E is higher than the growth threshold will grow new nodes in all free neighbouring grid positions. Model vector initialisation is based on the model vectors of the existing neighbours of the new nodes, and can be divided in four different cases, depending on the layout of the neighbourhood (for a detailed description, refer to [AHS00]).

With the growth being triggered by the boundary nodes itself, cases might arise where the boundary nodes have low error values, but a non-boundary node (i.e. a node in the centre of the network, where the grid is completely filled) has a high error value. Though the non-boundary node can not grow new nodes, and the boundary nodes do not trigger the growth (having a low error value, i.e. as they represent their input space adequately), the grid should still be expanded. Note that in the original SOM, this problem would not arise when the grid size is big enough, as the organisation process will result in the map spreading out.

To solve this problem, a concept of *error distribution* is proposed: a non-boundary node c for which the error value E fulfils $E > gt$ will distribute its error to the neighbouring nodes with the following formulas:

$$E^c_{t+1} = \frac{gt}{2} \tag{2.18}$$

where E^c_t is the error value of the node, and

$$E^{n_i}_{t+1} = E^{n_i}_t + \gamma \cdot E^{n_i}_t \tag{2.19}$$

where $E^{n_i}_t$ (i can take the values 1-4) is the error value of the i-th neighbour of c, and γ is a constant value controlling the increase in the error accumulation, called the *factor of distribution.* With 2.18 and 2.19, the error value of the error node is reduced, and the error value of the neighbouring nodes is increased. Thereby, the error is spread out from the error node, which shall eventually in time (i.e. iterations) ripple outwards and cause boundary nodes to grow. In other words, goal of these two formulas is to (indirectly) initiate the node growth on the perimeter of the grid.

2.4.3 Spread Factor

As mentioned above, the growth process is guided by a growth threshold, which in turn is calculated from a user-defined *spread factor* sf, $0 < sf < 1$. The spread factor determines the degree of spreading of the map, with a higher value resulting in a more spread out map. The abstraction from the growth threshold to the spread factor sf liberates the analyst of the need to have a-priori knowledge of the factors that determine the growth in the algorithm, as the same spread factor can be used to achieve the same degree of granularity for different input data. The basic idea behind the sf is that the growth threshold will depend mainly on the dimensionality of the data set; the higher the dimension, the higher the error values, in general, will become. To liberate the user from defining different growth thresholds for different dimensional data, the very same is calculated according to:

$$gt = D \times f(sf), \tag{2.20}$$

where D is the dimension of the data, and $f(sf)$ a function that should take values 0 to ∞. $-ln(sf)$ is a function that satisfies this requirement; the growth threshold can thereby be calculated according to:

$$gt = -D \cdot ln(sf). \tag{2.21}$$

2.4.4 Discussion

The GSOM, though having differences in the algorithm, has similar characteristics as the IGG - both do not need a-priori knowledge of the analyst to specify the size, with the GSOM, using the spread factor, being more flexible than the IGG, and de facto requiring no a-priori knowledge at all. However, the GSOM does not allow cluster detection as direct as the IGG with its concept of connections. For cluster detection, the authors stress the fact that the flexible structure and shape of the grid itself will give some hints (the GSOM will "branch out", which will attract attention to groups as well as "outliers"). The concept of the learning rate reduction for small grids is an interesting one, and could be incorporated in other, incrementally growing models.

Another interesting aspect is that the GSOM, as the first of the models presented so far, suggests a method how to create hierarchical structures. This is done by creating a GSOM with a low *spread factor*, and taking only subsets of this map to generate a new GSOM with a higher spread factor. This could subsequently be done a few more times, thus actually creating a set of hierarchies. However, this process (selecting the subset, and starting a new process with a higher spread factor) is done manually and therefore time-consuming.

One more interesting concept is proposed (but not yet incorporated in the model) in [AHS00]: extending the learning process by mechanisms to delete nodes from the grid, when they become obsolete.

In [AHS00], the GSOM has, however, been tested only on a demo data set, as well as a rather small *human genetic* data set, describing genetic distances between 42 different populations of the world (i.e., the data set contains 42 input patterns).

Implementation issues

The algorithm of the GSOM becomes more complex than the one of the SOM, especially the aspect of the error distribution and the more sophisticated model vector initialisation for newly added nodes might take additional computational effort. However, the authors argue that the speed compared to the SOM is increased, especially due to the fact that the GSOM can represent a set of data with fewer nodes than the SOM, at an equal amount of spread, due to its flexible shape. The authors maintain that this aspect becomes a significant advantage especially for very large data sets, as the reduction of nodes will result in a reduced processing time. However, to the best of our knowledge, this algorithm speed-up has not yet been verified in experiments.

2.5 Hierarchical Feature Map

The Hierarchical Feature Map (HFM), as presented in [Mii90], is the first of the models presented so far that allows (automatic) creation of a hierarchical structure, thereby explicitly visualising the hierarchical taxonomy of the input patterns. To achieve this, the Hierarchical Feature Map uses several standard, two-dimensional Self-Organizing Maps, arranged in several hierarchical layers. The layers are arranged in a way that for each node in one layer, a self-organising map is added in the next layer. The highest level is supposed to lay out the different categories of input patterns, while on lower levels the self-organisation process will form sub-categories and groups.

2.5.1 The Architecture

The Hierarchical Feature Map consists of several layers of Self-Organizing Maps. The HFM starts with a single SOM on the first hierarchical layer.

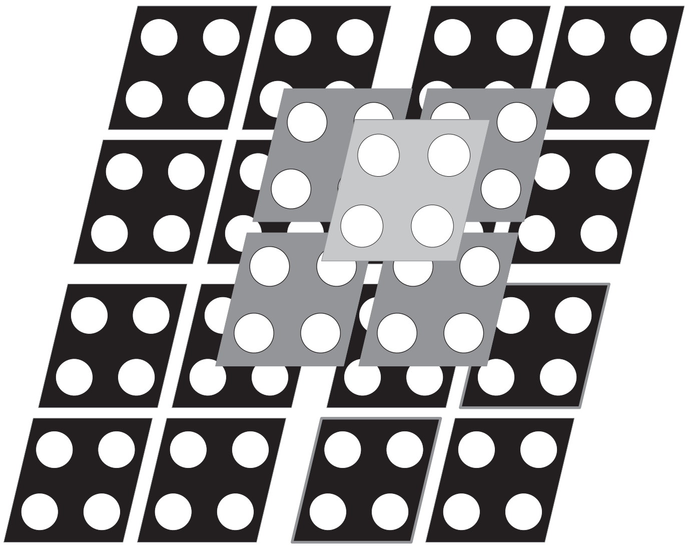


Figure 2.10: **Architecture of a three layered Hierarchical Feature Map**

For all the other hierarchical layers, which will be created in an incremental process, there are precisely as many SOMs as there are nodes in all the SOMs on the previous layer. If the SOM on layer 1 is a 2×2 SOM, then there are four different SOMs on the second layer. Let us say all these SOMs are also of the size 2×2, then there is $4 \cdot 2 \cdot 2 = 16$ SOMs on the third layer, etcetera. This example is shown in Figure 2.10

Each SOM on a lower level is linked to one node on one level above, giving a more spread-out representation of the input patterns in the parent node.

The SOMs used in the HFM model are standard SOMs, therefore, their size is a predefined, fixed one; also, the number of hierarchy layers is predefined by the user, and the hierarchies are fully balanced (i.e., all possible paths down the hierarchies have the same length). The SOM on the first level should be of small size to get a gross, high level organisation of the input patterns.

2.5.2 The Training Algorithm

The algorithm uses the standard SOM learning algorithm, applying it to all its SOMs on all the hierarchical levels. The maps are trained sequentially. In other words, SOMs on lower hierarchy levels will be trained only when the process is finished on the previous layer.

The first-level SOM is initialised, and the standard self-organising process will develop an ordered mapping of the input space. As learning rate and neighbourhood function, time-decreasing functions are chosen (as in the SOM algorithm).

After the self-organising process is finished, for each node on the highest level, a new SOM will be created in the second hierarchical level. For each SOM, there is exactly one parent node that will "pass on" the input patterns mapped onto it to this SOM. The training process continues with organising all SOMs on the second level. When this is done, the process continues on the next level, until the predefined depth of hierarchical layers is reached.

The training process therefore consists of the following steps:

1. Initialising one single SOM on the highest hierarchy level.

2. Training the SOM according to the standard SOM algorithm.

3. If the current hierarchical level is smaller than the maximum, predefined hierarchy level: For all the nodes $i, 1 \leq i \leq m \times n$ of the current SOM, where m and n are the width and height of the network grid: create a new SOM on the next hierarchical level.

   Otherwise, the training algorithm is finished.

4. For all the SOMs on the next hierarchical level: initialise the SOM and train it, starting from step 2

One additional concept is introduced in [Mii90]: When passing on the input patterns from a node to its corresponding lower-level SOM, the input patterns will be modified in its dimension: the higher-level map acts as a filter, choosing the relevant features for each sub-map. This is done as follows: the features are ordered with respect to their variance over the input patterns "won" by this node, and only a fixed fraction r_i of the features will be chosen for the next level's SOM. Instead of a fixed fraction, also a variance threshold could be used.

This approach is justified by the fact that input patterns matched to the same node will have the values of a number of features in common (otherwise, they would not form a cluster). Therefore, the input patterns will not be distinguishable by these features, and they are, therefore, not needed anymore for the organisation process. This mechanism, however, might be only of use when there are clearly separated clusters in the input patterns (as is the case in the example demonstrated in [Mii90]). When using this mechanism on arbitrary data collections, only the reduction of vector components based on a variance threshold seems to be justified; otherwise components that have the lowest variance (in relation to the other components), but are not in any way common for all the input patterns, might be dropped; then, the map may lose some essential information for the organisation process.

2.5.3 Discussion

The proposed model of the Hierarchical Feature Map tries to overcome the shortcomings of the SOM when it comes to visualising the hierarchical taxonomy of the input space. When using well-chosen parameters for the grid sizes and the number of hierarchical levels, the HFM is able to represent the hierarchies in the input data correctly.

This, however, can rarely be achieved without any a-priori knowledge of the data, and is a non-trivial problem. This is one of the major shortcomings of the Hierarchical Feature Map ([DMR00a], [Mer97], [Mer98b]), and is also mentioned by the author himself, as he states that finding the best parameters might need some experiments, while also suggesting the development of automated mechanisms for adjusting the sizes of the map as well as the depth of the hierarchy according to the input data. One drawback is, furthermore, that the hierarchies will on all possible paths be developed to the maximum depth; for real-world data, this will, however, not be a realistic representation: some subsets of the data will most likely have more hierarchy levels, and others may have none at all. The author himself states that the HFM is best suited for strongly hierarchical data; for data where only some subsets have strong hierarchies, the architecture might not fit, though. All these shortcomings are addressed by the model proposed in Section 2.6, the Growing Hierarchical Self-Organizing Map.

For cluster detection, the Hierarchical Feature Map has been shown to be well suited (for example in [Mer98b]); with the small size (compared to a standard SOM) of the network's maps, on higher levels, nodes themselves will represent clusters in the input data collection. However, [Mer97] mentions a slight deficiency when it comes to detection of inter-cluster similarities: because of separating clusters in different branches of the hierarchy, information about similarities of input patterns might be invisible when they are mapped on different branches.

Implementation issues

The HFM builds basically on the standard SOM algorithm and architecture. However, concerning the computational speed, the HFM has significant advantages compared to the SOM: First, this is due to omitting vector components in lower-layer SOMs - a decreased vector dimension will speed up operations like best-matching node search and model vector adaptation. Secondly, the reduced size of the network grids, as well as having only a reduced set of input patterns for lower-layer nodes, will also contribute significantly to a speed-up in the organising process. A third argument for the speed-up is the following: the maps on lower layers present sub-categories of the input space; therefore, cluster interferences, which will result in a longer training time, will not occur, and the map is free from maintaining the overall structure, as this is done rather by the architecture of the Hierarchical Feature Map; this will save much computational effort. For a detailed description, refer to [Mer97].

In [Mii90], an example for the speed comparison is given by a HFM performing the same self-organisation task approximately 190 times faster than a standard SOM. This might be explained by using a data set the HFM was actually developed for, having almost perfect hierarchical structure. In [Mer97], another experimental speed-test has been done, with the HFM having a computational time six-times faster than a standard SOM for the completely same input (note that this was achieved although the HFM maps in total had a higher number of nodes than the SOM), a degree of speeding up which sounds more reasonable for real-world data than the one mentioned in [Mii90].

2.6 Growing Hierarchical Self-Organizing Map

The Growing Hierarchical Self-Organizing Map (GHSOM), as described e.g. in [RMD02], is another hierarchical neural network model; it addresses the major limitations of the Hierarchical Feature Map (as described in Section 2.5), as it combines ideas from the HFM with the Growing Grid (see Section 2.2): it uses dynamically growing grids instead of standard Self-Organizing Maps. Moreover, the hierarchical depth is chosen during run-time depending on the input data, further decreasing the need for an a-priori knowledge about the structure of the input patterns. The hierarchy is not necessarily balanced, which seems to be a more accurate representation of real-world data than in the HFM.

2.6.1 The Architecture

The GHSOM consists of several layers, where each layer consists of a number of independent self-organising networks. The GHSOM uses an incrementally growing version of the standard SOM, comparable to the Growing Grid, i.e. the grids will expand during the training process. A "virtual" *layer 0*, consisting of only one node, representing all the input data, is the starting

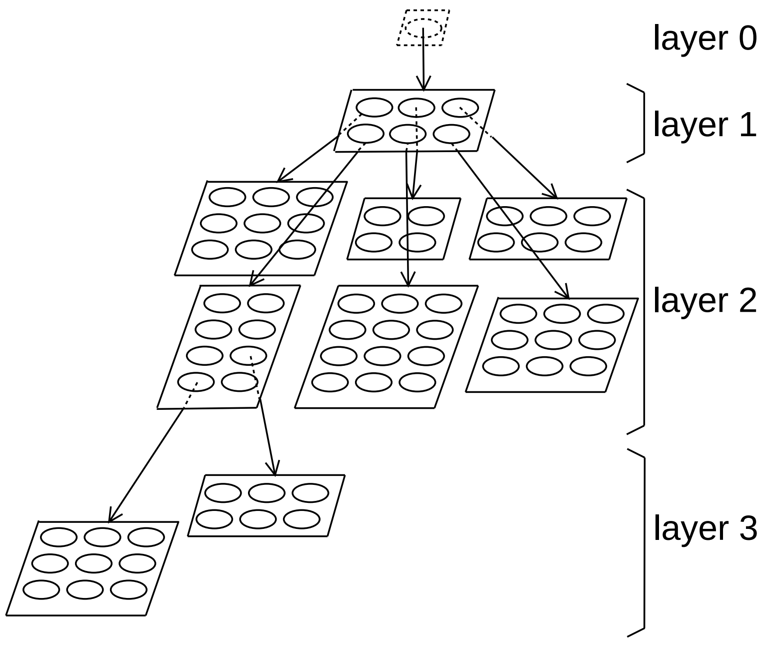


Figure 2.11: **Architecture of a Growing Hierarchical Self-Organizing Map**

point. This node has one map as *child* on the first layer. For each of the nodes in this first layer map, another map might be added to the second layer, to represent the mapped input patterns at a higher granularity level. This principle is repeated with the third layer as well as any further layers of the GHSOM. A sample architecture of a three-layered GHSOM is depicted in Figure 2.11. Note that the maps on the same level may develop different sizes, and that the hierarchy is not (necessarily) balanced.

Nodes store a value called *quantisation error*, a similar value is also stored for each map.

2.6.2 The Training Algorithm

Initialisation

The node in the layer 0 is initialised by its model vector m_0 taking the average value of all the input patterns, and its quantisation error qe_0 is computed according to

$$qe_0 = \sum_{x_i \in I} \|m_0 - x_i\|, \tag{2.22}$$

where C is the set of input patterns.

The SOM in layer 1 is proposed to be initialised with a small grid size, of e.g. 2×2. The nodes in the map are initialised with random model vectors.

Best matching node and model vector adaptation

Then, the map is trained according to the standard Growing Grid algorithm, i.e. input vector presentation and model vector adaptation. Contrary to the Growing Grid algorithm, however, a time-decreasing learning rate and neighbourhood range are used, with [DMR00a] mainly pointing at the fixed neighbourhood range in the Growing Grid to be problematic with increasing grid-sizes. This organising process is repeated for a fixed number of λ iterations.

Grid expansion

After the iterations, the grid is going to be expanded around the node representing its input space most inadequately. Contrary to the Growing Grid, this node is, comparable to the Incremental Grid Growing in Section 2.3.2, determined by the quantisation error qe_i,

$$qe_i = \sum_{x_j \in C_i} \|m_i - x_j\|, \tag{2.23}$$

where C_i is the set of input patterns mapped on node i, rather than on a winner count as in the Growing Grid.

Inserting rows or columns and initialisation of the new nodes follows the same scheme as in the Growing Grid (see Section 2.2.2).

In contrast to the Growing Grid, a map continues to grow until its *mean quantisation error MQE*, i.e. the mean of all the qe's of the map, is reduced to a certain fraction τ_1 of the quantisation error q_i of the node i in the upper layer of the hierarchy. The MQE of a map m is computed according to

$$MQE_m = \frac{1}{|U|} \cdot \sum_{i \in U} qe_i, \tag{2.24}$$

with U denoting the subset of nodes in the map where data has been mapped, and the stopping criterion can be formulated as

$$MQE_m < \tau_1 \cdot qe_{parent}, \tag{2.25}$$

where the subscript $_{parent}$ denotes the parent unit.

Hierarchical growth

As said before, the GHSOM is initialised with only one hierarchical layer, and the depth of the hierarchy (i.e. the number of layers) will be determined during run-time; also, not all nodes will grow into the full depth. A node is expanded hierarchically (i.e. for this node a new map in the next layer will be added and organised), when there is dissimilar input data mapped on this node, resulting in a high quantisation error qe_i. Another parameter, τ_2, is introduced as a threshold deciding about hierarchical growth in a node: a node is expanded hierarchically into another layer when its quantisation error is above the threshold. This threshold basically indicates the desired granularity level of data representation as a fraction of the initial quantisation error at layer 0.

If $q_i > \tau_2 \cdot qe_0$ holds true, then a new map will be added on the hierarchical layer below the current node's layer, and the input data mapped on this node is self-organised in this new map (which itself might grow new nodes or expand hierarchically as described above).

An important point with this training process is that it will not necessarily lead to a balanced hierarchy, a desired feature, as different depths in the hierarchy will better reflect non-uniformity in the data, an expected property of real-world data collections.

2.6.3 Applications of the GHSOM

The GHSOM has been tested and proved to be a useful tool in document organisation and creation of hierarchies on various real-world data sets: the rather small data sets of the *CIA World Factbook* [DMR00b], with 245 input patterns, and a collection of 420 of the *TIME* Magazine articles, as well as a rather large data set from an Austrian newspaper *Der Standard* ([RMD02], including 11,627 documents, each represented by a vector with a dimension of 3,799 components.

2.6.4 Discussion

The GHSOM addresses the main shortcomings of the Hierarchical Feature Map (as presented in Section 2.5.3): the fixed size of the individual maps and of the hierarchy depth, which need to be predefined and therefore require an a-priori knowledge of the input data. The GHSOM successfully deals with these problems by using a Growing Grid-like neural network model instead of the standard SOM, eliminating the need to predefine the grid sizes. Also, the depth of the hierarchies is determined during runtime; finally, the hierarchies are not balanced, which will in most cases, as real-world data tends to be non-uniformly distributed, represent the input space more accurately.

Still, similar criticism as for the Growing Grid can be brought forward: with inserting complete rows or columns respectively, the map might be spread out also in areas where it would not be necessary, thereby possibly creating nodes which do not represent any data of the input space. Similarly to a balanced hierarchy, which in most cases will not most accurately reflect a data collection, a rectangular structure has to face the same critique. This is proposed to be solved in the model presented in Chapter 3.

Additionally, the more flexible structure comes with the cost of specifying many more parameters than in the original SOM. These parameters are not depending on the input data sets (but give an abstraction of it), but need some experience to be set. Though, once this experience is gained, arbitrary data sets can be easily mapped.

Implementation issues

For the implementation, similar remarks as for the Hierarchical Feature Map (see 2.5.3) should be valid: especially for large data sets, the sizes of the individual network grids become small compared to the standard SOM, therefore the computational effort and the memory requirements can be significantly reduced.

2.7 Other Models

In this section, we describe models that are not based on a two-dimensional, rectangular output space, and therefore have limits in visualisation, but on the other hand, they suggest some interesting techniques and concepts worth investigating. We will start by describing the Growing Cell Structures in Section 2.7.1, followed by a presentation of the Neural Gas model in Section 2.7.2

2.7.1 Growing Cell Structures

The *Growing Cell Structures* (GCS), as introduced in [Fri92], also performs mappings from a high-dimensional data to a lower dimensional output space. In this model, though, the output space is not limited to two dimension, but can lie in the range of one to three dimensions. The output space is in general less regular, and in the two-dimensional representation, the grid is not rectangular anymore. This makes visualisation more complicated. Therefore, we are not going to consider this model as an alternative for our purposes.

However, the Growing Cell Structures introduces some concept that might be interesting for our purpose. The structure allows for an approach to adding new nodes, where the new cells are inserted between a node with the highest relative *resource value* (i.e. a winner counter), and this cell's most dissimilar neighbour (comparable to the Growing Grid in Section 2.2), without having to add any other new cells (in the Growing Grid, we had to insert whole rows or columns respectively). This will allow a very accurate representation of the input space. This strategy does not seem to be easy to incorporate in rectangular architectures though.

Still, the Growing Cell Structures holds another interesting concept, which was proposed to be investigated for example also in the Growing Self-Organizing Map: After a certain number of iterations, cells might be deleted from the grid when they represent areas in the input space with very low probability density. This concept might be worthy of inclusion in SOM-based models with flexible network architectures, like the Incremental Grid Growing or the Adaptive Hierarchical Incremental Grid Growing.

Of the Growing Cell Structures, also hierarchical extensions exist, for example the *TreeGCS* in [HA01]. In this paper, also issues of the *stability* of the algorithm are mentioned: the algorithm is found to be instable: it generates different cell structures when trained with the same set of data, but in a different order.

2.7.2 Neural Gas

The *Neural Gas* model, first presented in [MS91], is a neural network model that maps from a high-dimensional input space to a number of nodes in the output space A, with their model vectors $m_i \in \Re^n$, i.e. an n-dimensional output space. Though this property of being not necessarily two-dimensional makes this model not appealing for our purposes, it comprises some interesting features. The most interesting might be that it provides cluster visualisation by establishing connections between nodes. After each input vector presentation, a connection is established between the best and second-best matching nodes. Also, connections might be deleted; this is achieved by a concept called *aging*. All connections, except the one between the best and second-best matching node, will increase their age, and if this value becomes higher than a predefined maximum age, they will be deleted. By this connectivity approach, connections are not limited to neighbouring nodes in the output space, but could virtually be established between any nodes in the network (though, however, towards the end of the training process, only connections in some neighbourhood range will remain). Adaptation towards the input vectors will be performed only on the best matching node and its directly connected neighbours.

Fritzke [Fri95b] proposes an incrementally growing model called the *Growing Neural Gas*, based on the Neural Gas, which was based on a fixed, and predefined, network size. The growing scheme is similar to other models previously described in this chapter, it uses concepts also utilised in the Growing Hierarchical Self-Organizing Map and the Growing Cell Structures, with the nodes accumulating a (quantisation) error value as the distance between mapped vectors and the model vector. After a certain number of λ iterations, a new node is inserted between the node with the highest accumulated error and its neighbour with the highest error value, and the connection between these nodes is replaced by two corresponding connections to the new node. The growth process stops when a certain criterion, e.g. the network size, or some other performance measure, is fulfilled. In the Growing Neural Gas, a fixed neighbourhood and learning rate are proposed.

Chapter 3

Adaptive Hierarchical Incremental Grid Growing

The *Adaptive Hierarchical Incremental Grid Growing* (AHIGG), as proposed in [He01] and [MHDR03], was designed to combine the various features of the models presented in the previous chapter, and to overcome the limitations for each of them. Simplified, the AHIGG combines the features of the Growing Hierarchical Self-Organizing Map with the Incremental Grid Growing as the model for the individual maps (instead of the Growing Grid as it is the case in the Growing Hierarchical Self-Organizing Map). Some more additions and improvements have been applied to the algorithm of the Incremental Grid Growing, e.g. to reduce the learning time by an improved initialisation rule, and a mechanism to determine a stagnation in the training process. In the rest of this chapter, the architecture and the algorithm of the AHIGG will be presented, before the model will be tested on real-world data in the next chapter.

3.1 The Architecture

The AHIGG is basically composed of a number of independent *Incremental Grid Growing* networks, arranged in hierarchical layers. Each of these layers represents the input data at a specific level of granularity. On the first layer, a rough idea of the structure of the input data is given; each node of this layer may then be expanded to another IGG map in the next layer, thus giving a more detailed picture of this node's subset of the input data. The architecture of the IGG-maps, i.e. the size and layout, as well as the depth and balance of the hierarchy is determined automatically corresponding to the input space. The beginning of the AHIGG is a single node in a "virtual"

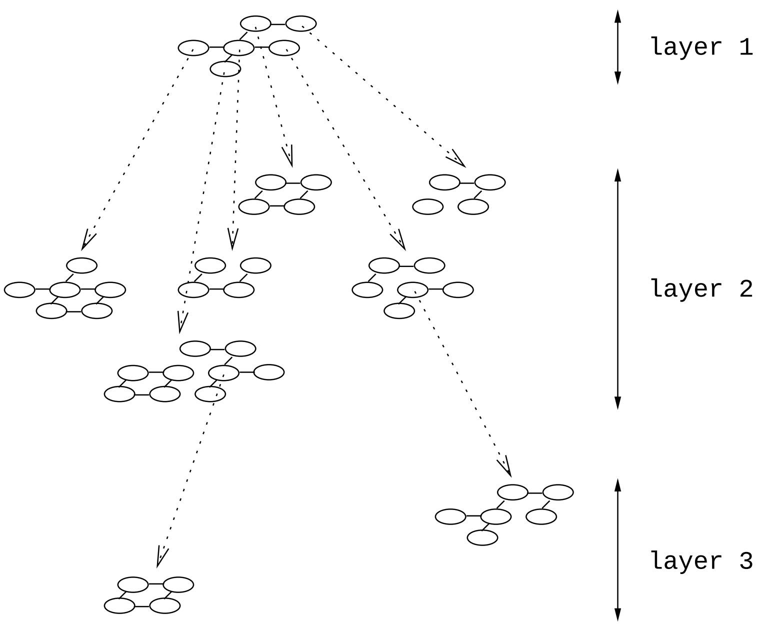


Figure 3.1: **Architecture of a three-layered Adaptive Hierarchical Incremental Grid Growing**, with differently sized and shaped IGG maps, and an unbalanced hierarchy

layer 0, representing a statistical mean of all the input data. All nodes store values for the mean quantisation error *mqe*; each map has a *vector mqe*, i.e. the average quantisation error per vector in the map, denoted as $vMqe$.

In Figure 3.1, one possible architecture of a trained AHIGG is depicted. Note that the hierarchy is not necessarily balanced.

3.2 The Training Algorithm

Three different phases can be identified in the training process in the *initialisation*, the *IGG-based training*, and the *hierarchical expansion* phase. They will be presented in detail now.

3.2.1 Initialisation

A single IGG map is initialised in the layer 1, usually having a 2×2 grid size, with all nodes connected in the grid. As a *parent node* to this map in the layer 1, the "virtual" layer 0 is initialised with a model vector m_0, being the average of all the input vectors x. This node's *mqe* is calculated according to:

$$mqe_0 = \frac{1}{|C|} \sum_{x \in C} \|m_0 - x| \tag{3.1}$$

where C is the set of input vectors. The mqe_0 will play a crucial role during the training process (mqe_0 is a measure for the diversity in the input set).

All nodes in layer 1 will be initialised with a random model vector; contrary to the IGG, however, this initialisation is not completely random, but takes available qualitative information in the form of the model vector of the parent node into account:

$$m_i = m_{parent} + mqe_{parent} \cdot \mathrm{v}_{rand} \tag{3.2}$$

where the subscript $_{parent}$ denotes this map's parent node, and v_{rand} is a random vector of length 1., i.e. we limit the range to the n-dimensional subspace with m_{parent} as centre, and mqe_{parent} as radius.

3.2.2 Training

This phase consists of three different parts: first, a slightly modified version of the IGG training as presented in [BM93] is carried out; after the grid has taken its final shape, a *fine tuning* and a *post-processing* phase will follow.

IGG based learning

The map is trained in training cycles according to the standard SOM algorithm: A randomly chosen input vector is presented to the map, and the winning node is chosen according to Equation 2.1. Then, the model vectors of the winning node and nodes within the neighbourhood-range are adapted, according to Equation 2.2.

For the learning rate α, a time decreasing function is taken, allowing a global organisation in the beginning of the training process, and a more local ordering towards its end. After each training cycle, the function is reset to its initial value.

Special attention has to be paid to the neighbourhood function: As in the IGG training, there is a different concept of the *distance* between two nodes: because of the non-existence of some connections, we have to define the distance between two nodes not via their position on the grid, but rather by the length of the *shortest path between two nodes.* This concept was illustrated in Figure 2.7.

One training cycle consists of $k \times \lambda$ adaptation steps, with k being the number of input patterns for this map, and λ determining how many adaptation steps per input pattern are performed on average. After one cycle is

qe_i	quantisation error	The quantisation error of a node i, i.e. the sum of the (Euclidean) distances between the input vectors mapped on a node i and the node's model vector; computed according to Equation 2.23.
mqe_i	mean quantisation error	A node's quantisation error, normalised by the number of mapped input vectors; computed according to 3.10.
QE	Quantisation Error	A map's quantisation error as the sum of the quantisation errors qe_i, or the mean quantisation errors mqe_i, of all the nodes in the map.
MQE	Mean Quantisation Error	A map's quantisation error as the sum of the quantisation errors qe_i, or the mean quantisation errors mqe_i, normalised by the number of nodes in the map; computed according to Equation 2.24, when using the quantisation errors qe_i.
$vMQE$	Vector-based Mean Quantisation Error	A map's as the sum of the nodes' qe_i, normalised by the number of input vectors on this map; computed according to Equation 3.3.

Table 3.1: Different variants to compute quantisation errors.

finished, the grid might grow new nodes; this is determined by examining the representation quality of the map. In principal, several different variants for computing *quantisation errors* can be distinguished; Table 3.1 gives an overview of commonly used measures.

In the AHIGG, the vector-based $vMQE$ is utilised; with the map's nodes qe_i as defined in equation 2.23, this measure is calculated according to:

$$vMQE = \frac{1}{|C|} \cdot \sum_{i=1}^{|C|} qe_i, \tag{3.3}$$

where C denotes the subset of vectors mapped on this map. In other words, the $vMQE$ is the quantisation error per vector.

The map will be expanded as long as

$$vMQE \geq \tau_1 \cdot mqe_{parent} \tag{3.4}$$

holds true; with $0 < \tau_1 < 1$, the desired fraction of the parent node's mean quantisation error (i.e. the desired improvement of the representation quality) is used as the stopping criterion. The parameter τ_1 is responsible for guiding the growth process on the IGG maps: the lower the value for τ_1, the bigger the maps will grow.

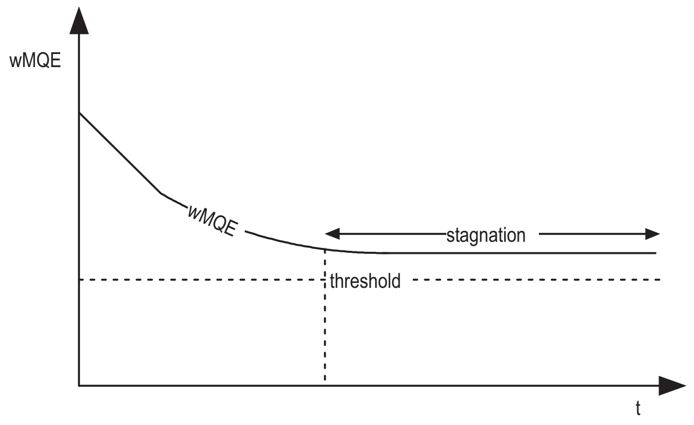


Figure 3.2: **Stagnation in the AHIGG training process**: Minimising the mean quantisation error goes into a stagnation.

However, there might be cases when the growth process is going into a stagnation phase; in other words, the quality of the representation (in terms of reducing the $vMQE$) would be hardly improved in the subsequent training cycles, and the training process might spend a long time, or even worse, the desired fraction τ_1 of the mqe_0 according to Equation 3.4 might not be achieved at all. A case where this might happen is described in Section 4.5.1

Therefore, a second criterion for stopping the growth process is proposed: with a user-defined improvement degree ς, the improvement of the $vMQE$ is checked according to

$$1 - \varsigma > \frac{vMQE(k+1)}{vMQE(k)}, \tag{3.5}$$

where $vMQE(k)$ is the map's $vMQE$ after the k-th training cycle. As soon as Equation 3.5 is not fulfilled anymore, a stagnation occurs, and the growth process is terminated. In Figure 3.2, a possible scenario is depicted, with the training process actually never achieving the desired representation quality. Therefore, the process should be stopped when detecting the stagnation.

New nodes are grown around the *boundary node*, i.e. any node that has neighbouring positions that are not yet taken by other nodes) which holds the highest cumulative error, calculated according to Equation 2.23. Note that in contrast to the IGG, the error value is calculated **after** a training cycle, instead of accumulating the error during the iterations: the settled model vectors are used to gain a more realistic description of the error.

In [BM93], a strategy preserving the local topology by taking into account statistical means is proposed to initialise new nodes (see Equations 2.15 and 2.16); [He01], however, suggests to initialise the new model vectors randomly in an ϵ-environment of the error node according to

$$m_{new} = m_{error} + \epsilon \cdot v_{rand}, \tag{3.6}$$

where m_{new} and m_{error} denote the model vectors of the new node and the error node, respectively; v_{rand} is a random vector with length of 1, and ϵ a small constant with $0 < \epsilon \ll 1$.

After new nodes have been added, the training cycle is completed by examining wether connections should be added or removed. This is done by examining the pairwise distances between neighbouring nodes; similar to the IGG, two thresholds, the *connect threshold* and the *disconnect threshold*, are used; however, as suitable absolute values are difficult to define, these thresholds are rather computed depending on the data. For this purpose, the overall average distance p between the number of m pairwise distances between neighbouring nodes' model vectors is calculated according to

$$\rho = \frac{1}{m} \cdot \sum_{i \neq j} \|m_i - m_j\|. \tag{3.7}$$

By this, the connection threshold is defined as the fraction φ_c of ρ: connections are added when the distance between two nodes i and j drops below the threshold according to:

$$\|m_i - m_j\| < \varphi_c \cdot \rho. \tag{3.8}$$

Analogous, connections are deleted when the distance becomes bigger than the disconnect threshold, calculated as the fraction φ_d of ρ according to:

$$\|m_i - m_j\| > \varphi_d \cdot \rho. \tag{3.9}$$

After examining the connections, the training process is continued by carrying out another training cycle, until Equations 3.4 and 3.5 are not fulfilled anymore.

Fine-tuning

Analogous to the concept introduced in the Growing Grid (Section 2.2.2), the AHIGG training process includes a fine-tuning phase, which is carried out after the map has reached its final size. No new nodes are added, but the aim

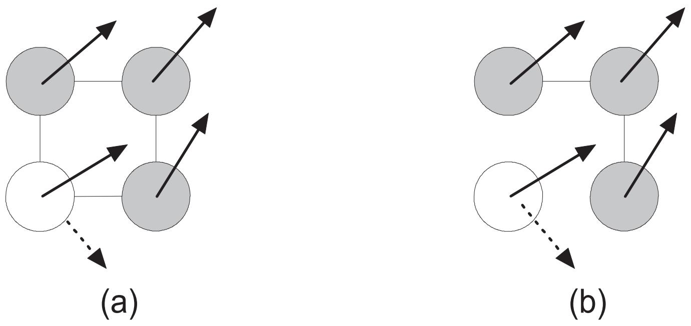


Figure 3.3: Cluster detection in the AHIGG post-processing phase

is to let the map converge to a stable organisation state. As the vectors should not fluctuate too much anymore, the learning rate is rather low compared to the usual self-organising process; furthermore, the neighbourhood adaptation is limited to the winning node only.

Post-processing

When the network has reached its final structure and the model vectors are settled, the representation quality of the clusters is to be improved: this is done by examining the connections between the nodes. A difference to the corresponding phase in the IGG is the following: [He01] instead suggests to use the statistical mean of the mapped vectors for comparing two nodes about connectivity. This is justified by the fact that the model vectors may "cover" cases where there are outliers mapped on a node. A possible scenario is depicted in Figure 3.3: on the white node, input patterns which are quite different to the node's model vector, are mapped. In (a), using the model vector as criterion, this node would still belong to the cluster which is formed by the three other nodes. However, as this node contains highly dissimilar data to the one otherwise represented in the cluster, it should form its own cluster. In Figure 3.3 (b), this is achieved by using the statistical mean as a criterion for deleting (or adding) connections. The scenario described here might occur when the *outlier(s)* are mapped late in the training process, and due to a very small learning rate do not adapt they model vector greatly.

Besides using a different measurement, the post-processing is identical to the process of examining connections during the IGG-based learning phase.

With the postprocessing, the training of an individual IGG map is completed.

3.2.3 Hierarchical Expansion

When a map in the AHIGG is completely trained, it is examined whether any of the map's nodes requires a higher resolution of its input patterns, and therefore should be expanded hierarchically, i.e. a new IGG map in the next layer will be added for this node. As a measure for nodes representing their input space inadequately, a node i's mean quantisation error, calculated according to

$$mqe_i = \frac{1}{|C_i|} \sum_{x \in C_i} \|m_i - x\|, \tag{3.10}$$

where C_i is the set of input patterns mapped on this node, and m_i its model vector, is utilised. The, a simple threshold logic is used: with a parameter τ_2, $0 < \tau_2 < 1$, all nodes for which

$$mqe_i > \tau_2 \times mqe_0 \tag{3.11}$$

holds true are expanded. Here, the parameter τ_2 is thus responsible for guiding the hierarchical growth process: the lower the value for τ_2, the deeper the hierarchy will develop.

The newly established map in the next layer is initialised in a similar way as the map in layer 1: the new nodes are initialised according to Equation 3.2. Then, the new map is trained as described in Section 3.2.2, and its nodes might in turn be expanded hierarchically.

3.3 Automatic Labelling

The AHIGG provides automatic cluster visualisation by the concept of connectivity; however, for a proper interpretation of the mapping, labelling of the clusters by their relevant features is desired. For that purpose, a technique similar to the *LabelSOM* technique, as presented in Section 2.1.5, is proposed.

However, contrary to the LabelSOM, the importance of a feature i in a data set of k items is proposed to be based on its statistical mean $\overline{x_i}$,

$$\overline{x_i} = \frac{1}{k} \sum_{j=0}^{k} x_{ji}, \tag{3.12}$$

as well as on the statistical standard deviation σ_i, according to

$$\sigma_i = \sqrt{\frac{1}{k-1} \sum_{j=1}^{k} (x_{ji} - \overline{x_i})^2}. \tag{3.13}$$

Then, a formula for the importance imp_i of the feature i might be calculated according to:

$$imp_i = \frac{\overline{x_i}}{\sqrt{1+\sigma_i}}. \tag{3.14}$$

Note that if there is a zero variance, only the average value determines the importance of a feature.

The main argument for this approach is that features that may have a high average value, but also a high variance, are nevertheless important for describing the input data. In the LabelSOM method, the importance is firstly based on the quantisation error; therefore, a high variance, leading to a high quantisation error, will rule out these features from being selected as labels.

3.4 Discussion

The *Adaptive Hierarchical Incremental Grid Growing* builds on utilising various features of the models presented throughout chapter 2, resulting in a very adaptive architecture that promises to be able to represent arbitrary input data sets. Also, it liberates the user from the need to have an a-priori knowledge of the input data to predefine grid sizes or numbers of hierarchical layers; rather, this is computed during run-time by the AHIGG, depending on the given input patterns. The usefulness and applicability of the model has been proven in [He01] on a demo data set as well as on a real-world text collection, namely a collection of the *Time* magazine articles. However, this was a rather small collection, and the usefulness in larger applications has yet to be shown.

The benefits of the AHIGG, though, do not come totally free: compared to the original SOM, the algorithm has lost in terms of stability and robustness. More training parameters are to be specified by the user; hence, the need to carry out some experiments to get a "feeling" for the parameters. However, [He01] claims that as soon as this feeling has developed, the AHIGG can be used for arbitrary data sets. This will be tested with another real-world data set in the next chapter.

Implementation issues

Similar arguments as for the GHSOM (Section 2.6.4) will also apply for the AHIGG: Due to the reduction in grid sizes by the hierarchical structuring, the training process should be carried out much faster compared to a standard SOM, especially for large data sets.

However, the implementation of the AHIGG becomes quite complex, as various different mechanisms are to be implemented.

Chapter 4

Application of the AHIGG

In this chapter, we will describe the application of the Adaptive Hierarchical Incremental Grid Growing in experiments. We will start by giving an overview of our implementation of the AHIGG in Section 4.1.

Then we will test our implementation of the Adaptive Hierarchical Incremental Grid Growing on three data sets: Firstly, for giving an intuitively understandable proof of the functioning of the AHIGG, we will apply well known *demo data sets*. We will start with the small *animals* data set, which allows an easy recognition of how the AHIGG works by its simple and distinct input patterns (Section 4.2). Then, the comparably bigger *zoo* demo data set will be used to give another example in Section 4.3; while the higher number of input patterns allows a more realistic verification, this higher number makes it more difficult to "manually" analyse the results.

In Section 4.4, a real-world data set will be examined. The data set comes from the domain of *tourism*, and contains free-text, unstructured descriptions of hotels in Austria. The data set is available in two versions, as the hotel descriptions are given in English and German. However, the data has yet to be pre-processed, to obtain a representation that is suitable for use in the AHIGG. This is illustrated in Section 4.4.1. In Sections 4.4.2 to 4.4.4, we will present the results obtained for our data set with various models, namely the Self-Organizing Map, respectively the Growing Grid, the Growing Hierarchical Self-Organizing Map, and the Adaptive Hierarchical Incremental Grid Growing itself.

The chapter is completed by a conclusion about the results of the application experiments in Section 4.5.

4.1 Implementation

The main objective of the implementation, an integral part of this thesis, was to provide a clean and user-friendly implementation of the AHIGG. For user-friendliness, some key features have been identified:

- The implementation should be easily *portable* and not limited to a specific operating system.
- The implementation should be well documented.
- The implementation should produce an easy to use, easy to share, and easy to port visualisation. No specific software should be necessary to view the output.
- The implementation should be convenient to use both for "experts" and "beginners".

To achieve a portable implementation, the *Java* programming language was chosen[1]; though this implies a loss of computational speed, as the Java programming language is rather slow in run-time, the portability seems to justify this decision.

To implement a software that is convenient to use both for beginners and experts is in general not an easy task. However, a good solution seems to be found by providing experts with a command-line based interface, thus allowing for unsupervised batch processing of many different training processes, while new users might find the included graphical interface to be an easy way to begin using the model.

4.1.1 Visualisation

For an easy solution to offer a representation of the output, where one can navigate through different layers, the concept of *hypertext* is very appealing. Furthermore, an output generated in the *hypertext markup language* (HTML) format is easily readable by any so-called *browser* software, and therefore not limited to a specific software. This is why this representation technique has been chosen. However, to automate and simplify viewing the output, a simple browser was implemented in the Java language as well.

In Figures 4.1 to 4.4, the visualisation of a possible outcome of an AHIGG training process is depicted. In the top of the output, some statistical data about the currently viewed map is shown - the layer it is in, the map ID,

[1]http://java.sun.com/

which is composed of the parent node's ID, and information about the map size, namely the number of nodes, the grid size of the map, and the number of mapped input patterns. Additionally, *hypertext links* to view the Cluster Connections visualisation of this map (see Section 4.1.2), as well as viewing this map in a previous training state (this is only applicable if the map was expanded horizontally at least once).

Then, the actual map follows. Nodes are symbolised with gray-coloured boxes, while grid positions which are not occupied by nodes are left white. The map's connectivity is indicated by the existence of green-coloured connections between the individual nodes. The node boxes contain some statistical information on their top (the number of vectors mapped, as well as the *mqe* of this node). Below, the list of *labels* describing the mapped data follows; if there are any input patterns mapped, a list containing some of them will appear below the feature labels. If a node is expanded to another layer, this is indicated by a hyper-text link labelled *lower hierarchy*. Following this link will display the map that represents this node's input in the next hierarchical layer. From a lower-hierarchical layer, a hyper-text link labelled *Upper hierarchy* will lead the user back to the previous layer.

Finally, in the bottom of the output, some information about the training process of this map can be found, like the number of iterations, and the *mqe* of the map and the parent node.

When we refer to a node, we use a x/y scheme, where x denotes the column, starting with 0, and y denotes the row, starting with 0. Therefore, the top-left node would be referred to as 0/0, the second node in the first row with 1/0, while the second node in the first column is indicated as 0/1.

4.1.2 Improvements of the Original Model

Compared to the original model of the Adaptive Hierarchical Incremental Grid Growing, we included a couple of improvements in our implementation; they will be described below.

Growth

As already proposed in [He01], we used a slightly different approach for the grid-growth, by growing more than one new node at the same time. The user can choose, parameter driven, how many new nodes should be grown during one training cycle. The algorithm then, at the end of each SOM training cycle, adds new nodes around the n nodes with the highest error values. This algorithm improvement, as described in [He01], should lead to a shorter training phase; in Section 4.4.4, we will give our conclusions about this assumption.

Cluster visualisation

In the Adaptive Hierarchical Incremental Grid Growing, clusters are easily recognisable because of the connectivity of the network structure: clusters are sets of interconnected nodes, while nodes belonging to different clusters do not have any connection-path between them.

However, this approach might be a bit too simplistic; it is still not possible to see if there would be inter-cluster similarities between nodes, and it is not possible to recognise the degree of similarity between nodes in the same cluster. Therefore, a solution might be to apply the concept of *Cluster Connections*, as described in Section 2.1.4, to the trained map. One possible way would be to introduce the following statuses of relationships between two nodes: *very similar* and *similar* for nodes that are in the same cluster, as well as *partially similar* and *dissimilar* for nodes that are not in the same cluster; this was adopted in our implementation.

The states *similar* and *dissimilar* are represented by the way it is currently done in the AHIGG, i.e. *connected* and *disconnected*. To avoid more user-defined thresholds, the thresholds for states *very similar* and *partially similar* are computed by the existing thresholds: φ_v, the threshold for very similar nodes, is calculated as

$$\varphi_v = \frac{\varphi_c}{2}, \tag{4.1}$$

and $\varphi_{p,i}$, the threshold for partially similar nodes in a map i, is computed according to

$$\varphi_{p,i} = \varphi_d + \frac{maxdistance_i - \varphi_d}{2}, \tag{4.2}$$

where *maxdistance* is the maximum distance between two nodes' model vectors in the map i.

The different connection states may be visualised in the output by different colours, and by a missing connection for dissimilar nodes. This different approach to visualising inter-cluster and intra-cluster relationships is integrated in our implementation such that both variants of the output are generated; the user can choose via, hypertext-links, to view the preferred output mode.

Name	small	medium	big	2 legs	4 legs	hair	hooves	mane	feathers	hunt	run	fly	swim	Species
dove	1	0	0	1	0	0	0	0	1	0	0	1	0	bird
hen	1	0	0	1	0	0	0	0	1	0	0	0	0	bird
duck	1	0	0	1	0	0	0	0	1	0	0	0	1	bird
goose	1	0	0	1	0	0	0	0	1	0	0	1	1	bird
owl	1	0	0	1	0	0	0	0	1	1	0	1	0	bird
hawk	1	0	0	1	0	0	0	0	1	1	0	1	0	bird
eagle	0	1	0	1	0	0	0	0	1	1	0	1	0	bird
fox	0	1	0	0	1	1	0	0	0	1	0	0	0	mammal
dog	0	1	0	0	1	1	0	0	0	0	1	0	0	mammal
wolf	0	1	0	0	1	1	0	0	0	1	1	0	0	mammal
cat	1	0	0	0	1	1	0	0	0	1	0	0	0	mammal
tiger	0	0	1	0	1	1	0	0	0	1	1	0	0	mammal
lion	0	0	1	0	1	1	0	1	0	1	1	0	0	mammal
horse	0	0	1	0	1	1	1	1	0	0	1	0	0	mammal
zebra	0	0	1	0	1	1	1	1	0	0	1	0	0	mammal
cow	0	0	1	0	1	1	1	0	0	0	0	0	0	mammal

Table 4.1: The *animals* demo data set

4.2 Animals Demo Data Set

As a first approach to show the usefulness of the Adaptive Hierarchical Incremental Grid Growing for clustering and hierarchical ordering, we use a data set well known in the SOM-community, the so-called *animals* demo data set. This data set consists of 16 input patterns with 13 distinctive features; the complete data set is shown in Table 4.1.

The animals demo data set is a rather small one, but the structure and hierarchies of this data space are well known, with the animals forming distinct "clusters", namely mammals and birds, as seen in Table 4.2. These clusters are further separated into subgroups: birds could be separated into sub-groups of airborne or hunting birds, while the mammals can be classified as hoofed, hunting, or mammals with a mane. Additionally, both clusters can be distinguished according to size.

4.2.1 Results

In Figures 4.1 to 4.4, a graphical representation of a AHIGG trained with the animals demo data set is depicted. In Figure 4.1, the first layer of the trained mapped is shown. The map shows three distinctive clusters. On the left side, altogether four nodes form the cluster of birds. While the upper three nodes

Species	Count
Birds	7
Mammals	9
Total	16

Table 4.2: Categories in the *animals* demo data set

(1/0, 0/1, 1/1) represent airborne birds, and are grouped according to their size respectively the *hunting* ability, the lower node 1/2 represents the non-airborne birds. They are represented on a higher granularity level in the next layer, where the swimming birds *goose* and *duck* are separated from the non-swimming *hen*, as seen in Figure 4.2.

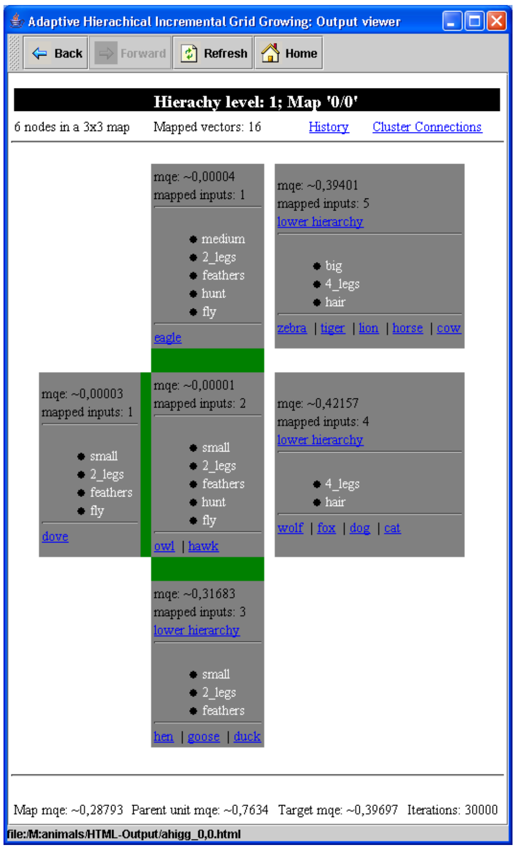
In Figure 4.3, the second layer map of *big* mammals is shown; other groupings, with the hoofed mammals forming a cluster, might be possible as well, however, the one learned by the AHIGG is justified by clusters of *running* and non-running mammals emerging. In Figure 4.4, the second layer map for *small and medium* sized mammals is depicted. Here, *dog* and *wolf* form a cluster of *running* mammals, while *cat* and *fox* form their own cluster.

Discussion

Based on the results from organising the animal demo data set, it can be stated that the mapping generated by the AHIGG training process is easy to verify and logical; therefore, at least for small data sets, the AHIGG can be seen as a functional tool for structuring data, and to create a hierarchical view. The data set itself lacks in dimensionality, therefore ambiguous results, which are however all correct and explainable, can emerge. Further, the small size of the data set doesn't allow demonstration of the hierarchical functionality. Therefore, we will test the AHIGG on another demo data set in the next Section 4.3.

4.3 Zoo Demo Data Set

As another demo data set, we use the so-called *zoo* demo data set. This data set is larger than the animal data set, as it consists of 100 input patterns, describing animals with 20 distinct features. Still, this is a rather small data set, but once again, the structure and hierarchies of this data space are well known; here, we can identify more different clusters, such as mammals, birds, fish, and so on; these clusters then again have more sub-categories. A part of this data set is shown in table 4.3. Note that, for the sake of space, not


Adaptive Hierachical Incremental Grid Growing: Output viewer

Hierachy level: 1; Map '0/0'

6 nodes in a 3x3 map Mapped vectors: 16 History Cluster Connections

mqe: ~0,00004
mapped inputs: 1
- medium
- 2_legs
- feathers
- hunt
- fly

eagle

mqe: ~0,39401
mapped inputs: 5
lower hierarchy
- big
- 4_legs
- hair

zebra | tiger | lion | horse | cow

mqe: ~0,00003
mapped inputs: 1
- small
- 2_legs
- feathers
- fly

dove

mqe: ~0,00001
mapped inputs: 2
- small
- 2_legs
- feathers
- hunt
- fly

owl | hawk

mqe: ~0,42157
mapped inputs: 4
lower hierarchy
- 4_legs
- hair

wolf | fox | dog | cat

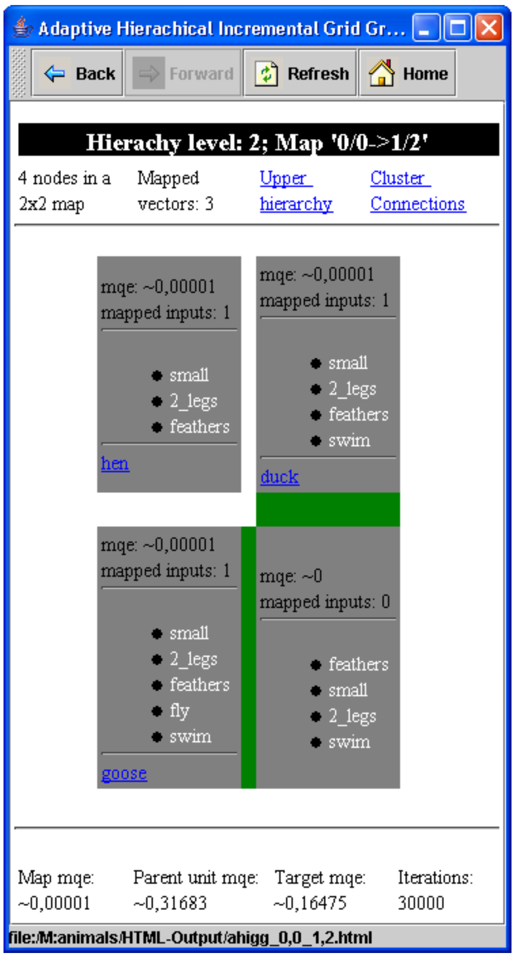
mqe: ~0,31683
mapped inputs: 3
lower hierarchy
- small
- 2_legs
- feathers

hen | goose | duck

Map mqe: ~0,28793 Parent unit mqe: ~0,7634 Target mqe: ~0,39697 Iterations: 30000

file:/M:animals/HTML-Output/ahigg_0,0.html

Figure 4.1: **AHIGG of the animals data set**: first layer map.


Adaptive Hierachical Incremental Grid Gr...

Hierachy level: 2; Map '0/0->1/2'

4 nodes in a 2x2 map Mapped vectors: 3 Upper hierarchy Cluster Connections

mqe: ~0,00001
mapped inputs: 1
- small
- 2_legs
- feathers

hen

mqe: ~0,00001
mapped inputs: 1
- small
- 2_legs
- feathers
- swim

duck

mqe: ~0,00001
mapped inputs: 1
- small
- 2_legs
- feathers
- fly
- swim

goose

mqe: ~0
mapped inputs: 0
- feathers
- small
- 2_legs
- swim

Map mqe: ~0,00001 Parent unit mqe: ~0,31683 Target mqe: ~0,16475 Iterations: 30000

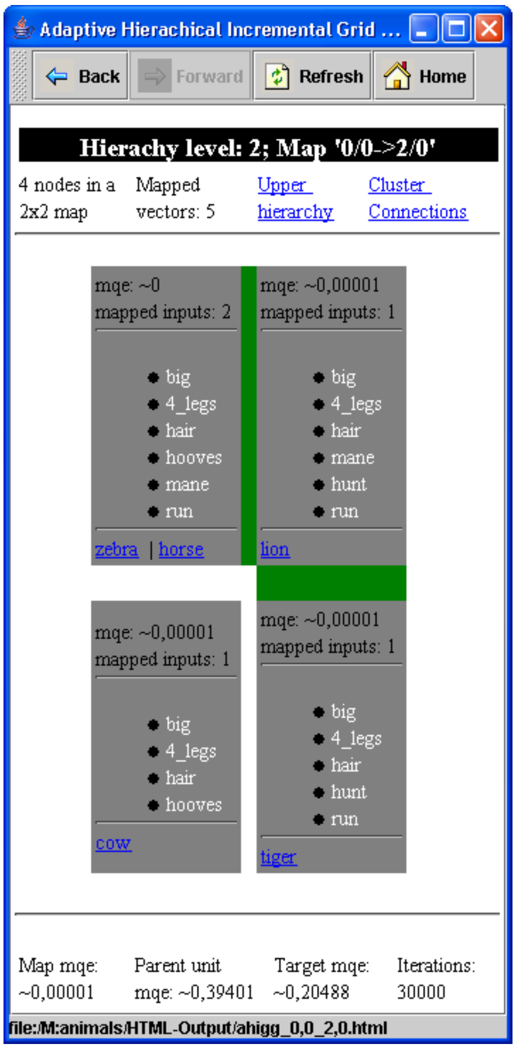
file:/M:animals/HTML-Output/ahigg_0,0_1,2.html

Figure 4.2: **AHIGG of the animals data set**: second layer map of some birds.

all input vectors and features are shown in this table; a full table of the data set can be found in Appendix B, Table B.1.

Compared to the commonly used zoo data set, some changes were made:

- The *type* attribute, commonly encoded as a numeric value ranging from 1..7, is not used as a component of the feature vectors. This is because this encoding, on the one hand, does not fit the other features, which are binary values, and this feature describes something (namely, different clusters of animals) we actually want to *detect* using our neural network model. Therefore, this feature is used solely as a good indicator to verify the results.



Hierachy level: 2; Map '0/0->2/0'

4 nodes in a 2x2 map | Mapped vectors: 5 | Upper hierarchy | Cluster Connections

mqe: ~0
mapped inputs: 2

- big
- 4_legs
- hair
- hooves
- mane
- run

zebra | horse

mqe: ~0,00001
mapped inputs: 1

- big
- 4_legs
- hair
- mane
- hunt
- run

lion

mqe: ~0,00001
mapped inputs: 1

- big
- 4_legs
- hair
- hooves

cow

mqe: ~0,00001
mapped inputs: 1

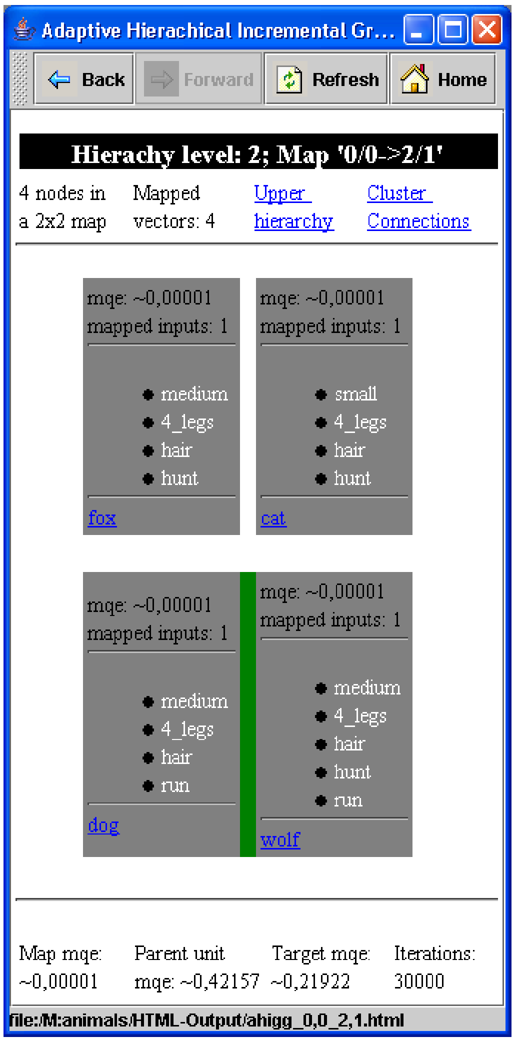
- big
- 4_legs
- hair
- hunt
- run

tiger

Map mqe: ~0,00001 | Parent unit mqe: ~0,39401 | Target mqe: ~0,20488 | Iterations: 30000

file:/M:animals/HTML-Output/ahigg_0,0_2,0.html

Figure 4.3: **AHIGG of the animals data set**: second layer maps of big mammals.



Hierachy level: 2; Map '0/0->2/1'

4 nodes in a 2x2 map | Mapped vectors: 4 | Upper hierarchy | Cluster Connections

mqe: ~0,00001
mapped inputs: 1

- medium
- 4_legs
- hair
- hunt

fox

mqe: ~0,00001
mapped inputs: 1

- small
- 4_legs
- hair
- hunt

cat

mqe: ~0,00001
mapped inputs: 1

- medium
- 4_legs
- hair
- run

dog

mqe: ~0,00001
mapped inputs: 1

- medium
- 4_legs
- hair
- hunt
- run

wolf

Map mqe: ~0,00001 | Parent unit mqe: ~0,42157 | Target mqe: ~0,21922 | Iterations: 30000

file:/M:animals/HTML-Output/ahigg_0,0_2,1.html

Figure 4.4: **AHIGG of the animals data set**: second layer map of small and medium sized mammals.

Name	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	...	type
antelope	1	0	0	1	0	0	0	1	...	mammal
buffalo	1	0	0	1	0	0	0	1	...	mammal
carp	0	0	1	0	0	1	0	1	...	fish
chicken	0	1	1	0	1	0	0	0	...	bird
deer	1	0	0	1	0	0	0	1	...	mammal
dove	0	1	1	0	1	0	0	0	...	bird
duck	0	1	1	0	1	1	0	0	...	bird
elephant	1	0	0	1	0	0	0	1	...	mammal
flea	0	0	1	0	0	0	0	0	...	insect
giraffe	1	0	0	1	0	0	0	1	...	mammal
...	...	...	...	...	...	...	...	...	...	...

Table 4.3: A part of the *zoo* demo data set

Species	Count
Amphibians	4
Birds	20
Fish	13
Insects	8
Invertebrates	10
Mammals	40
Reptiles	5
Total	100

Table 4.4: Categories in the Zoo data set

- Commonly, there is a feature called *(number of) legs*. However, similarly as for the *type* feature, the encoding scheme, using a value in the range of 1..8, does not fit well with the other values being either one or zero. Secondly, *number of legs* is a quantitative rather then a qualitative attribute, and therefore will not contribute to describing an input pattern with labels - a higher value does not mean that this feature is more relevant for the described animal. Therefore, a different encoding scheme, using the features *2_legs*, *4_legs*, *5_legs*, *6_legs* and *8_legs*, respectively, adopting binary values, was chosen (this approach was inspired by the *animals* demo data set described in Section 4.2, where a similar encoding scheme is used).

Table 4.4 contains available a-priori information about the zoo data set. This information indicates that the data set comprises distinct clusters with distinct quantities of elements. Therefore, the results should show an unbalanced hierarchical tree.

4.3.1 Results

In Figure 4.5, a first layer map of an AHIGG trained with the zoo demo data set is depicted. The training algorithm of the AHIGG generated a mapping with six disconnected nodes, forming six distinct clusters (of which one node does not represent any part of the input space). On the left side, node 0/1 represents *reptiles*. Onto node 1/0, *fish* and other aquatic animals, but not aquatic mammals, have been mapped, while node 1/1 represents the *insects* and some *invertebrates*. Node 1/2 represents the *birds*, and Node 2/1 is a cluster of *mammals*.

In Figures 4.6 to 4.8, selected maps of the second hierarchical layer are depicted.

Figure 4.6 shows a clear clustering between similar animals, like, for example, the invertebrates *slug* and *worm*, or the insects *wasp* and *bee*. Some of these clusters are shown on a higher granularity level in the next layer.

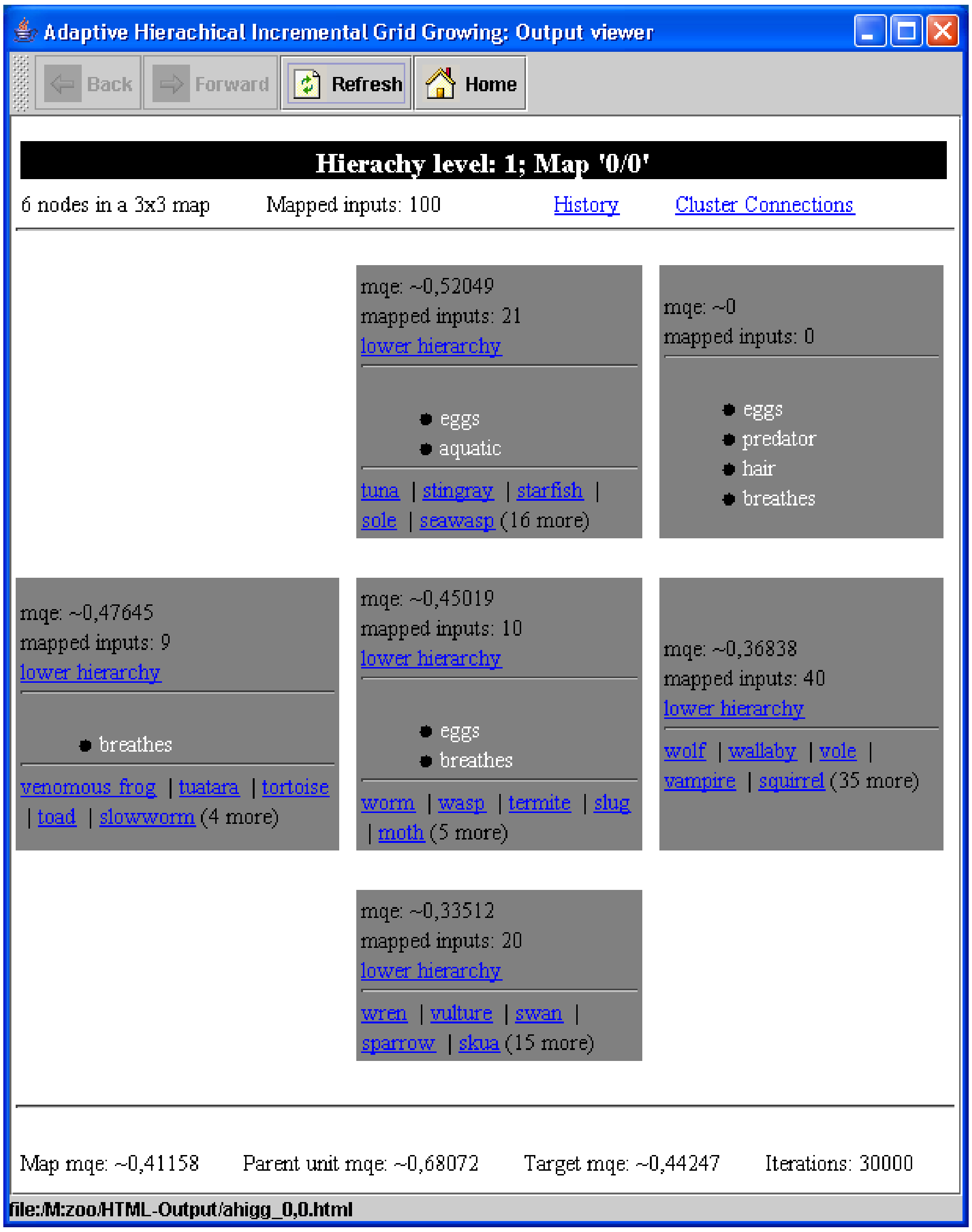
Figure 4.7 shows the reptiles, where for example *snakes* form a separate cluster on node 0/1. Node 2/1 shows toads and related animals, while on node 1/0, turtles are represented.

Figure 4.8 shows the rather big cluster of mammals. On node 1/2, aquatic mammals, like the seal or the dolphin, are grouped together. On the top-left node 0/0, mammals with only two legs are grouped together. Onto the centre nodes, animals with four legs are grouped in one cluster, where node 0/1 represents non-hunting, and 1/1 and 2/1 hunting animals. The inputs mapped on node 0/1 may seem to be belonging not to the same group, with a *vole* and a *reindeer* in the same group; this, however, is due to the demo data set itself, which doesn't contain enough features and therefore describes these animals similarly.

Node 0/1 is shown in two more hierarchical layers in Figures 4.3.1 and 4.3.1, respectively. In Figure 4.3.1, the mammals are separated into domestic and not domestic. Node 0/0 is expanded into another hierarchical layer as shown in Figure 4.3.1; here, the mammals are further distinguished by having a tail or not.

4.3.2 Conclusion

The zoo demo data set, while still being rather intuitively understandable, allows, additionally to the animal demo data set, the demonstration of the usefulness for creating a hierarchically correct mapping of an imbalanced data


Adaptive Hierachical Incremental Grid Growing: Output viewer
Back
Forward
Refresh
Home

Hierachy level: 1; Map '0/0'

6 nodes in a 3x3 map Mapped inputs: 100 History Cluster Connections

mqe: ~0,52049
mapped inputs: 21
lower hierarchy
- eggs
- aquatic

tuna | stingray | starfish | sole | seawasp (16 more)

mqe: ~0
mapped inputs: 0
- eggs
- predator
- hair
- breathes

mqe: ~0,47645
mapped inputs: 9
lower hierarchy
- breathes

venomous frog | tuatara | tortoise | toad | slowworm (4 more)

mqe: ~0,45019
mapped inputs: 10
lower hierarchy
- eggs
- breathes

worm | wasp | termite | slug | moth (5 more)

mqe: ~0,36838
mapped inputs: 40
lower hierarchy

wolf | wallaby | vole | vampire | squirrel (35 more)

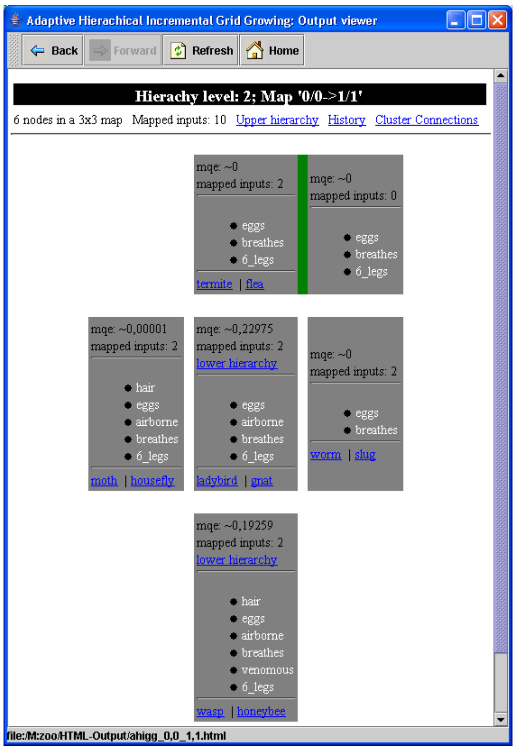
mqe: ~0,33512
mapped inputs: 20
lower hierarchy

wren | vulture | swan | sparrow | skua (15 more)

Map mqe: ~0,41158 Parent unit mqe: ~0,68072 Target mqe: ~0,44247 Iterations: 30000

file:/M:zoo/HTML-Output/ahigg_0,0.html

Figure 4.5: **AHIGG of the zoo data set**: first layer map


Adaptive Hierachical Incremental Grid Growing: Output viewer
Back
Forward
Refresh
Home

Hierachy level: 2; Map '0/0->1/1'

6 nodes in a 3x3 map Mapped inputs: 10 Upper hierarchy History Cluster Connections

mqe: ~0
mapped inputs: 2

- eggs
- breathes
- 6_legs

termite | flea

mqe: ~0
mapped inputs: 0

- eggs
- breathes
- 6_legs

mqe: ~0,00001
mapped inputs: 2

- hair
- eggs
- airborne
- breathes
- 6_legs

moth | housefly

mqe: ~0,22975
mapped inputs: 2
lower hierarchy

- eggs
- airborne
- breathes
- 6_legs

ladybird | gnat

mqe: ~0
mapped inputs: 2

- eggs
- breathes

worm | slug

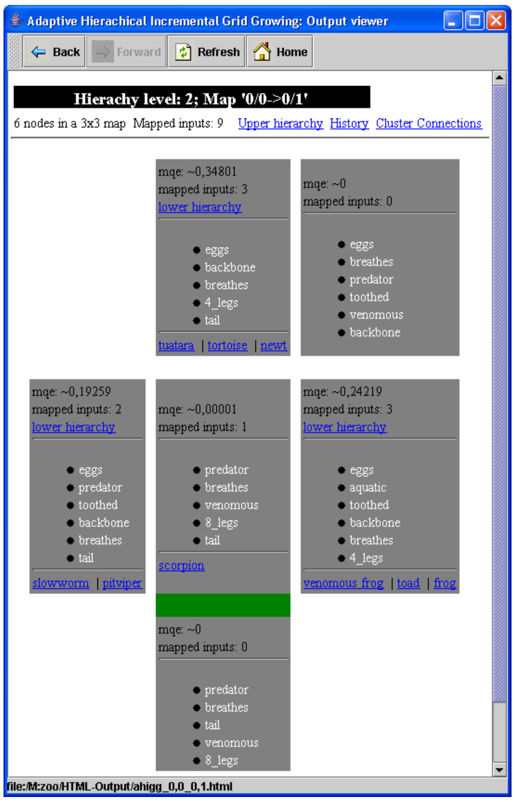
mqe: ~0,19259
mapped inputs: 2
lower hierarchy

- hair
- eggs
- airborne
- breathes
- venomous
- 6_legs

wasp | honeybee

file:/M:zoo/HTML-Output/ahigg_0,0_1,1.html

Figure 4.6: **AHIGG of the zoo data set:** second layer map of the insects.


Adaptive Hierachical Incremental Grid Growing: Output viewer
Back
Forward
Refresh
Home

Hierachy level: 2; Map '0/0->0/1'

6 nodes in a 3x3 map Mapped inputs: 9 Upper hierarchy History Cluster Connections

mqe: ~0,34801
mapped inputs: 3
lower hierarchy

- eggs
- backbone
- breathes
- 4_legs
- tail

tuatara | tortoise | newt

mqe: ~0
mapped inputs: 0

- eggs
- breathes
- predator
- toothed
- venomous
- backbone

mqe: ~0,19259
mapped inputs: 2
lower hierarchy

- eggs
- predator
- toothed
- backbone
- breathes
- tail

slowworm | pitviper

mqe: ~0,00001
mapped inputs: 1

- predator
- breathes
- venomous
- 8_legs
- tail

scorpion

mqe: ~0,24219
mapped inputs: 3
lower hierarchy

- eggs
- aquatic
- toothed
- backbone
- breathes
- 4_legs

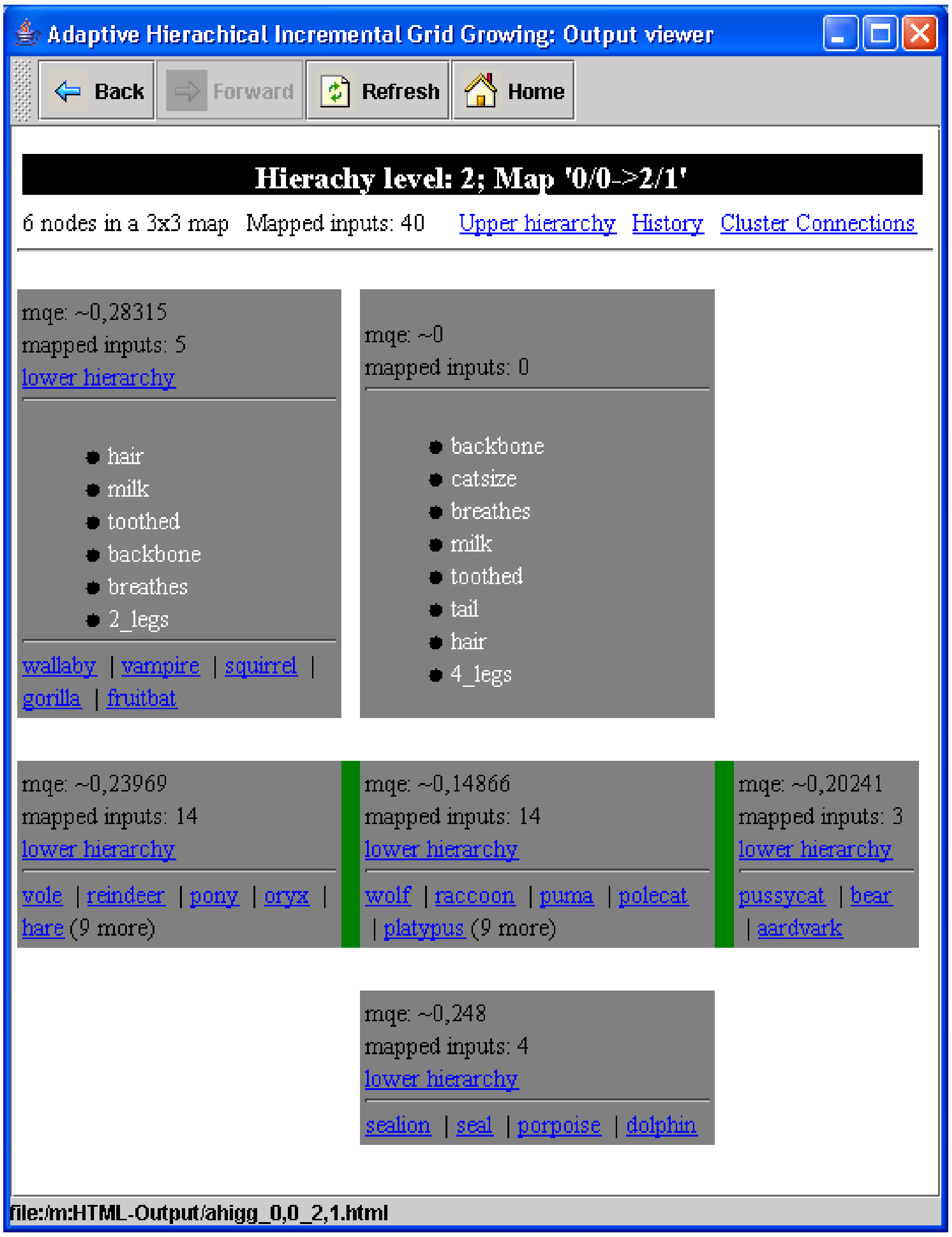
venomous frog | toad | frog

mqe: ~0
mapped inputs: 0

- predator
- breathes
- tail
- venomous
- 8_legs

file:/M:zoo/HTML-Output/ahigg_0,0_0,1.html

Figure 4.7: **AHIGG of the zoo data set:** second layer map of the reptiles.


Adaptive Hierachical Incremental Grid Growing: Output viewer
Back
Forward
Refresh
Home

Hierachy level: 2; Map '0/0->2/1'

6 nodes in a 3x3 map Mapped inputs: 40 Upper hierarchy History Cluster Connections

mqe: ~0,28315
mapped inputs: 5
lower hierarchy

- hair
- milk
- toothed
- backbone
- breathes
- 2_legs

wallaby | vampire | squirrel | gorilla | fruitbat

mqe: ~0
mapped inputs: 0

- backbone
- catsize
- breathes
- milk
- toothed
- tail
- hair
- 4_legs

mqe: ~0,23969
mapped inputs: 14
lower hierarchy

vole | reindeer | pony | oryx | hare (9 more)

mqe: ~0,14866
mapped inputs: 14
lower hierarchy

wolf | raccoon | puma | polecat | platypus (9 more)

mqe: ~0,20241
mapped inputs: 3
lower hierarchy

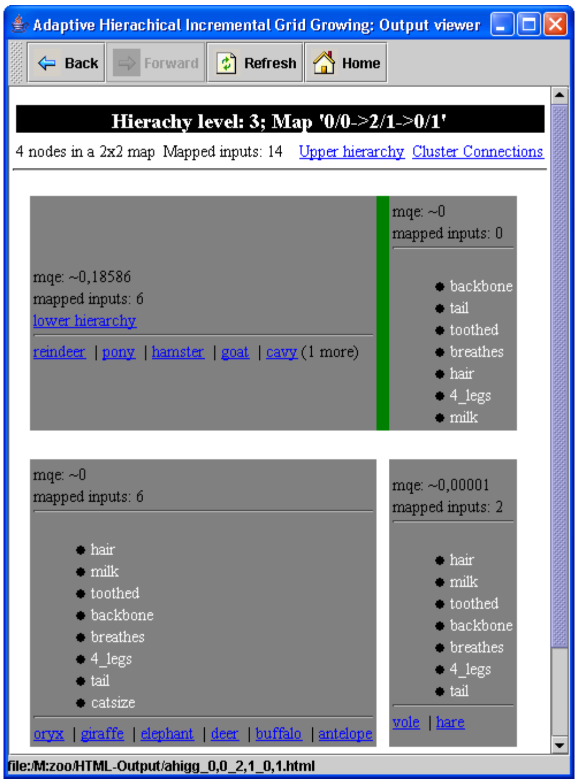
pussycat | bear | aardvark

mqe: ~0,248
mapped inputs: 4
lower hierarchy

sealion | seal | porpoise | dolphin

file:/m:HTML-Output/ahigg_0,0_2,1.html

Figure 4.8: **AHIGG of the zoo data set**: second layer map of the mammals.


Adaptive Hierachical Incremental Grid Growing: Output viewer
Back
Forward
Refresh
Home

Hierachy level: 3; Map '0/0->2/1->0/1'

4 nodes in a 2x2 map Mapped inputs: 14 Upper hierarchy Cluster Connections

mqe: ~0,18586
mapped inputs: 6
lower hierarchy
reindeer | pony | hamster | goat | cavy (1 more)

mqe: ~0
mapped inputs: 0

- backbone
- tail
- toothed
- breathes
- hair
- 4_legs
- milk

mqe: ~0
mapped inputs: 6

- hair
- milk
- toothed
- backbone
- breathes
- 4_legs
- tail
- catsize

oryx | giraffe | elephant | deer | buffalo | antelope

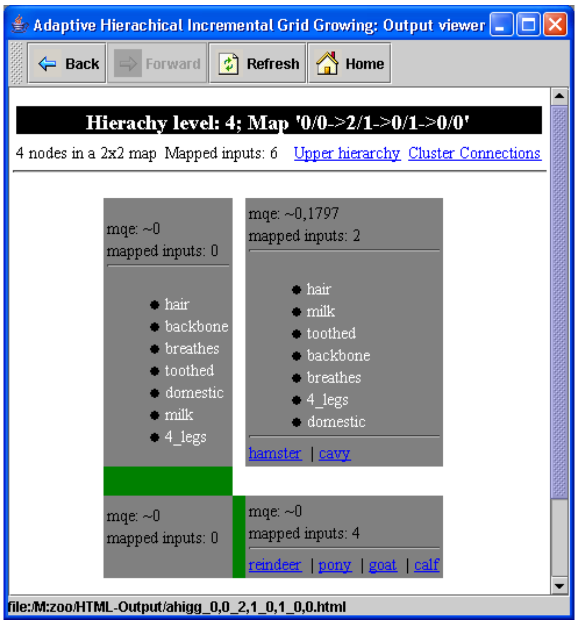
mqe: ~0,00001
mapped inputs: 2

- hair
- milk
- toothed
- backbone
- breathes
- 4_legs
- tail

vole | hare

file:/M:zoo/HTML-Output/ahigg_0,0_2,1_0,1.html

Figure 4.9: **AHIGG of the zoo data set**: third layer maps of some mammals.


Adaptive Hierachical Incremental Grid Growing: Output viewer
Back
Forward
Refresh
Home

Hierachy level: 4; Map '0/0->2/1->0/1->0/0'

4 nodes in a 2x2 map Mapped inputs: 6 Upper hierarchy Cluster Connections

mqe: ~0
mapped inputs: 0

- hair
- backbone
- breathes
- toothed
- domestic
- milk
- 4_legs

mqe: ~0,1797
mapped inputs: 2

- hair
- milk
- toothed
- backbone
- breathes
- 4_legs
- domestic

hamster | cavy

mqe: ~0
mapped inputs: 0

mqe: ~0
mapped inputs: 4
reindeer | pony | goat | calf

file:/M:zoo/HTML-Output/ahigg_0,0_2,1_0,1_0,0.html

Figure 4.10: **AHIGG of the zoo data set**: fourth layer map of some mammals.

set. In this example, the mammals cluster was by far the largest, the AHIGG generated a mapping which was imbalanced in this respect, and therefore, this model can bee seen as usable for hierarchical clustering. Still, an example with a larger, and higher dimensional, data set is needed to prove this.

Concerning the zoo data set itself, however, it has to be stated that it is a bit illogical, with apparently wrongly assigned values for features like "catsize"; further, real features for the size of animals, as in the animals demo data set, are missing. Therefore, its purpose, the intuitive verification of results, is not completely fulfilled.

4.4 Tourism Data

The data we will use in our experiment comes from the domain of tourism: our document collection is made up from *free-text descriptions* of Austrian hotels (and other types of accommodation). The data is available as plain text; most of the hotels were described in two languages, English and German, in separate documents, sharing a similar document name (though not all hotels were presented in both languages).

The data source is the Austrian part of the Tourist Information System *TIScover* (http://www.tiscover.com). The system's main part is an information system where users can inquire about hotels in Austria, for example, prices, location, equipment, etcetera, of the registered hotels. Users can define search-criteria by selecting values from combo-boxes (for example, the type of accommodation, or the dates, number of persons, etcetera), and entering other criteria in text-fields, like the region, a name of the accommodation, or a maximum price. The search performed on this is performed via *structured data* in a database. The information about accommodations is retrieved via a keyword search, where the keywords are combined according to Boole's algebra. The information retrieved is displayed in a sorted list.

On the very same data source, experiments have been carried out [DMB02], offering the user a free-text search, i.e. the user specifies a search by entering a free-text query like *"I am looking for a double room in the centre of Salzburg with indoor pool."*. However, this experiment was a more conventional approach, where this free-text was analysed and key-words were extracted to form a database query.

Besides the structured data in the database, the TIScover system moreover provides free-text descriptions about the hotels, most of the times written by the owners of the accommodations themselves. This is the data we are going to use in our experiments. The objective of the experiments was, Holiday Inn**** Welcome! The Holiday Inn Vienna is a first-class hotel for congresses, busines travellers and tourists, located in the south of Vienna right in the Vienna business park. Here you will find the perfect infrastructure with offices, all kinds of services, shopping malls and underground car parks. 76 comfortably furnished rooms and 5 apartments await you. They are all equipped with air conditioning, bath, WC, telephone, mini bar and satellite TV. In terms of dining and wining you won't miss a thing. The restaurant Wiener's, the Sommerterrasse, the Heurigen inn at the Wienerberg and the Lobby Bar await you. Our 8 conference rooms offer space for 500 persons. All rooms have daylight and feature the latest technical equipment. For your recreation you will find various sports facilites like golf and tennis in the ultimate vicinity of the hotel. The cosmopolitan city of Vienna awaits you - we are looking forward to your visit!

Figure 4.11: An exemplary hotel description, containing (underlined) spelling errors and German expressions.

on the one hand, to prove the applicability of the AHIGG on text collections, and on the other hand, to see whether tourism data like the one present in the experiments can be reasonably categorised, thereby providing the user with an additional search possibility.

Prior to the experiments, we expected to be able to obtain a categorisation into different hotel classes, as e.g. *seminar and conference*, *(winter and summer) sports*, *spa* and *wellness*, and other types of hotels, as well as a geographic categorisation, because many descriptions included the name of the village or town they are located in, as well as of nearby sites and attractions.

In Figure 4.11, you can see an exemplary hotel description for a four-star hotel in Vienna, apparently focusing on business customers. It is to note that there are *spelling mistakes*, as well as some German expressions (mainly names of sites, or attractions, and specific Austrian expressions). The spelling mistakes and the German words might cause some problems; this will be described in Section 4.4.1.

The rest of this Section is organised as follows: first, we will describe how the data was pre-processed to allow it to be used with the AHIGG (or any other self-organising model that uses vectors of features to represent data); then, we will give an overview of the results obtained from training various models with the English part of the data-set; in describing the results, we will follow an order that matches the "evolution" of the used models; we

Province	original	minus empty	minus short	minus German	usable
Burgenland	972	638	516	455	46.8%
Kärnten	1,437	1,040	1,012	1,004	69.8%
Niederösterreich	703	265	252	160	22,7%
Oberösterreich	982	743	737	553	56.3%
Salzburg	1,967	1,937	885	686	34.8%
Steiermark	1,078	831	811	522	51.2%
Tirol	5,153	4,312	3,992	3,911	75.8%
Vorarlberg	860	444	416	415	48.3%
Wien	145	124	120	118	81.4%
Total	13,333	9,334	8,741	7824	58.7%

Table 4.5: **English hotel data set**: number of documents before and after cleaning the data set, grouped by province.

therefore start with presenting the outcome of training a *Self-Organizing Map* (respectively a *Growing Grid*), followed by the *Growing Hierarchical Self-Organizing Map*, and the *Adaptive Hierarchical Incremental Grid Growing*. Each section includes a discussion about the results.

4.4.1 Data Pre-processing

Our document collection is in plain-text, and therefore no formatting information had to be removed. Originally, the *English* hotel data set contained the number of documents according to the column "original" in Table 4.5. However, a number of documents were empty, in other words, for these hotels no description was available from the source; these documents were removed, and the data set resulting from this action is shown in the column "minus empty" in Table 4.5; other documents just contained a few generic words like "welcome to our house", or "web-site under construction"; these documents would also not be reasonably usable for our application, and therefore it was decided to remove all the documents that had a document length of less than 200 characters.

The data-set thus cleaned served as our **initial** input set, and is seen in the columns "minus short" of Table 4.5. However, the data set was still of bad quality, as there was a high number of German documents among the English ones. To filter them out in a half-automated way, both a GHSOM and an AHIGG were trained (altogether three times) with this input data. As a result of the training process, the German documents were mapped on different nodes on the first layer already, and formed distinctive clusters to

Province	original	minus empty
Burgenland	916	903
Kärnten	1.472	1.407
Niederösterreich	611	598
Oberösterreich	804	793
Salzburg	1.767	1.621
Steiermark	795	780
Tirol	5.100	4.936
Vorarlberg	859	807
Wien	147	144
Total	12.471	11.989

Table 4.6: **German hotel data set**: number of documents before and after cleaning the data set.

the other (English) documents. A certain (constant) number of documents was each time organised into "German" clusters, and were, after manual spot checks which confirmed them to be German, deleted. The final result for our input data set is shown in Table 4.5, column "minus German".

In the end, the English data set contained just about 60% of its initial documents. For the German data set, as seen in Table 4.6, only empty documents have been removed, but still one could expect the quality in the German data set to be better from the given numbers, because of the fact of having far fewer empty German documents.

The subsequent steps of our experiments were carried out with the English part of the data set.

Term stemming

In order to reduce the number of words in the documents, we apply a *stemming* program to the document collection; the stemmer creates a mapping from the individual words to their corresponding stem, yielding the so-called *terms* used to represent documents in the collection. Prior to stemming the final data set, the vocabulary of the document collection contained 18,614 unique terms, while after stemming, 15,114 unique terms remained - the stemming process created a mapping from words to terms, which leads to a reduction of features by approximately 19%. Besides the better computational performance caused by a reduced vector dimensionality, stemming (in general) also leads to a better clustering result, as documents that initially had very similar words now contain the same terms. However, concerning visualisation purposes, we have to accept a bit less readable labels.

Document vector generation

In order to use any data set for analysis in the Adaptive Hierarchical Incremental Grid Growing, or other similar models described in this thesis, it has to be described as a feature vector, containing a certain number of components. Therefore, we have to use a technique to describe our document collection in a *vector-space representation*: for each document, we should obtain a vector that will describe the very same document. This can be achieved be extracting the list of all words present in the document collection. However, this list would be rather long, and would not necessarily be adequate to unequivocally represent the documents. Therefore, we remove terms that do not contribute to a content description. One approach would be to use so-called *stop word* lists, i.e. lists that contain among others grammatical function words such as conjugations, articles, or pronouns, etc. These terms exhibit approximately equal frequency in all the documents of a collection, and therefore do not discriminate between the single documents. Furthermore, terms that are specific for the data set should be discarded. In our application, the term "hotel" e.g. will be present in a very high percentage of the documents, and will therefore not contribute to describing the documents.

However, we prefer not to rely on content and language specific stop word list, but we discard terms based on their relative occurrence in the documents, i.e. we discard terms that occur in more than a certain percentage of documents, as well as terms that occur in a number of documents below another percentage threshold. In our experiment, we only accepted words that consisted of at least three letters, and we discarded terms that appeared in more than 45%, and in less than 1 % of the documents. Note that the upper limit for term frequencies does not contribute much to term reduction: by omitting terms that occur in more than 45% of the documents, the number of features dropped by only ten terms. By contrast, skipping terms that are present in less than 1 % of the documents led to omitting 14,273 terms. The final number of terms, i.e. the dimension of our feature vectors, was therefore 833.

To generate weighted values for the components in our input vectors, several approaches exist in the literature [BYRN99], amongst others, the rather simple *Boolean Model*, which uses only boolean values for the weights, or the more sophisticated *Probabilistic Model*. We chose the still rather simple (yet more accurate than the boolean model) *term frequency* $\times$ *inverse document frequency* weighting scheme, as described in [Sal89]. This scheme assigns high values to terms that are important to describe and discriminate between documents, and low values to other terms.

The scheme is based on two assumptions: terms are of importance when they occur more frequently in one document, and less frequently in the rest of the document collection. We therefore extract the term frequency tf_{ij} of a term T_j in document D_i, i.e., how many times the term appears in this document. Additionally, the document frequency df_i, defined as the number of documents in a collection of N documents in which the term T_j occurs, is calculated. An appropriate indication of the term value as a document discriminator can be given by using an *inverse document frequency idf*, like $\log \frac{N}{df_i}$. Then, we can define a measure of the importance of a term for the feature vector x by computing a weighted value for its components k according to

$$x_k = tf_{ij} \cdot \log \frac{N}{df_i}. \tag{4.3}$$

As we have seen in Figure 4.11, some documents contain spelling mistakes, as well as including German expressions. This might result in the creation of two different components in the vectors if they are frequent enough, or if they are just occurring in a few documents, this will lead to omitting these terms for the vector representation. In both cases, similarities between the documents will get "lost". The same might apply for the German words: some of them might be translated, as for example *Heurigen*, which is an expression for a special type of winery, resulting in either these terms being omitted, or in two components for precisely the same term being created. This will lead to a poorer clustering result.

Vector normalisation

For the remaining documents in the collection, there were only minor differences in the average length of the documents; English documents have on average 640 characters, German 652 characters. However, big differences are found when one compares the length of single documents in the collections - the shortest documents were in the range of a few words (sometimes only three words, just naming the type of the hotel), while the longest descriptions were up to 300 words long. These large discrepancies in the input data set has a fundamental impact on the vector representations generated. As described above, the weights for the vector components are calculated based (partly) on the *term frequency* within the documents. However, it seems reasonable for longer documents (on average) to lead to higher term frequencies. In our application, though, these different term-frequencies stem from different document quality, and we don't want this to affect our results. Therefore, we will normalise the vectors to the same length. By this, the relation of the weights within a feature vector stays the same, while all the vectors now lie

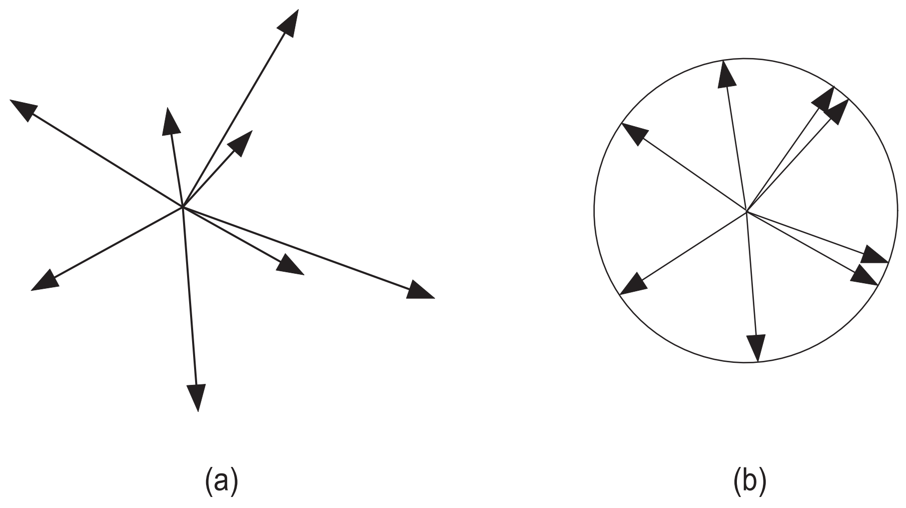


Figure 4.12: **Vector normalisation** to the same length.

in a similar range, and are therefore more easily comparable. An illustration of this is given in Figure 4.12, where (a) shows the document vectors before, and (b) after normalisation to the same length.

4.4.2 Results with the Self-Organizing Map and Growing Grid

In this section, we will present the results obtained from training a Self-Organizing Map, respectively a Growing Grid, with the hotel data set. For both models, we used the GHSOM package, available at `http://www.ifs.tuwien.ac.at/~andi/ghsom/`. By defining appropriate values for the parameters τ_1 and τ_2 (see Section 2.6.2), the package can be used to train static or only horizontally growing maps, thereby, we utilised the GHSOM algorithm for training a standard Self-Organizing Map (both τ_1 and τ_2 are set to values 1), and a dynamically Growing Grid (only τ_2 is set to value 1). Note that similarly, the AHIGG could be utilised to train a connectivity-enhanced Self-Organizing Map, a standard Incremental Grid Growing, or the AHIGG itself).

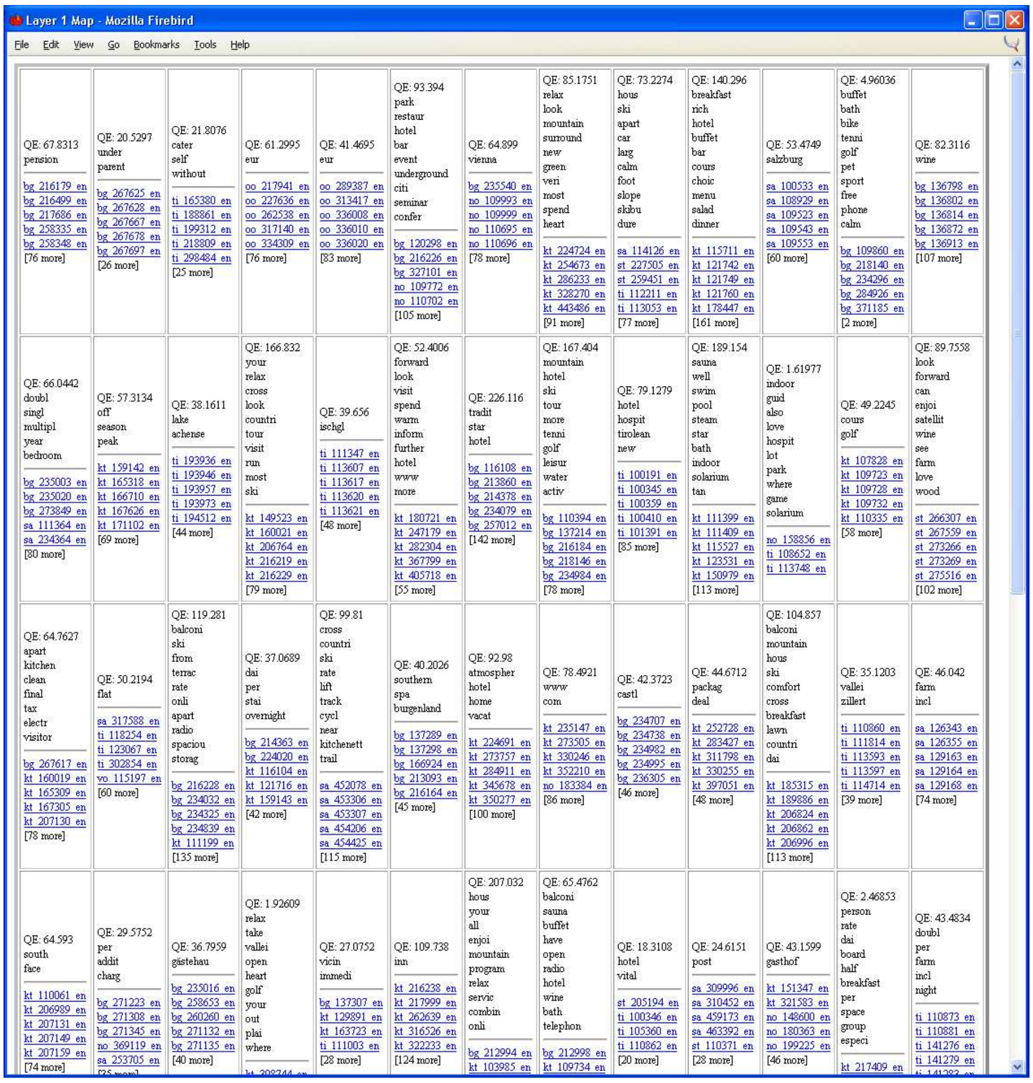
As the SOM requires an a-priori defined grid-size, in our experiments, we first trained a Growing Grid with the hotel data set, to get a "feeling" about the approximate grid sizes. Having a rough idea about how big the grid should be, we also trained various SOMs. Then, the outcome for both models was de facto the same. To give a comparison about the run-time performance of the algorithms is difficult, as it will depend mainly on the number of iterations. However, through the incremental training process in the Growing Grid, a much lower number of iterations for each *training cycle* is sufficient; therefore the parameters are not easily comparable.

A Section of the outcome of one Growing Grid training is depicted in Figure 4.13. The algorithm developed a 13×7 map with 91 nodes, a map-size which is not big compared to other projects (as mentioned in Section 2.1.3, the WEBSOM projects had about 100,000 nodes), and the mapping thus generated shows intuitively correct results. Still, the map has become rather large for human interpretation. Clearly, a representation like this could not be utilised in computer-based applications, as the map produced does not fit the screen size, and thereby, the user can hardly get an overview of the mapping. We will now describe some parts of the generated mapping.

Some nodes in the map contain solely names of places or regions, for example *Vienna* (node 6/0), *Achensee* (node 2/1) or *Ischgl* (node 4/1), indicating the location of the accommodations. This might be a useful grouping for the user; on the given application, however, a mere geographical order could also be achieved by using the existing structured data. A different grouping, though, might emerge, by some documents not containing the geographic information in the structured data, but only in the free-text description, and vice-versa.

Other nodes show only other single terms, like *eur* (nodes 4/0 and 5/0), indicating that these documents contain price information in the free text, or *pension*, indicating these documents describe *bed and breakfast* (or *guest)* houses. These node-labels, however, do not contain too much information for the user; someone who is interested in a bed-and-breakfast-style accommodation will not be eager to browse through the 81 documents mapped on this node - a more detailed representation of these documents in another hierarchical layer could greatly assist the user.

However, clusters of specific hotel types are visible; for example, on node 5/0, seminar and conference hotels, located in a city area, with an *underground* connection mentioned, are grouped together. These documents will mostly describe hotels in Vienna (as Vienna is the only Austrian city with an underground transport system), and therefore, the group of documents to the right, on node 6/0, with *Vienna* as the only label, indicates the topology-preserving mapping ability of the Self-Organizing Map. The node 7/1 indicates hotels that offer a lot of sports facilities, while on node 9/1, so-called wellness hotels, offering facilities like sauna, steam baths, swimming pools or a solarium.



QE: 67.8313 pension bg 216179 en bg 216499 en bg 217686 en bg 258335 en bg 258348 en [76 more]	QE: 20.5297 under parent bg 267625 en bg 267628 en bg 267667 en bg 267678 en bg 267697 en [26 more]	QE: 21.8076 cater self without ti 165380 en ti 188861 en ti 199312 en ti 218809 en ti 298484 en [25 more]	QE: 61.2995 eur oo 217941 en oo 227636 en oo 262538 en oo 317140 en oo 334309 en [76 more]	QE: 41.4695 eur oo 289387 en oo 313417 en oo 336008 en oo 336010 en oo 336020 en [83 more]	QE: 93.394 park restaur hotel bar event underground citi seminar confer bg 120298 en bg 216226 en bg 327101 en no 109772 en no 110702 en [105 more]	QE: 64.899 vienna bg 235540 en no 109993 en no 109999 en no 110695 en no 110696 en [78 more]	QE: 85.1751 relax look mountain surround new green veri most spend heart kt 224724 en kt 254673 en kt 286233 en kt 328270 en kt 443486 en [91 more]	QE: 73.2274 hous ski apart car larg calm foot slope skibu dure sa 114126 en st 227505 en st 259451 en ti 112211 en ti 113053 en [77 more]	QE: 140.296 breakfast rich hotel buffet bar cours choic menu salad dinner kt 115711 en kt 121742 en kt 121749 en kt 121760 en kt 178447 en [161 more]	QE: 53.4749 salzburg sa 100533 en sa 108929 en sa 109523 en sa 109543 en sa 109553 en [60 more]	QE: 4.96036 buffet bath bike tenni golf pet sport free phone calm bg 109860 en bg 218140 en bg 234296 en bg 284926 en bg 371185 en [2 more]	QE: 82.3116 wine bg 136798 en bg 136802 en bg 136814 en bg 136872 en bg 136913 en [107 more]
QE: 66.0442 doubl singl multipl year bedroom bg 235003 en bg 235020 en bg 273849 en sa 111364 en sa 234364 en [80 more]	QE: 57.3134 off season peak kt 159142 en kt 165318 en kt 166710 en kt 167626 en kt 171102 en [69 more]	QE: 38.1611 lake achense ti 193936 en ti 193946 en ti 193957 en ti 193973 en ti 194512 en [44 more]	QE: 166.832 your relax cross look countri tour visit run most ski kt 149523 en kt 160021 en kt 206764 en kt 216219 en kt 216229 en [79 more]	QE: 39.656 ischgl ti 111347 en ti 113607 en ti 113617 en ti 113620 en ti 113621 en [48 more]	QE: 52.4006 forward look visit spend warm inform further hotel www more kt 180721 en kt 247179 en kt 282304 en kt 367799 en kt 405718 en [55 more]	QE: 226.116 tradit star hotel bg 116108 en bg 213860 en bg 214378 en bg 234079 en bg 257012 en [142 more]	QE: 167.404 mountain hotel ski tour more tenni golf leisur water activ bg 110394 en bg 137214 en bg 216184 en bg 218146 en bg 234984 en [78 more]	QE: 79.1279 hotel hospit tirolean new ti 100191 en ti 100345 en ti 100359 en ti 100410 en ti 101391 en [85 more]	QE: 189.154 sauna well swim pool steam star bath indoor solarium tan kt 111399 en kt 111409 en kt 115527 en kt 123531 en kt 150979 en [113 more]	QE: 1.61977 indoor guid also love hospit lot park where game solarium no 158856 en ti 108652 en ti 113748 en	QE: 49.2245 cours golf kt 107828 en kt 109723 en kt 109728 en kt 109732 en kt 110335 en [58 more]	QE: 89.7558 look forward can enjoi satellit wine see farm love wood st 266307 en st 267559 en st 273266 en st 273269 en st 275516 en [102 more]
QE: 64.7627 apart kitchen clean final tax electr visitor bg 267617 en kt 160019 en kt 165309 en kt 167305 en kt 207130 en [78 more]	QE: 50.2194 flat sa 317588 en ti 118254 en ti 123067 en ti 302854 en vo 115197 en [60 more]	QE: 119.281 balconi ski from terrac rate onli apart radio spaciou storag bg 216228 en bg 234032 en bg 234325 en bg 234839 en kt 111199 en [135 more]	QE: 37.0689 dai per stai overnight bg 214363 en bg 224020 en kt 116104 en kt 121716 en kt 159143 en [42 more]	QE: 99.81 cross countri ski rate lift track cycl near kitchenett trail sa 452078 en sa 453306 en sa 453307 en sa 454206 en sa 454425 en [115 more]	QE: 40.2026 southern spa burgenland bg 137289 en bg 137298 en bg 166924 en bg 213093 en bg 216164 en [45 more]	QE: 92.98 atmospher hotel home vacat kt 224691 en kt 273757 en kt 284911 en kt 345678 en kt 350277 en [100 more]	QE: 78.4921 www com kt 235147 en kt 273505 en kt 330246 en kt 352210 en no 183384 en [86 more]	QE: 42.3723 castl bg 234707 en bg 234738 en bg 234982 en bg 234995 en bg 236305 en [46 more]	QE: 44.6712 packag deal kt 252728 en kt 283427 en kt 311798 en kt 330255 en kt 397051 en [48 more]	QE: 104.857 balconi mountain hous ski comfort cross breakfast lawn countri dai kt 185315 en kt 189886 en kt 206824 en kt 206862 en kt 206996 en [113 more]	QE: 35.1203 vallei zillert ti 110860 en ti 111814 en ti 113593 en ti 113597 en ti 114714 en [39 more]	QE: 46.042 farm incl sa 126343 en sa 126355 en sa 129163 en sa 129164 en sa 129168 en [74 more]
QE: 64.593 south face kt 110061 en kt 206989 en kt 207131 en kt 207149 en kt 207159 en [74 more]	QE: 29.5752 per addit charg bg 271223 en bg 271308 en bg 271345 en no 369119 en sa 253705 en	QE: 36.7959 gästehau bg 235016 en bg 258653 en bg 260260 en bg 271132 en bg 271135 en [40 more]	QE: 1.92609 relax take vallei open heart golf your out plai where	QE: 27.0752 vicin immedi bg 137307 en kt 129891 en kt 163723 en ti 111003 en [28 more]	QE: 109.738 inn kt 216238 en kt 217999 en kt 262639 en kt 316526 en kt 322233 en [124 more]	QE: 207.032 hous your all enjoi mountain program relax servic combin onli bg 212994 en kt 103985 en	QE: 65.4762 balconi sauna buffet have open radio hotel wine bath telephon bg 212998 en kt 109734 en	QE: 18.3108 hotel vital st 205194 en ti 100346 en ti 105360 en ti 110862 en [20 more]	QE: 24.6151 post sa 309996 en sa 310452 en sa 459173 en sa 463392 en st 110371 en [28 more]	QE: 43.1599 gasthof kt 151347 en kt 321583 en no 148600 en no 180363 en no 199225 en [46 more]	QE: 2.46853 person rate dai board half breakfast per space group especi kt 217409 en	QE: 43.4834 doubl per farm incl night ti 110873 en ti 110881 en ti 141276 en ti 141279 en

Figure 4.13: **Output of a Self-Organizing Map** trained with the hotel data set.

Discussion

The SOM and GG algorithms produced intuitively correct results, and showed some useful clustering according to hotel categories and geographical information. Thus, they might be helpful. In general, though, the standard representation of the SOM needs rather big network sizes, which does not seem to greatly assist the user, as it does not allow easy interpretation and browsing through the data. This is for example also acknowledged by the WEBSOM and SOMLib projects offering a representation at different "zoom" levels. Smaller grid-sizes, with different levels of granularity, seem to be more adequate in our application as well.

4.4.3 Results with the Growing Hierarchical Self-Organizing Map

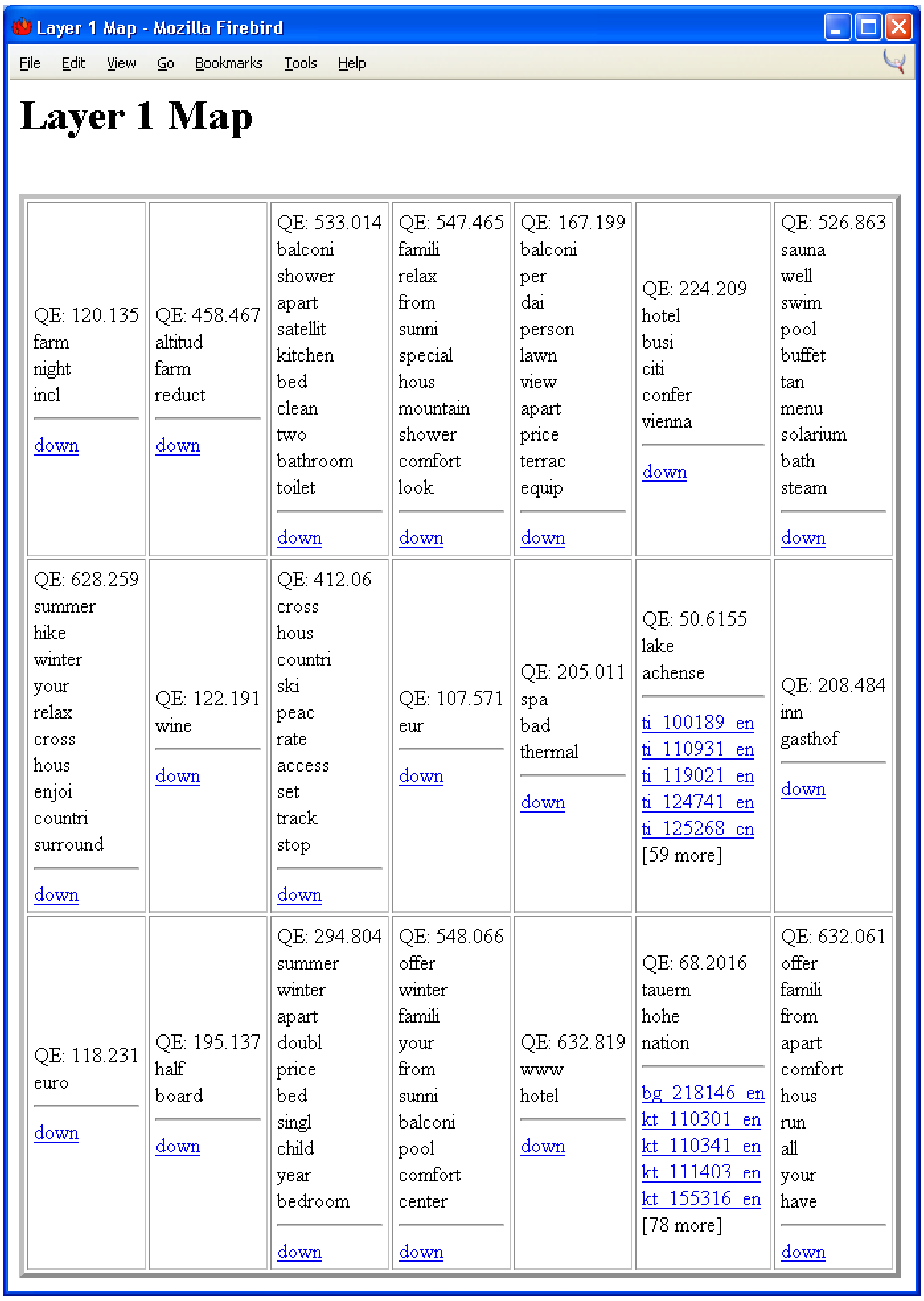
After carrying out the experiments with the SOM, we then utilised the Growing Hierarchical Self-Organizing Map to generate mappings of the hotel data set. In Figure 4.14, the first layer of a GHSOM thus trained is depicted. Compared with a SOM, the hierarchical structure allows a much smaller map on the first layer; this allows the user to recognise the clusters in the data more easily.

In the given example, we can identify various types of hotels. In the top left corner, on nodes 0/0 and 1/0, the accommodation type on *farms* is grouped. As in the example of the SOM, we can see clusters of business and conference hotels (node 5/0) and wellness hotels (6/0 and 4/1). Further, hotels characterised as *inns* are located on node 6/1. In the centre-top of the map, we can find hotels that describe their equipment in detail (nodes 2/0 - 4/0). On the middle-left, we can find hotels stressing on sports activities (0/1, 2/1), while in the bottom-centre, we find hotels that offer services for families and children (2/2, 3/2, 6/2). The hotels on node 1/1 stress *wine* and wine tasting, while the documents on 1/2 have a *half board* service.

Geographical clustering emerged on nodes 5/1 and 5/2, with the lake *Achensee* and the mountains of the *Tauern* being the common locations. Note that these two nodes are also the only ones not expanded to another hierarchical layer.

Additionally, some rather unexpected clusters emerged: grouping documents by the common information of mentioning the price in the free-text description (nodes 2/0, 3/1), as well as grouping hotels which mention their WWW-address (4/2).

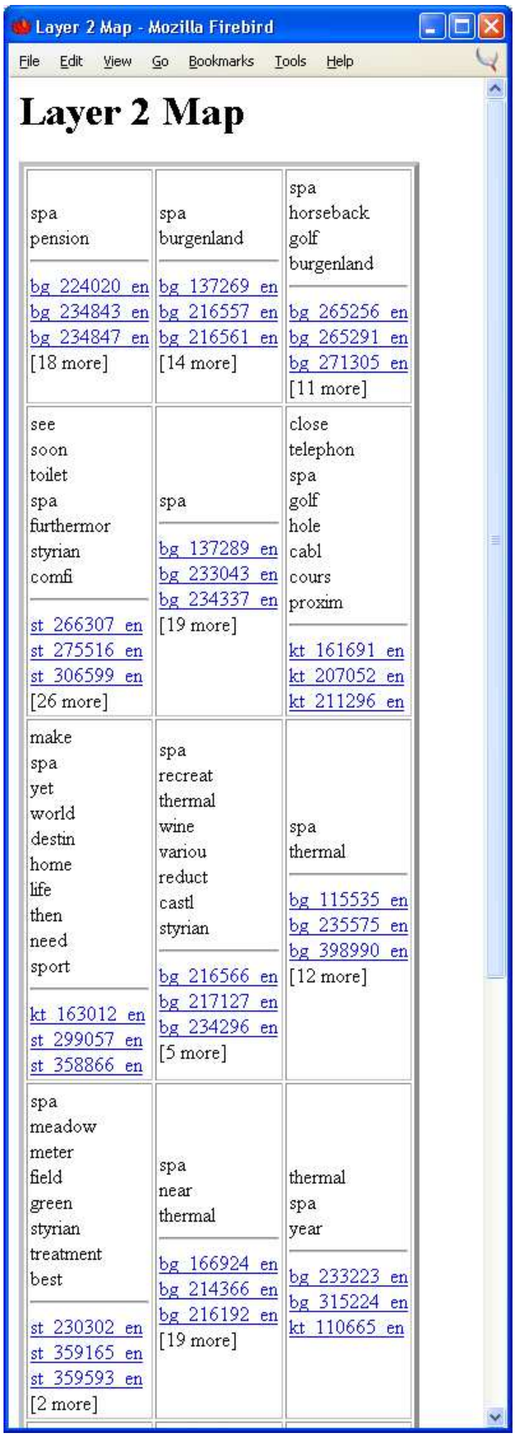
In Figure 4.15, a second layer map of node 4/1, showing hotels stressing their vicinity to spas, is depicted. The hotels are mapped on different nodes



Layer 1 Map

QE: 120.135 farm night incl down	QE: 458.467 altitud farm reduct down	QE: 533.014 balconi shower apart satellit kitchen bed clean two bathroom toilet down	QE: 547.465 famili relax from sunni special hous mountain shower comfort look down	QE: 167.199 balconi per dai person lawn view apart price terrac equip down	QE: 224.209 hotel busi citi confer vienna down	QE: 526.863 sauna well swim pool buffet tan menu solarium bath steam down
QE: 628.259 summer hike winter your relax cross hous enjoi countri surround down	QE: 122.191 wine down	QE: 412.06 cross hous countri ski peac rate access set track stop down	QE: 107.571 eur down	QE: 205.011 spa bad thermal down	QE: 50.6155 lake achense ti_100189_en ti_110931_en ti_119021_en ti_124741_en ti_125268_en [59 more]	QE: 208.484 inn gasthof down
QE: 118.231 euro down	QE: 195.137 half board down	QE: 294.804 summer winter apart doubl price bed singl child year bedroom down	QE: 548.066 offer winter famili your from sunni balconi pool comfort center down	QE: 632.819 www hotel down	QE: 68.2016 tauern hohe nation bg_218146_en kt_110301_en kt_110341_en kt_111403_en kt_155316_en [78 more]	QE: 632.061 offer famili from apart comfort hous run all your have down

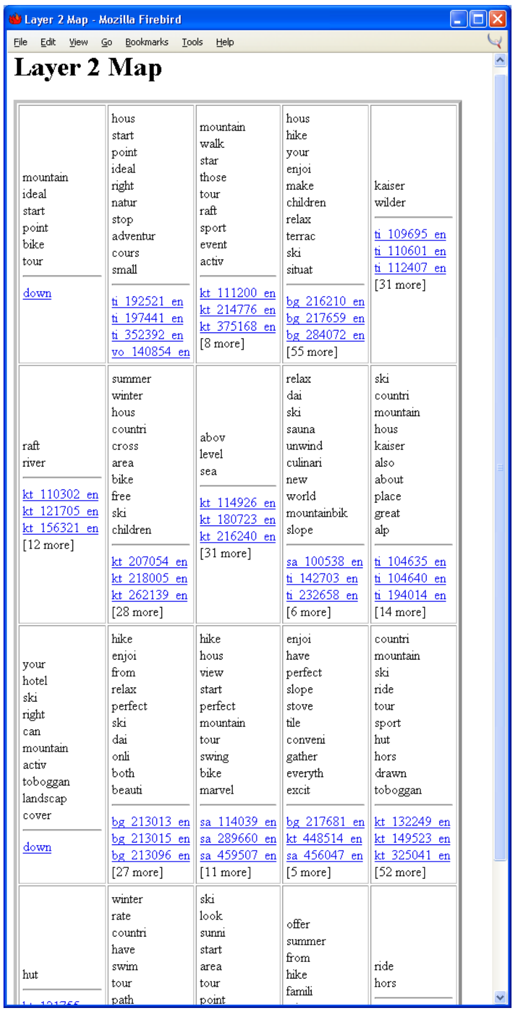
Figure 4.14: **First layer map of a Growing Hierarchical Self-Organizing Map** trained with the hotel data set.



Layer 2 Map

spa pension bg 224020 en bg 234843 en bg 234847 en [18 more]	spa burgenland bg 137269 en bg 216557 en bg 216561 en [14 more]	spa horseback golf burgenland bg 265256 en bg 265291 en bg 271305 en [11 more]
see soon toilet spa furthermor styrian comfi st 266307 en st 275516 en st 306599 en [26 more]	spa bg 137289 en bg 233043 en bg 234337 en [19 more]	close telephon spa golf hole cabl cours proxim kt 161691 en kt 207052 en kt 211296 en
make spa yet world destin home life then need sport kt 163012 en st 299057 en st 358866 en	spa recreat thermal wine variou reduct castl styrian bg 216566 en bg 217127 en bg 234296 en [5 more]	spa thermal bg 115535 en bg 235575 en bg 398990 en [12 more]
spa meadow meter field green styrian treatment best st 230302 en st 359165 en st 359593 en [2 more]	spa near thermal bg 166924 en bg 214366 en bg 216192 en [19 more]	thermal spa year bg 233223 en bg 315224 en kt 110665 en

Figure 4.15: **GHSOM of the hotel data set**: layer 2 map of hotels next to spas.



Layer 2 Map

mountain ideal start point bike tour down	hous start point ideal right natur stop adventur cours small ti 192521 en ti 197441 en ti 352392 en vo 140854 en	mountain walk star those tour raft sport event activ kt 111200 en kt 214776 en kt 375168 en [8 more]	hous hike your enjoi make children relax terrac ski situat bg 216210 en bg 217659 en bg 284072 en [55 more]	kaiser wilder ti 109695 en ti 110601 en ti 112407 en [31 more]
raft river kt 110302 en kt 121705 en kt 156321 en [12 more]	summer winter hous countri cross area bike free ski children kt 207054 en kt 218005 en kt 262139 en [28 more]	abov level sea kt 114926 en kt 180723 en kt 216240 en [31 more]	relax dai ski sauna unwind culinari new world mountainbik slope sa 100538 en ti 142703 en ti 232658 en [6 more]	ski countri mountain hous kaiser also about place great alp ti 104635 en ti 104640 en ti 194014 en [14 more]
your hotel ski right can mountain activ toboggan landscap cover down	hike enjoi from relax perfect ski dai onli both beauti bg 213013 en bg 213015 en bg 213096 en [27 more]	hike hous view start perfect mountain tour swing bike marvel sa 114039 en sa 289660 en sa 459507 en [11 more]	enjoi have perfect slope stove tile conveni gather everyth excit bg 217681 en kt 448514 en sa 456047 en [5 more]	countri mountain ski ride tour sport hut hors drawn toboggan kt 132249 en kt 149523 en kt 325041 en [52 more]
hut	winter rate countri have swim tour path	ski look sunni start area tour point	offer summer from hike famili	ride hors

Figure 4.16: **GHSOM of the hotel data set**: layer 2 map of sport hotels.

according to geographical (for example, *Styria* and *Burgenland*) aspects, or on the additional services they provide (*golf*, *horseback riding*, etcetera).

In Figure 4.16, the documents mapped on node 0/1 are represented in more detail. The mapped documents describe hotels mainly concentrating on (summer) sport activities; hotels are separated according to the kind of sports they provide, for example *rafting*, *hiking*, *swimming*, or *biking*. Additionally, some documents are grouped according to geographical aspects. Furthermore, a rather unexpected grouping of documents providing information about the *sea level* of the accommodation emerged.

Discussion

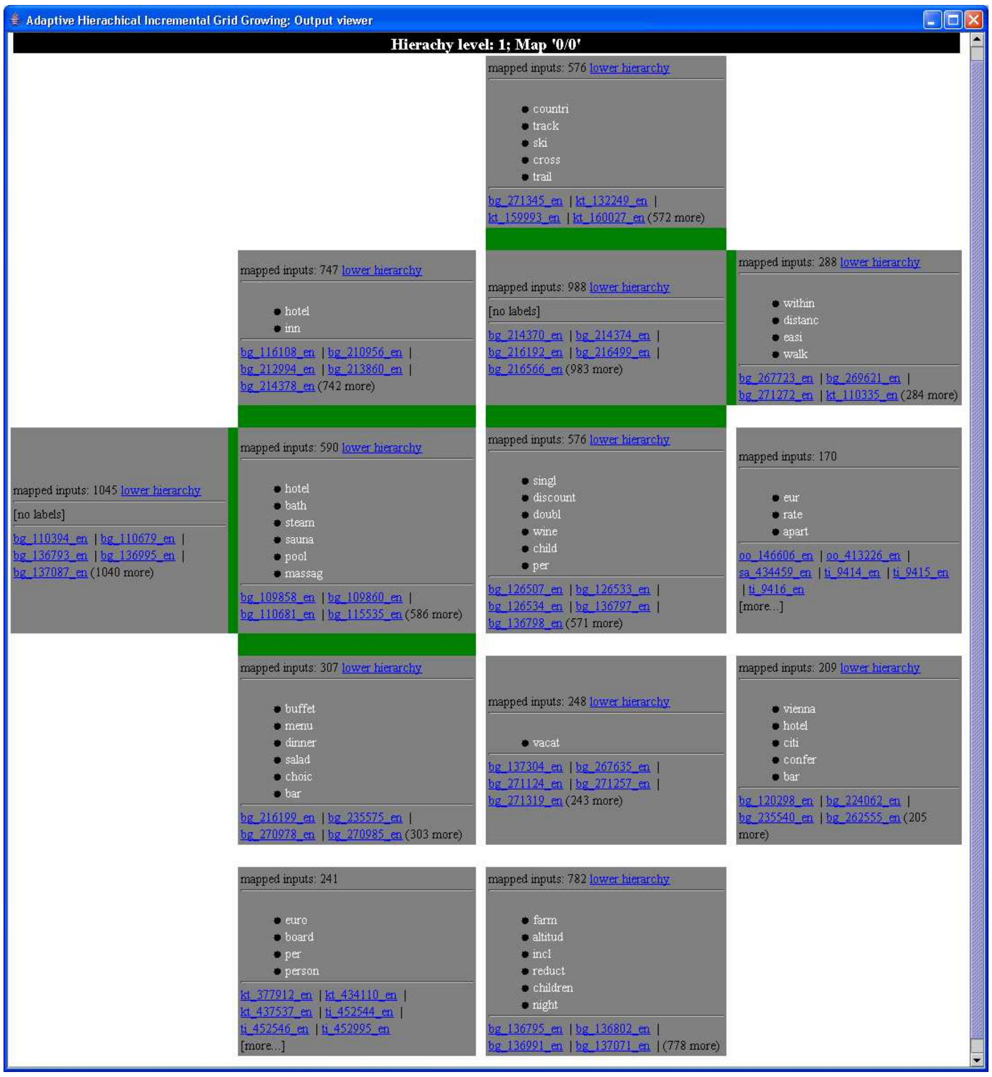
The training algorithm of the GHSOM generates intuitively understandable hierarchical mappings. Especially the hierarchical aspect is vital, as it allows smaller map sizes, which are easier to interpret, and representation at a higher granularity level on a lower layer in the hierarchy. Thus, the user can reach the desired subspace of documents very quickly. However, cluster boundaries, as well as intra-cluster similarities, are visible only through comparing the labels of the nodes. This is not always accurate, and not straight-forward; therefore, a direct cluster visualisation could become useful for the user.

Concerning the algorithm speed, the smaller grid-sizes resulted in a much faster training time. Compared to the Growing Grid described in the previous Section 4.4.2, the run-time for this training process was approximately four times faster.

4.4.4 Results with the Adaptive Hierarchical Incremental Grid Growing

In our concluding experiments, we trained an AHIGG with the hotel data set. A first layer map is presented in Figure 4.17, and similar results as with the GHSOM can be observed.

Again, we can identify (the rather useless) clusters of documents that contain a price information, on nodes 1/4 and 3/2, respectively. Contrary to the experiment with the GHSOM, though, we have additional labels indicating that one cluster more likely contains accommodations that offer *apartments*, while the other cluster includes hotels which offer half or full board. Inspecting documents in these clusters suggested that there are two types of documents: either very short documents with not much more than the price information, or, actually very long and detailed documents, which, however, contained the word "eur" or "euro" up to five or six times, as they stated various prices for different kinds of rooms, supplementary services, or final


Adaptive Hierachical Incremental Grid Growing: Output viewer

Hierachy level: 1; Map '0/0'

mapped inputs: 576 lower hierarchy

- countri
- track
- ski
- cross
- trail

bg_271345_en | kt_132249_en |
kt_159993_en | kt_160027_en (572 more)

mapped inputs: 747 lower hierarchy

- hotel
- inn

bg_116108_en | bg_210956_en |
bg_212994_en | bg_213860_en |
bg_214378_en (742 more)

mapped inputs: 988 lower hierarchy

[no labels]

bg_214370_en | bg_214374_en |
bg_216192_en | bg_216499_en |
bg_216566_en (983 more)

mapped inputs: 288 lower hierarchy

- within
- distanc
- easi
- walk

bg_267723_en | bg_269621_en |
bg_271272_en | kt_110335_en (284 more)

mapped inputs: 1045 lower hierarchy

[no labels]

bg_110394_en | bg_110679_en |
bg_136793_en | bg_136995_en |
bg_137087_en (1040 more)

mapped inputs: 590 lower hierarchy

- hotel
- bath
- steam
- sauna
- pool
- massag

bg_109858_en | bg_109860_en |
bg_110681_en | bg_115535_en (586 more)

mapped inputs: 576 lower hierarchy

- singl
- discount
- doubl
- wine
- child
- per

bg_126507_en | bg_126533_en |
bg_126534_en | bg_136797_en |
bg_136798_en (571 more)

mapped inputs: 170

- eur
- rate
- apart

oo_146606_en | oo_413226_en |
sa_434459_en | ti_9414_en | ti_9415_en
| ti_9416_en
[more...]

mapped inputs: 307 lower hierarchy

- buffet
- menu
- dinner
- salad
- choic
- bar

bg_216199_en | bg_235575_en |
bg_270978_en | bg_270985_en (303 more)

mapped inputs: 248 lower hierarchy

- vacat

bg_137304_en | bg_267635_en |
bg_271124_en | bg_271257_en |
bg_271319_en (243 more)

mapped inputs: 209 lower hierarchy

- vienna
- hotel
- citi
- confer
- bar

bg_120298_en | bg_224062_en |
bg_235540_en | bg_262555_en (205
more)

mapped inputs: 241

- euro
- board
- per
- person

kt_377912_en | kt_434110_en |
kt_437537_en | ti_452544_en |
ti_452546_en | ti_452995_en
[more...]

mapped inputs: 782 lower hierarchy

- farm
- altitud
- incl
- reduct
- children
- night

bg_136795_en | bg_136802_en |
bg_136991_en | bg_137071_en | (778 more)

Figure 4.17: **First layer map of an Adaptive Hierarchical Incremental Grid Growing** trained with the hotel data set.

cleaning. Therefore, these two terms were clearly dominating values in the document vectors. This problem is discussed in detail in Section 4.5.2.

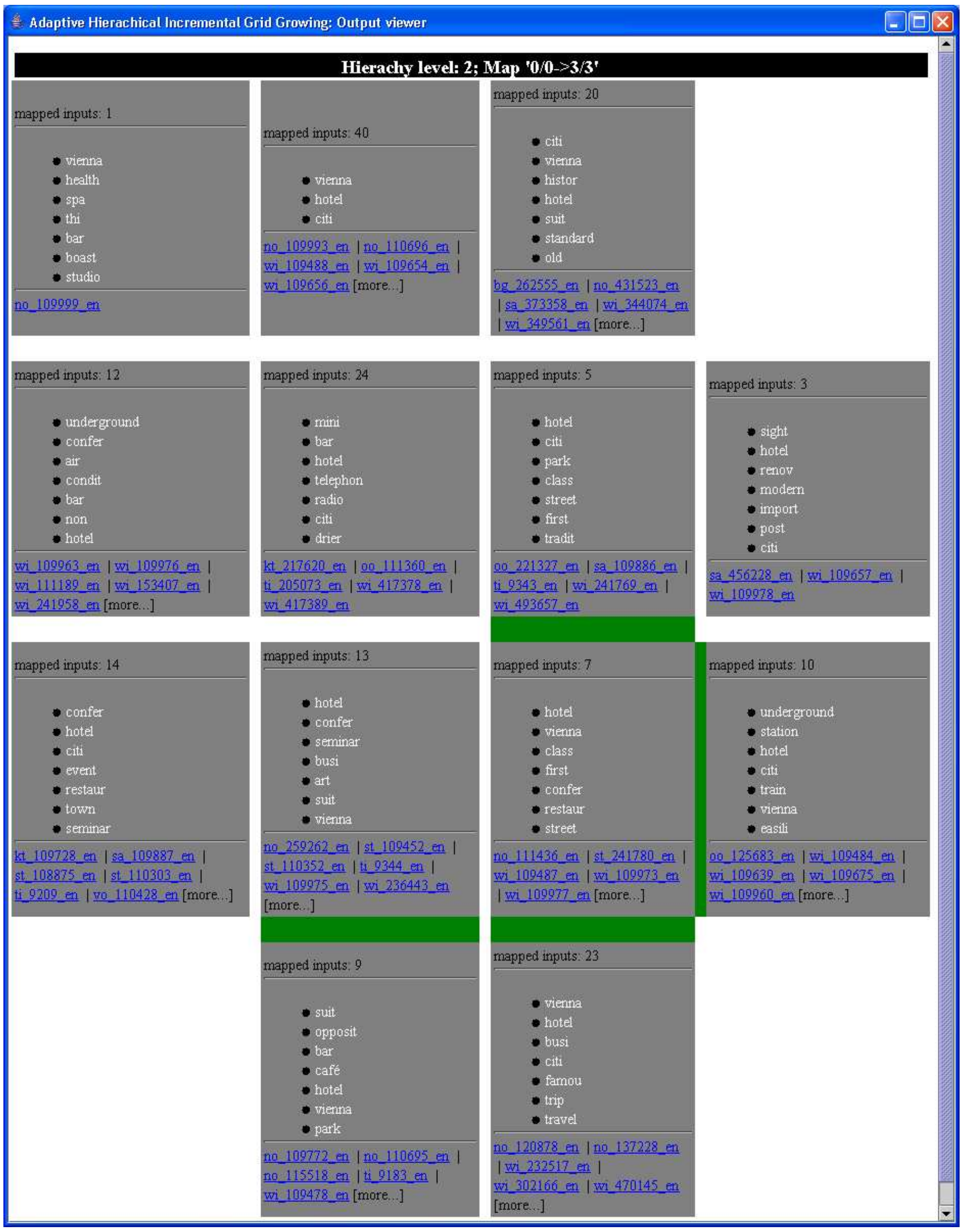
We can further identify other (already well known) clusters, for example, of *farm holidays* (node 2/4), or *conference* and *high standard hotels* on node 3/4. A cluster of documents emphasising various sports activities emerged on node 2/0, while node 1/2 groups accommodations offering *wellness* facilities. On node 0/2, a cluster without any descriptive label has emerged. Inspecting the documents gathered on this node suggested that the documents, though somehow different, all describe hotels which emphasise outdoor activities, nature, and mountains.

We will now describe a couple of second layer maps in detail.

In Figure 4.18, the second layer map representing node 3/4, in a more detailed way, is depicted. Examining the documents on this map, one can observe that a rather high number of hotels is **not** situated within Vienna, although the most descriptive label for this cluster was *Vienna*. This can, however, be easily explained by the fact that most of the hotels not located in Vienna are instead situated in the two provinces close to the city, namely *Burgenland* and *Niederösterreich*, and these documents mention for example the vicinity, or possible excursions, to Vienna. Most of the nodes have Vienna as one of their labels, and can be distinguished by either conference facilities, or labels indicating that they cluster *hotels in the city*.

However, special attention has to be paid to node 0/2, as it raises some problems. On this node, out of the 14 hotels mapped, only one describes a hotel in (or around) Vienna. The other hotels mapped on this node have in common that they are all conference or downtown hotels, but they are spread all over Austria. The same applies, to some extent, to nodes 1/1, 1/2 and 1/3. A user searching for conference or high-standard hotels elsewhere than in Vienna might be interested in these documents. However, the user might not consider this map to be useful, as the parent node seems to indicate that **all** hotels would be in Vienna, as this is the most descriptive label; similar events may arise with other clusters as well. Therefore, the label text should also contain the number of documents containing the specific term. If users get the additional information that only 70% of the hotels are actually in (or around) Vienna, they might also consider this cluster to be important for their search.

In Figure 4.19, a higher granularity level of node 1/2 is depicted. Here, we can observe a rather high number of nodes in the lower part as being interconnected to form one big cluster (nodes 0/2 - 5/2, 0/3 - 4/3, and 1/4). These documents all have in common that they describe hotels offering sauna,



Adaptive Hierachical Incremental Grid Growing: Output viewer

Hierachy level: 2; Map '0/0->3/3'

mapped inputs: 1

- vienna
- health
- spa
- thi
- bar
- boast
- studio

no_109999_en

mapped inputs: 40

- vienna
- hotel
- citi

no_109993_en | no_110696_en | wi_109488_en | wi_109654_en | wi_109656_en [more...]

mapped inputs: 20

- citi
- vienna
- histor
- hotel
- suit
- standard
- old

bg_262555_en | no_431523_en | sa_373358_en | wi_344074_en | wi_349561_en [more...]

mapped inputs: 12

- underground
- confer
- air
- condit
- bar
- non
- hotel

wi_109963_en | wi_109976_en | wi_111189_en | wi_153407_en | wi_241958_en [more...]

mapped inputs: 24

- mini
- bar
- hotel
- telephon
- radio
- citi
- drier

kt_217620_en | oo_111360_en | ti_205073_en | wi_417378_en | wi_417389_en

mapped inputs: 5

- hotel
- citi
- park
- class
- street
- first
- tradit

oo_221327_en | sa_109886_en | ti_9343_en | wi_241769_en | wi_493657_en

mapped inputs: 3

- sight
- hotel
- renov
- modern
- import
- post
- citi

sa_456228_en | wi_109657_en | wi_109978_en

mapped inputs: 14

- confer
- hotel
- citi
- event
- restaur
- town
- seminar

kt_109728_en | sa_109887_en | st_108875_en | st_110303_en | ti_9209_en | vo_110428_en [more...]

mapped inputs: 13

- hotel
- confer
- seminar
- busi
- art
- suit
- vienna

no_259262_en | st_109452_en | st_110352_en | ti_9344_en | wi_109975_en | wi_236443_en [more...]

mapped inputs: 7

- hotel
- vienna
- class
- first
- confer
- restaur
- street

no_111436_en | st_241780_en | wi_109487_en | wi_109973_en | wi_109977_en [more...]

mapped inputs: 10

- underground
- station
- hotel
- citi
- train
- vienna
- easili

oo_125683_en | wi_109484_en | wi_109639_en | wi_109675_en | wi_109960_en [more...]

mapped inputs: 9

- suit
- opposit
- bar
- café
- hotel
- vienna
- park

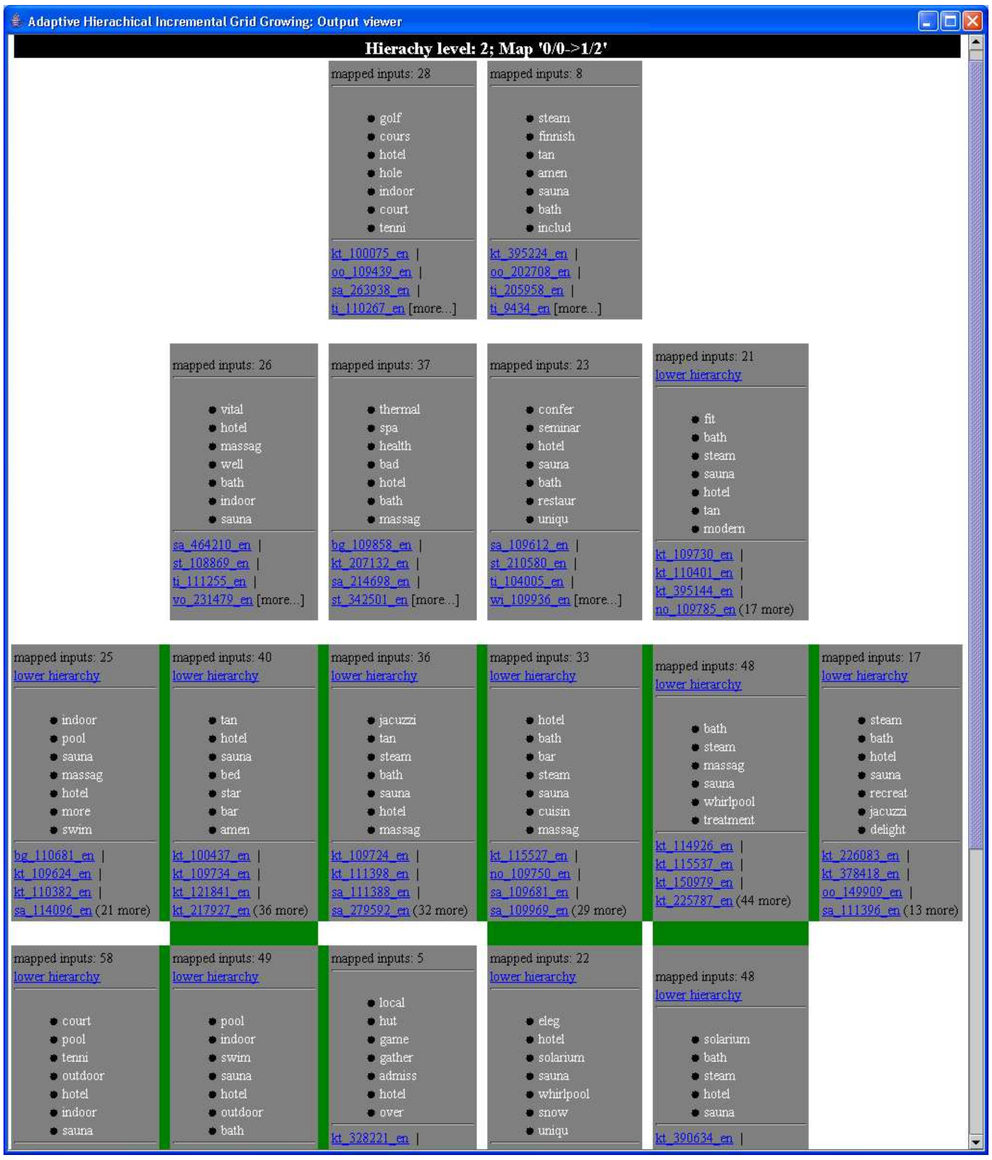
no_109772_en | no_110695_en | no_115518_en | ti_9183_en | wi_109478_en [more...]

mapped inputs: 23

- vienna
- hotel
- busi
- citi
- famou
- trip
- travel

no_120878_en | no_137228_en | wi_232517_en | wi_302166_en | wi_470145_en [more...]

Figure 4.18: **AHIGG of the hotel data set**: layer 2 map of *conference* and *high standard* hotels.


Adaptive Hierachical Incremental Grid Growing: Output viewer

Hierachy level: 2; Map '0/0->1/2'

mapped inputs: 28
- golf
- cours
- hotel
- hole
- indoor
- court
- tenni

kt_100075_en |
oo_109439_en |
sa_263938_en |
ti_110267_en [more...]

mapped inputs: 8
- steam
- finnish
- tan
- amen
- sauna
- bath
- includ

kt_395224_en |
oo_202708_en |
ti_205958_en |
ti_9434_en [more...]

mapped inputs: 26
- vital
- hotel
- massag
- well
- bath
- indoor
- sauna

sa_464210_en |
st_108869_en |
ti_111255_en |
vo_231479_en [more...]

mapped inputs: 37
- thermal
- spa
- health
- bad
- hotel
- bath
- massag

bg_109858_en |
kt_207132_en |
sa_214698_en |
st_342501_en [more...]

mapped inputs: 23
- confer
- seminar
- hotel
- sauna
- bath
- restaur
- uniqu

sa_109612_en |
st_210580_en |
ti_104005_en |
wi_109936_en [more...]

mapped inputs: 21
lower hierarchy
- fit
- bath
- steam
- sauna
- hotel
- tan
- modern

kt_109730_en |
kt_110401_en |
kt_395144_en |
no_109785_en (17 more)

mapped inputs: 25
lower hierarchy
- indoor
- pool
- sauna
- massag
- hotel
- more
- swim

bg_110681_en |
kt_109624_en |
kt_110382_en |
sa_114096_en (21 more)

mapped inputs: 40
lower hierarchy
- tan
- hotel
- sauna
- bed
- star
- bar
- amen

kt_100437_en |
kt_109734_en |
kt_121841_en |
kt_217927_en (36 more)

mapped inputs: 36
lower hierarchy
- jacuzzi
- tan
- steam
- bath
- sauna
- hotel
- massag

kt_109724_en |
kt_111398_en |
sa_111388_en |
sa_279592_en (32 more)

mapped inputs: 33
lower hierarchy
- hotel
- bath
- bar
- steam
- sauna
- cuisin
- massag

kt_115527_en |
no_109750_en |
sa_109681_en |
sa_109969_en (29 more)

mapped inputs: 48
lower hierarchy
- bath
- steam
- massag
- sauna
- whirlpool
- treatment

kt_114926_en |
kt_115537_en |
kt_150979_en |
kt_225787_en (44 more)

mapped inputs: 17
lower hierarchy
- steam
- bath
- hotel
- sauna
- recreat
- jacuzzi
- delight

kt_226083_en |
kt_378418_en |
oo_149909_en |
sa_111396_en (13 more)

mapped inputs: 58
lower hierarchy
- court
- pool
- tenni
- outdoor
- hotel
- indoor
- sauna

mapped inputs: 49
lower hierarchy
- pool
- indoor
- swim
- sauna
- hotel
- outdoor
- bath

mapped inputs: 5
- local
- hut
- game
- gather
- admiss
- hotel
- over

kt_328221_en |

mapped inputs: 22
lower hierarchy
- eleg
- hotel
- solarium
- sauna
- whirlpool
- snow
- uniqu

mapped inputs: 48
lower hierarchy
- solarium
- bath
- steam
- hotel
- sauna

kt_390634_en |

Figure 4.19: **AHIGG of the hotel data set**: layer 2 map of *wellness* hotels.

swimming pools, whirlpools, or solaria, all in-house, but are not necessarily aimed at classifying as *wellness hotels*. By contrast, the clusters on nodes 1/1 and 2/1 are addressing this issue. While the former represents a cluster of hotels that offer services in-house, and additionally have the slogan of *vitality*, the latter consists of hotel descriptions close to a spa.

While nodes 3/0 and 4/1 are more related to the big cluster in the lower part of the map, 2/0 represents hotels that focus more on sports (tennis and golf in this case), and mention spas, saunas and the likes only additionally; 3/1 is a cluster of conference and seminar hotels, also offering saunas and steam baths.

Discussion

The self-organising algorithm of the AHIGG, similarly to the GHSOM, generates intuitively understandable hierarchical mappings. Again, the hierarchical aspect makes it extremely attractive for visualising hierarchically structured data, as in our application: the user can deal with smaller, easier to interpret map sizes, and can browse through representations at different granularity levels. Additionally to the GHSOM, here, cluster boundaries become directly visible. With the Cluster Connections, also inter-cluster similarities and the degree of intra-cluster similarity become directly visible. However, it has to be stated that the GHSOM could be enhanced by the Cluster Connections visualisation technique as well.

Concerning the algorithm speed, no comparisons can be done to the other models, as we relied on a *Java* implementation, while the GHSOM package was implemented in *C++*; software implemented in the Java programming language has in general longer running times. Additionally, different computing environments were used.

However, using the same environments and programming language, similar (but slightly slower) running times as with the GHSOM could be expected. Compared to the GHSOM, the algorithm becomes slower with the fine-tuning phase, and examining the connections after each grid expansion. Also, the grid expansion is in general slower than in the GHSOM, and therefore requires more expansion cycles. This can be explained by the fact that in the AHIGG, at the most three new nodes can be added in each training cycle (as there are at the most three unoccupied, neighbouring grid positions for each node). By inserting complete new rows and columns, the GHSOM might be expanded much faster. However, with the algorithm improvement of expanding around more than one node (as described in Section 4.1.2), faster expansion, and thereby shorter running times, can be achieved. This technique was tested in our experiment, with growing *two* or *three* new nodes at the end of each training cycle; the results showed that the thus generated mappings were highly similar. However, some problems arose from that technique as well; they are described in Section 4.5.1.

4.5 Conclusion

In our experiment, we have successfully shown the usefulness of the Adaptive Hierarchical Incremental Grid Growing for Text Mining purposes. The user can start from a rather small map in the first layer, and then browse through the hierarchies, "zooming" in and out to get a more detailed or a more general representation of the data. With the concept of connections, the AHIGG further provides the user with the possibility to easily detect cluster boundaries; integrating the Cluster Connections representation methods, we further can detect inter and intra-cluster similarities to a better extent. Applying a labelling technique allows the user to easily understand the kind of documents represented by a certain node. Using all these techniques, the AHIGG becomes a powerful tool for analysing high dimensional data.

We showed the usefulness of the AHIGG on two demo data sets. These demo data sets are rather small, however they give the possibility to intuitively verify the algorithm. In the *animals* data set, we could observe the clustering abilities of the algorithm; in the *zoo* data set, we could additionally observe the hierarchically structured mappings generated by the AHIGG. Then, we applied the AHIGG to a real-world example; there as well, we could observe satisfying results.

However, during the experiments with the hotel data set, we could also observe some problems. The quality of the outcome was determined by two separate factors, namely the model used for organising on the one hand, and the data itself on the other hand. Therefore, we will discuss these two components separately now.

4.5.1 Adaptive Hierarchical Incremental Grid Growing

Compared with the standard Self-Organizing Map, the underlying model for the individual maps in the Adaptive Hierarchical Incremental Grid Growing, namely the *Incremental Grid Growing*, has been subject to only a limited amount of research and experiments. For the SOM, in many theoretical and experimental approaches, the functionality and importance of the parameters has been examined; for the IGG, however, they remain relatively unexplored. Some of the uncertainties concerning the AHIGG, for example the effects of parameters or the concept of connectivity, stem from adopting a not too well-explored underlying model.

The highly adaptive structure of the AHIGG, though advantageous for generating adequate representation structures, also imposes some difficulties on the user. Though in contrast to the SOM, the final grid-size has not to be specified in advance, still, the user has to specify a number of other parameters (see Section 3.2). Most importantly, appropriate values for τ_1 and τ_2, and for the *connect* and *disconnect* thresholds have to be found. These parameters, however, will depend to a certain degree on the input data as well; therefore, a number of experiments still has to be carried out to fine-tune the parameters. Inappropriate parameters for τ_1 may lead to too large maps, while a non-fitting τ_2 parameter may lead to too deep hierarchical structures; in both events, or when both effects are combined, the mapping might be of not much use. Some of the improvements presented in Chapter 5, however, might reduce the effect of "bad" parameters.

Problems that may arise because of inappropriate connectivity thresholds are described now.

Hierarchical model

Though the hierarchical structure generated by both the GHSOM and the AHIGG is advantageous for the user in the way that the single maps become smaller and therefore easier to interpret, it also holds one disadvantage: when the user decides to zoom into one branch of the hierarchy, moving to similar maps on the same hierarchical layers is not directly possible. Instead, the user has to zoom out to the previous layer, and then select the sub-maps of similar nodes. It might, therefore, be advantageous to provide hypertext-links to similar maps on the same layer.

Growing around more than one node

The technique of growing new nodes around more than one node, as described in Section 4.4.4, proved to produce satisfying results in many events, and led to a speed-up of the training time.

Sometimes, however, growing around more than one node may lead to higher map sizes than actually needed to achieve the representation granularity specified by the user; especially when the map's $vMQE$ is already very close to the desired value, it might in many events be sufficient to grow

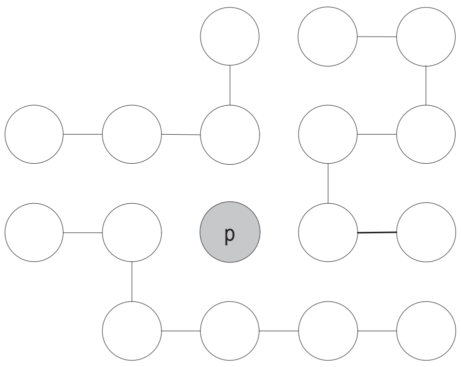


Figure 4.20: Problematic cluster emergence in the Incremental Grid Growing

new nodes around only one new node. Moreover, maps that represent only a small number of input patterns might grow too fast. These problems are discussed, and possible solutions are presented in Section 5.1.

Non-growing clusters

By the architecture and growth algorithm of the Incremental Grid Growing, and with non fitting parameters for the connect and disconnect thresholds, respectively, cases might arise where in fact "large" clusters (i.e., containing many input patterns) become mapped on comparably small areas in the IGG, which cannot grow, as it is located in the centre of the map. A possible scenario is depicted in Figure 4.20, where the node p forms one cluster. The model vector of p actually represents an area with high density in the input vector space, while the neighbouring nodes of p have developed in a totally different way.

This scenario occurred several times during our experiments, when the threshold for deleting a connection was set too low, and therefore the connections from p have been deleted early in the training process. By missing connections to its neighbouring nodes, these cannot be adapted towards the input patterns on p, and therefore, no other node can share the vectors mapped on p. Though small in size in the output space A, the cluster represented by the node p may actually contain a large number of input patterns; this may lead to a relatively high mqe of the node. As the node is situated in the centre of the map, it cannot grow new nodes; therefore, the mqe of the node, and possibly also the $vMQE$ of the map, cannot be reduced to the desired level, and the training process might either end in an endless loop, or when using the *improvement degree* described in Section 3.2.2, be aborted without reaching the desired representation granularity. Therefore, special attention has to be laid on selecting fitting connection thresholds. Alternatively, at least for not extreme scenarios, this problem could be solved by implementing the suggestion described in Section 5.7.

Note that scenarios like this cannot happen in the *Growing Hierarchical Self-Organizing Map*, due to the model it builds on: the *Growing Grid* does, on the one hand, not have the concept of connections, allowing the neighbouring nodes of p develop model vectors similar to p, and therefore sharing the input patterns mapped on p; on the other hand, growth in the Growing Grid is possible also in the centre of the map, therefore also next to the node p.

4.5.2 Hotel data

As for the data, it has to be mentioned that even after cleaning the data set from small documents, the discrepancies between longer and shorter documents seem to be too extreme. A good example are the clusters that emerged around the terms "sea level" and "eur"/ "euro". These contain documents that include a price or information about the altitude of the accommodation, but not many other details. Forming a cluster around hotels in the same price category, or with hotels that are in the altitude would be meaningful, however, here the cluster is formed from the mere existence of these keywords. Using a further reduced data set, taking only much longer documents, will, however, only partly solve the problem. This is due to the fact that there are also longer documents mapped on these nodes. These are documents with a high frequency of these terms, for example because of giving several prices for different services. To get a better vector representation, and therefore better clustering results, one should therefore consider, in applications like this, additionally using a stop-word list for reducing the list of terms, for example by the ones mentioned. Additionally, other typical stop-words like "you", "not", etcetera, remained in the list of terms due to the fact that many documents were not using "normal" free text, but contained, more or less, some keywords only. Therefore, the document frequency of these terms, easily recognised as stop-words in other text collections, was not high enough to be filtered out.

Moreover, some documents contained spelling mistakes; it would be desirable to apply some spelling correction technique before generating the vector representations.

Another problem stems from the underlying model of the document representation. In the vector model we used, independence between the terms is assumed [BYRN99]. However, often, combinations of terms might be interesting. For example, the terms "swimming" and "pool" may together form one term, when the hotel offers a swimming pool, or might be independent, when there is, for example, just "swimming" mentioned in the document, indicating that there is any possibility for swimming close by. However, both hotels might be organised into the same cluster, and "swimming" might be a label for this cluster. Then a user expecting a swimming pool might find many irrelevant documents. Another example are the terms "underground" and "underground parking": documents indicating an underground public transport stop nearby might get mixed up with others offering parking facilities.

To solve this problem, one could use phrases rather than only words as the index terms, or, an approach as in the WEBSOM project could be utilised (see Section 2.1.3).

Besides the afore said, the hotel data contains actually two dimensions, namely the category of the hotel and its equipment and possible activities, and a geographic dimension. However, these dimensions are not covered to the same extent in all documents, leading to a mixture of geographical and categorial clustering. Additionally, some of the documents contain parts of their structured data in the free-text descriptions, like the price information, or the altitude. Therefore, some rather non useful groupings around terms like "euro" or "altitude" respectively "sea level" emerged.

For the two dimensions, one idea might be to provide the user with two different ways of browsing through the data according to the two different dimensions.

Chapter 5

Future work

In this chapter, some possible future extensions and improvements of the original Adaptive Hierarchical Incremental Grid Growing algorithm, as described in Section 3.2, are discussed. Not all of these suggestions might actually improve the usefulness of the AHIGG, however, we consider them worthy of additional research.

5.1 Growth

As described in Section 4.1.2, we used a slightly different approach for the growth of the IGG-maps, with growing more than one new node at the same time. However, in our opinion this approach needs to be refined to some extent. When it is decided to grow the grid according to Equation 3.4, there will be at least one node having a high error value, i.e. an error value $mqe_i > \tau_1 \times qMQE_{parent}$, as otherwise, Equation 3.4 could not apply. But, it is not necessarily guaranteed that there will be **more** than this one node having an error value that would justify its growth. This could be explained by one node having many dissimilar input patterns mapped onto itself, while the other nodes have either very few, or very similar vectors mapped onto themselves. A possible scenario is illustrated in Figure 5.1: the node marked in black has a high *mqe*, and therefore, new nodes should be grown around it. The user had specified to grow around three nodes at the same time, and therefore, also the gray-shaded nodes will grow; however, these nodes have a rather small *mqe*, and growth in these areas might not be necessary and target-oriented.

Therefore, we propose to guide the growth of the nodes beyond the first one by a similar approach as in the Growing Hierarchical Self-Organizing Map model (as described in Section 2.4), with a threshold value (comparable

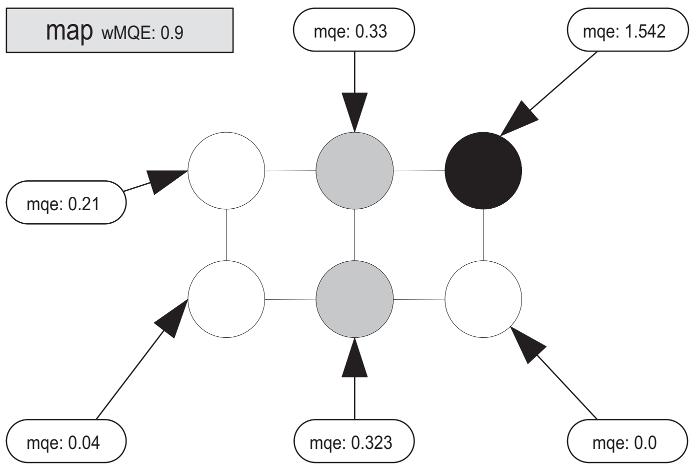


Figure 5.1: **Problematic growth around multiple nodes**: growing where it is not needed.

to the *growth threshold* g) as a comparison to the node's *mqe*. Only if the node's *mqe* is higher than this threshold is it decided to grow this node. A simple solution for the threshold, not requiring any additional parameters to be specified by the user, could be the map's $vMQE$ itself, or a multiple of it. This approach has, however, not been taken into account in our implementation.

Still, with the above mentioned technique, problems as described in Section 4.4.4, i.e. the map grows bigger than actually needed to reach the desired granularity level, might arise. Especially in events when the $vMQE$ of the map is already close to the specified target value, it may be sufficient to grow only around one node to reduce the $vMQE$ enough to satisfy the growth control condition as defined in Equation 3.4. But if the user has defined to grow around two or more nodes with each training cycle, we may end up with a larger map than desired. A possible solution for this problem would be to reduce the growth to expand only around one node when the $vMQE$ lies within a certain environment of the desired value.

Another problem stemming from growing more than one new node in each training cycle may arise for maps that represent only a small number of input patterns. The number of possible new nodes is relatively high compared to the number of existing nodes in the first few training cycles. For example, if the user specified to grow two nodes simultaneously in one training cycle, and the initial grid size was 2×2, then, when the condition for horizontal

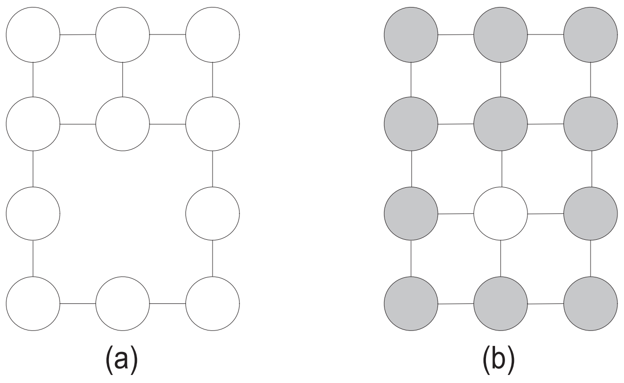


Figure 5.2: **Deleting nodes in the AHIGG**: Structures as in (a) might not be achievable, rather the case as in (b), with no vectors mapped on the central node, will occur.

growth is fulfilled, in the first training cycle, the number of new nodes would be four; thus, the number of nodes in the grid would *double*. When the user specified to grow around three or four nodes, then the number of nodes would even *triple* or increase *four times*, respectively. For small input data sets, or for maps on lower hierarchical layers, which contain only a subset of the input data, this rapid growth may lead to a far too large map. One solution to prevent events like this would be to limit the number of new nodes to a certain percentage of the existing number. For example, we could say that we expand the map only by 50 percent at the most in each training cycle. This mechanism will reduce the number of new nodes grown in the first few cycles, while in later cycles, when the relative increase in nodes is not so dramatically anymore, it will not be effective anymore.

5.2 Deleting Nodes

The Adaptive Hierarchical Incremental Grid Growing, using the Incremental Grid Growing for its maps on the individual layers, can develop very flexible maps, thereby being able to map arbitrary input data sets. However, some network topologies, as for example the one depicted in Figure 5.2 (a), might not always be achievable with the current algorithm. Depending on the ordering of input patterns, the topology might be achieved; likely however, a structure as in (b) will be developed, with no vectors mapped on the node marked white.

Moreover, when adding new maps on lower layers, they might be too large in initial size for their subset of input patterns. For both events, we suggest considering a mechanism for deleting nodes from the grid.

It is, however, difficult to decide on when to delete a node. [Bay95] states that deleting a node when the area of the input space it represents has zero density, is problematic, as it declares the input data set as *training set*. Other existing solutions are to delete nodes that are not adapted during the training process, or to delete nodes whose function could be fulfilled by other neurons without loss. The former approach is problematic, as there might be nodes that were not adapted, but nonetheless contain input patterns - this is the case when the model vector of the node exactly represented the mapped input patterns. For the latter approach, [Bay95] states that it is intuitively the best one, but the computational effort is demanding.

However, deleting nodes that have no vectors mapped on them, in other words, representing an input space with zero density, seems to be a satisfying approach. Alternatively, one could examine whether the model vector of the node has a position in the input space with a very low probability density; this approach is utilised for example in the Growing Cell Structures (see Section 2.7.1.

In our opinion, deleting nodes makes sense in the fine-tuning phase only; this can be explained by a scenario where we might delete the same amount of nodes we just added during the grid expansion. Then, the map may not grow in size, and the training process will not lead to minimising the quantisation errors, or at least will become slower. Therefore, if we consider only the completely trained map, we will avoid cases like this.

However, similarly to the arguments presented above, when introducing the concept of deleting nodes, one should also think about the application the map will be used for. There might be cases where the map is trained by one sub-set of the data, and then the thus generated structure is used to visualise different data sets. For example, in our application, one could train a map with the data-subset of hotels in *Salzburg*, and then visualise the input patters from the subset of *Niederösterreich*. When an application like this is desired, deleting nodes that represent a region of the input space with zero density in *one* data-subset might easily represent a much denser region in another subset of the data.

In general, however, applications like this might not be very common, and therefore, deleting nodes to represent the input data more accurately might make sense in other applications. One solution to combine both aspects might be to only "virtually" delete nodes, i.e. we just do not show nodes with no input data mapped onto in the visualisation.

5.3 Improvement Degree

In the AHIGG-Algorithm (see Section 3.2), a so-called *improvement degree* was introduced as a way to stop the growth process in a stagnation phase. However, this improvement degree should not only take the last training cycle into account, but a longer period. This can be explained by (possible) scenarios where we have only a slight representation improvement, or maybe none at all, in one cycle, but in subsequent cycles, the quality of the mapping would still increase reasonably. A scenario like this may be due to facts like a low number of iterations. To avoid them, we therefore, propose to calculate the improvement degree over a longer period of, say, two or three cycles.

5.4 Feature Vector Reduction

In the *Hierarchical Feature Map*, the concept of reducing the input vectors by omitting some components on lower hierarchical layers, was introduced. The reason for this was that with clearly hierarchical data (as it was especially the case with the data used for demonstration in [Mii90]), on lower layers, where the maps represent clusters and subsets in the data, all the mapped vectors will have a number of features in common. Therefore, they are not needed for the organisation process anymore, as you cannot distinguish input patterns by their *common* features. This concept resulted in a speed-up of the organisation process.

Furthermore, it would also improve the explanatory power of the *labels* assigned to each node. If we consider that we have a certain number of components with the same value for *all* the documents, the quantisation error for these components would be zero, and therefore these components have a high likelihood of being selected as labels. However, labelling all the nodes on one map with the same label does not reveal much information to distinguish between the inputs mapped onto it. An approach would be to assign features that are common for all the inputs to the map, and the node-labels would be generated from the remaining components. To give an intuitive example, consider all the animals from the species *bird* in the zoo data-set from Chapter 4 (see Tables 4.3 and B.1, respectively). All of them have the same weights for the features *toothed*, *backbone*, *breathes*, *venomous*, *fins*, *hair*, *feathers*, *eggs*, and *milk*, while differences are only found on the features *airborne*, *aquatic*, and *predator*. If we have a map with birds only, and with the goal to immediately see the clusters, it would be advantageous to have only the distinctive features as labels.

However, when it comes to implementing this strategy, some difficulties might arise. In the two examples stated above, i.e. the zoo data set and the *script* data-set in [Mii90], we have **discrete** values for the component weights (with the most simple case, namely binary values, in the zoo data-set), whereas in an example like the tourism data, as presented in Section 4.4, we have continuous values. Then, it might be rarely the case that we have *exactly* the same values for all the nodes on one map. It seems rather necessary to define a similarity measure, maybe an ϵ environment in which all component weights must lie in. However, this would mean introducing another parameter. Care must be taken not choose this environment too wide, to avoid losing components which could actually well distinguish between input data patterns.

5.5 Algorithm Speed-up

Through its hierarchical structure, where the individual maps are smaller in size and have a lower number of input patterns, the AHIGG already promises some speed-ups compared to the SOM. However, for making the model more usable, additional speed-ups are desirable. Of the improvements described in Section 2.1.6, however, not all seem to be applicable.

Rapid construction of large maps doesn't seem to be applicable, as, on the one hand, the AHIGG will not (at least for most applications) have big maps, the interpolation of an irregular grid, like the IGG is, doesn't seem to be so easy, and most importantly, we have no a-priori information about the final grid-size. However, the approach presented in Section 5.1 is aimed at accelerating the growth process.

However, the suggestions for *rapid fine-tuning*, like the techniques of addressing the old winners, or the batch map principle, may be applicable to the AHIGG. It remains unclear, though, how the concept of *connections* used in the AHIGG will comply with these algorithm changes; for addressing the old winners for example, the question whether to search only in the *connected* neighbourhood, or to perform the search regardless of the map's connectivity, has to be answered. Furthermore, it has to be shown in experiments how much speed improvement could actually be gained out of these methods, which were originally designed for *large* maps, while the map sizes in the AHIGG remain rather small.

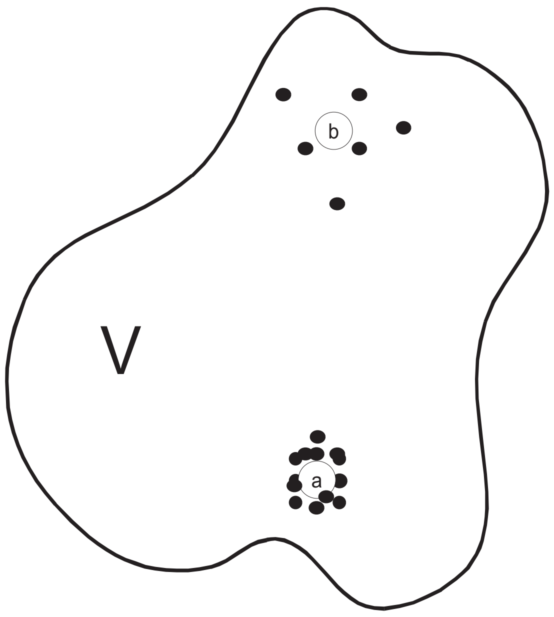


Figure 5.3: **Mean quantisation error vs. quantisation error**: a different criterion for hierarchical growth.

5.6 Growth Control Measures

In the AHIGG, a *mean quantisation error* is used as a measure for expanding a single IGG map, as well as for growing hierarchically, while in the GHSOM, the *quantisation error* is used. The difference between these two measures can most easily be described by the scenario illustrated in Figure 5.3: A small number of input vectors, which are relatively far away, has been mapped onto node *b*, while onto node *a*, a large number of very similar input vectors have been mapped. For both nodes, the *quantisation error*, i.e. the sum of all the distances between the model vector and the mapped vectors, will be approximately the same; however, the *mean quantisation error*, i.e. the *average* distance between the model vector and the mapped input vectors, will be much smaller for node *a*. The consequence is that in cases where the quantisation error is taken as a measure for the quality of a nodes representation, *both a* and *b* will need to represent their mapped vectors more adequately, and therefore will grow a new hierarchical layer. By contrast, if the *mean quantisation error* is used as the quality measure, *only* node *b* will be represented on the next hierarchical layer.

An overview over various quantisation error computations was given in Table 3.1. It can be stated that the mean quantisation error is a more statistical measure, while the quantisation error more closely follows the concept of the SOM, where more output space is provided for areas with a higher density in the input space [RMD02]. To give a qualitative evaluation about the different measures, however, is difficult and would have to be tested through experiments.

5.7 Connectivity

In the model of the AHIGG, as well as in the IGG, there are only two different statuses for connections between nodes - either they are connected, or they are not. This approach, however, might be too simplistic: as well as ignoring inter-cluster similarities, the degree of similarity between nodes within a cluster also remains unconsidered. An approach to provide better *visualisation* of these inter- and intra-cluster similarities was adopted in our implementation (refer to Section 4.1.2). However, it might be worthwhile considering these similarities during the training process as well: the adaptation of neighbouring nodes could, besides the distance, also be based on the similarity. So far, in the AHIGG and IGG models, the connectivity status determines whether to adapt a node to its full extent (as computed by neighbourhood function and learning rate), or not at all, when there is no connection between the nodes. We suggest changing this adaptation scheme to adapt to the full extent only for highly similar nodes, and for nodes which are located in different clusters, but still show some similarity to the winning node, at least perform a small adaptation as well. One could either utilise discrete statuses, like those introduced in the Cluster Connections visualisation in Section 4.1.2, where nodes would be adapted in discrete steps, depending on whether they are very similar, similar, or partly similar. Alternatively, a continuous function of the similarity degree could be utilised. In the event of using a discrete function, introducing new user-defined parameters should be avoided; rather, an approach like in Section 4.1.2, where we computed the thresholds out of the already existing *connect* and *disconnect* thresholds, should be adopted.

5.8 Automatic Parameter Fine-tuning

As in any unsupervised neural network model, automatic parameter fine-tuning would be a desirable feature. However, this is not easily achieved without user-feedback (and the AHIGG would not be an unsupervised model anymore). One approach could be, when the user-defined parameters form extremely flat, or extremely deep hierarchies, to adjust the parameters for a better balance between the sizes of the individual maps, and the hierarchical depth. However, any approach like this should not abandon the mapping generated with the initial parameters, but rather provide an additional mapping with adjusted parameters to the user.

Chapter 6

Conclusion

In this thesis, we have given an overview of unsupervised neural network models which can be utilised for text mining purposes, and tested some of them for their usefulness. The models we investigated were based on the *Self-Organizing Map* (SOM) [Koh82]; the self organising training process generates a mapping from a high-dimensional input space to a lower dimensional space; this output space is in many models a two-dimensional, rectangular grid of nodes. These nodes, by their so-called *model vectors*, represent a certain area of the input space. Input patterns which are similar will be mapped spatially closely to the nodes in the output space. The SOM is a powerful tool for data visualisation, as well as for exploratory data analysis. However, the model of the Self-Organizing Map has some weaknesses: the size of the output grid has to be pre-defined, which in general will require an a-priori knowledge of the input space, and the SOM does not allow hierarchical structures to be uncovered. Therefore, several different models, extending the original SOM algorithm, have been proposed.

Some models addressing the problem of the need of an a-priori knowledge for specifying the network size have been presented in detail in Chapter 2. The *Growing Grid* (GG), introduced in [Fri95a], introduces a dynamically growing grid, where, until a stopping criterion is fulfilled, rows or columns are inserted around the one node that represents its corresponding input space most inadequately. Thus, the Growing Grid is capable of dynamically determining the grid size. The *Incremental Grid Growing* (IGG) [BM93], is another model growing dynamically in size; new nodes are added on the perimeter of the grid, until a certain stopping criterion is fulfilled. However, here the grid is no longer strictly rectangular, as there might be sparse grid positions, i.e. grid positions not occupied by any node. Additionally, the nodes are not fully interconnected; by contrast, connections between nodes might be deleted when the nodes' model vectors become too dissimilar during the training process. Thereby, direct cluster visualisation becomes possible. The *Growing Self-Organizing Map* (GSOM) [AHS00] is a model similar to the IGG, also growing nodes at the perimeter. A difference can be found in how the growth is initiated, and in the stopping criterion, which is abstracted from the input data by a user-defined factor determining the desired granularity level. Additionally, [AHS00] discusses an approach how to (manually) create an hierarchical structuring of the data.

Then, we have described models that automatically generate a hierarchically structured mapping. Hierarchies are desired, as on the one hand they are inherited in many document collections, and therefore a proper representation of these collections should also reflect the hierarchical structure. On the other hand, for *large* document collections, the SOM requires an adequately large grid to represent the data properly. This will result in increased computational effort; additionally, a large map does not support explorative search in a reasonable way. In an hierarchical model, though, a number of smaller maps, arranged on different hierarchical layers, is sufficient. Maps on lower hierarchical layers will represent the input data mapped on their *parent nodes* in the higher layer in greater detail; the training of these maps is performed only with the subsets of input patterns mapped on this parent node. The user thus can "zoom" in to and out of different granularity levels of the mapping.

From among the existing models, we described the *Hierarchical Feature Map* (HFM) [Mii90]; there, several standard SOMs are arranged in multiple layers. The hierarchy is, however, fully balanced - real-world data will not necessarily fit into this structure. Moreover, the size of the individual maps, and the number of hierarchical layers, have to be defined in advance, and therefore will require an a-priori knowledge of the input data space. This is where the *Growing Hierarchical Self-Organizing Map* (GHSOM) [RMD02] comes in: the individual maps are dynamically Growing Grids, and the hierarchical layers are created also dynamically during runtime, depending on the input data. With these features, the GHSOM can adequately represent unevenly distributed and unevenly hierarchically structured data.

Additionally, we presented two models that are not two-dimensional, or not based on a rectangular grid. The *Growing Cell Structures* [Fri92] and the *Neural Gas* [MS91], have been described, as they suggest interesting features like deleting nodes, and a different approach for the connectivity, respectively.

Finally, the *Adaptive Hierarchical Incremental Grid Growing* (AHIGG) [He01] has been described in Chapter 3, as a model combining features of the GHSOM with the IGG, by replacing the individual GG with IGG maps. Therefore, clusters become visible as well, and the idea of a completely filled rectangular grid is abandoned. We implemented the AHIGG, with slight improvements, and tested the model on two demo data sets, as well as an application from the real-world, using a collection of free-text hotel descriptions. There, the usefulness of the AHIGG has been shown, and advantages over the SOM and GHSOM have been elaborated. However, the AHIGG, as well as its basis, the IGG, still remains rather unexplored, and requires further research.

With the suggestions for future work in Chapter 5, the AHIGG could be significantly improved, and its value for text mining or other data analysis applications could increase significantly.

Appendix A

Glossary of terms

AHIGG	Adaptive Hierarchical Incremental Grid Growing, see Section 3
GCS	Growing Cell Structures, see Section 2.7.1
GG	Growing Grid, see Section 2.2
GHSOM	Growing Hierarchical Self-Organizing Map, see Section 2.6
GSOM	Growing Self-Organizing Map, see Section 2.4
HFM	Hierarchical Feature Map, see Section 2.5
IGG	Incremental Grid Growing, see Section 2.3
Input pattern	An item of the input data set. Sometimes called *input vector.*
Iteration	One iteration of a SOM training algorithm
Learning rate	Decides about how much the model vectors will be adapted towards the input pattern.
Model vector	A vector assigned to a node; sometimes referred to as *weight vector* or *reference vector.*
Neighbourhood adaptation	Decides which nodes will be adapted besides the winner, and to what extent.
NG	Neural Gas, see Section 2.7.2
Node	A processing unit in a neural network. Sometimes referred to as *unit* or *cell.*
SOM	Self-Organizing Map, see Section 2.1
Winner	The node who's model vector is the closest to a presented input pattern. Sometimes referred to as *best matching* node.

Appendix B

Zoo data set

Table B.1: The *zoo* demo data set

Name	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	breathes	venomous	fins	legs					tail	domestic	catsize	species
													2	4	5	6	8				
aardvark	1	1	0	1	0	0	1	1	1	1	0	0	0	1	0	0	0	0	0	1	mammal
antelope	1	0	0	1	0	0	0	1	1	1	0	0	0	1	0	0	0	1	0	1	mammal
bass	0	0	1	1	0	1	1	1	1	0	0	1	0	0	0	0	0	1	0	0	fish
bear	1	0	0	1	0	0	1	1	1	1	0	0	0	1	0	0	0	0	0	1	mammal
boar	1	0	0	1	0	0	1	1	1	1	0	0	0	1	0	0	0	1	0	1	mammal
buffalo	1	0	0	1	0	0	0	1	1	1	0	0	0	1	0	0	0	1	0	1	mammal
calf	1	0	0	1	0	0	0	1	1	1	0	0	0	1	0	0	0	1	1	1	mammal
carp	0	0	1	0	0	1	0	1	1	0	0	1	0	0	0	0	0	1	1	0	fish
catfish	0	0	1	0	0	1	1	1	1	0	0	1	0	0	0	0	0	1	0	0	fish
cavy	1	0	0	1	0	0	0	1	1	1	0	0	0	1	0	0	0	0	1	0	mammal
cheetah	1	0	0	1	0	0	1	1	1	1	0	0	0	1	0	0	0	1	0	1	mammal
chicken	0	1	1	0	1	0	0	0	1	1	0	0	1	0	0	0	0	1	1	0	bird
chub	0	0	1	0	0	1	1	1	1	0	0	1	0	0	0	0	0	1	0	0	fish
clam	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	invertebrate
crab	0	0	1	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	invertebrate
crayfish	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	invertebrate
crow	0	1	1	0	1	0	1	0	1	1	0	0	1	0	0	0	0	1	0	0	bird
deer	1	0	0	1	0	0	0	1	1	1	0	0	0	1	0	0	0	1	0	1	mammal
dogfish	0	0	1	0	0	1	1	1	1	0	0	1	0	0	0	0	0	1	0	1	fish
dolphin	0	0	0	1	0	1	1	1	1	1	0	1	0	0	0	0	0	1	0	1	mammal
dove	0	1	1	0	1	0	0	0	1	1	0	0	1	0	0	0	0	1	1	0	bird
duck	0	1	1	0	1	1	0	0	1	1	0	0	1	0	0	0	0	1	0	0	bird

continued on next page

Table B.1: The *zoo* demo data set *(continued from previous page)*

Name	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	breathes	venomous	fins	legs 2	4	5	6	8	tail	domestic	catsize	species
elephant	1	0	0	1	0	0	0	1	1	1	0	0	0	1	0	0	0	1	0	1	mammal
flamingo	0	1	1	0	1	0	0	0	1	1	0	0	1	0	0	0	0	1	0	1	bird
flea	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	insect
frog	0	0	1	0	0	1	1	1	1	1	0	0	0	1	0	0	0	0	0	0	amphibian
venomous frog	0	0	1	0	0	1	1	1	1	1	1	0	0	1	0	0	0	0	0	0	amphibian
fruitbat	1	0	0	1	1	0	0	1	1	1	0	0	1	0	0	0	0	1	0	0	mammal
giraffe	1	0	0	1	0	0	0	1	1	1	0	0	0	1	0	0	0	1	0	1	mammal
gnat	0	0	1	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	insect
goat	1	0	0	1	0	0	0	1	1	1	0	0	0	1	0	0	0	1	1	1	mammal
gorilla	1	0	0	1	0	0	0	1	1	1	0	0	1	0	0	0	0	0	0	1	mammal
gull	0	1	1	0	1	1	1	0	1	1	0	0	1	0	0	0	0	1	0	0	bird
haddock	0	0	1	0	0	1	0	1	1	0	0	1	0	0	0	0	0	1	0	0	fish
hamster	1	0	0	1	0	0	0	1	1	1	0	0	0	1	0	0	0	1	1	0	mammal
hare	1	0	0	1	0	0	0	1	1	1	0	0	0	1	0	0	0	1	0	0	mammal
hawk	0	1	1	0	1	0	1	0	1	1	0	0	1	0	0	0	0	1	0	0	bird
herring	0	0	1	0	0	1	1	1	1	0	0	1	0	0	0	0	0	1	0	0	fish
honeybee	1	0	1	0	1	0	0	0	0	1	1	0	0	0	0	1	0	0	1	0	insect
housefly	1	0	1	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	insect
kiwi	0	1	1	0	0	0	1	0	1	1	0	0	1	0	0	0	0	1	0	0	bird
ladybird	0	0	1	0	1	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0	insect
lark	0	1	1	0	1	0	0	0	1	1	0	0	1	0	0	0	0	1	0	0	bird
leopard	1	0	0	1	0	0	1	1	1	1	0	0	0	1	0	0	0	1	0	1	mammal
lion	1	0	0	1	0	0	1	1	1	1	0	0	0	1	0	0	0	1	0	1	mammal

continued on next page

Table B.1: The *zoo* demo data set *(continued from previous page)*

Name	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	breathes	venomous	fins	legs 2	legs 4	legs 5	legs 6	legs 8	tail	domestic	catsize	species
lobster	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	invertebrate
lynx	1	0	0	1	0	0	1	1	1	1	0	0	0	1	0	0	0	1	0	1	mammal
mink	1	0	0	1	0	1	1	1	1	1	0	0	0	1	0	0	0	1	0	1	mammal
mole	1	0	0	1	0	0	1	1	1	1	0	0	0	1	0	0	0	1	0	0	mammal
mongoose	1	0	0	1	0	0	1	1	1	1	0	0	0	1	0	0	0	1	0	1	mammal
moth	1	0	1	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	insect
newt	0	0	1	0	0	1	1	1	1	1	0	0	0	1	0	0	0	1	0	0	amphibian
octopus	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	1	invertebrate
opossum	1	0	0	1	0	0	1	1	1	1	0	0	0	1	0	0	0	1	0	0	mammal
oryx	1	0	0	1	0	0	0	1	1	1	0	0	0	1	0	0	0	1	0	1	mammal
ostrich	0	1	1	0	0	0	0	0	1	1	0	0	1	0	0	0	0	1	0	1	bird
parakeet	0	1	1	0	1	0	0	0	1	1	0	0	1	0	0	0	0	1	1	0	bird
penguin	0	1	1	0	0	1	1	0	1	1	0	0	1	0	0	0	0	1	0	1	bird
pheasant	0	1	1	0	1	0	0	0	1	1	0	0	1	0	0	0	0	1	0	0	bird
pike	0	0	1	0	0	1	1	1	1	0	0	1	0	0	0	0	0	1	0	1	fish
piranha	0	0	1	0	0	1	1	1	1	0	0	1	0	0	0	0	0	1	0	0	fish
pitviper	0	0	1	0	0	0	1	1	1	1	1	0	0	0	0	0	0	1	0	0	reptile
platypus	1	0	1	1	0	1	1	0	1	1	0	0	0	1	0	0	0	1	0	1	mammal
polecat	1	0	0	1	0	0	1	1	1	1	0	0	0	1	0	0	0	1	0	1	mammal
pony	1	0	0	1	0	0	0	1	1	1	0	0	0	1	0	0	0	1	1	1	mammal
porpoise	0	0	0	1	0	1	1	1	1	1	0	1	0	0	0	0	0	1	0	1	mammal
puma	1	0	0	1	0	0	1	1	1	1	0	0	0	1	0	0	0	1	0	1	mammal
pussycat	1	0	0	1	0	0	1	1	1	1	0	0	0	1	0	0	0	1	1	1	mammal

continued on next page

Table B.1: The *zoo* demo data set *(continued from previous page)*

Name	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	breathes	venomous	fins	legs					tail	domestic	catsize	species
													2	4	5	6	8				
raccoon	1	0	0	1	0	0	1	1	1	1	0	0	0	1	0	0	0	1	0	1	mammal
reindeer	1	0	0	1	0	0	0	1	1	1	0	0	0	1	0	0	0	1	1	1	mammal
rhea	0	1	1	0	0	0	1	0	1	1	0	0	1	0	0	0	0	1	0	1	bird
scorpion	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	1	1	0	0	invertebrate
seahorse	0	0	1	0	0	1	0	1	1	0	0	1	0	0	0	0	0	1	0	0	fish
seal	1	0	0	1	0	1	1	1	1	1	0	1	0	0	0	0	0	0	0	1	mammal
sealion	1	0	0	1	0	1	1	1	1	1	0	1	1	0	0	0	0	1	0	1	mammal
seasnake	0	0	0	0	0	1	1	1	1	0	1	0	0	0	0	0	0	1	0	0	reptile
seawasp	0	0	1	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	invertebrate
skimmer	0	1	1	0	1	1	1	0	1	1	0	0	1	0	0	0	0	1	0	0	bird
skua	0	1	1	0	1	1	1	0	1	1	0	0	1	0	0	0	0	1	0	0	bird
slowworm	0	0	1	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1	0	0	reptile
slug	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	invertebrate
sole	0	0	1	0	0	1	0	1	1	0	0	1	0	0	0	0	0	1	0	0	fish
sparrow	0	1	1	0	1	0	0	0	1	1	0	0	1	0	0	0	0	1	0	0	bird
squirrel	1	0	0	1	0	0	0	1	1	1	0	0	1	0	0	0	0	1	0	0	mammal
starfish	0	0	1	0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	invertebrate
stingray	0	0	1	0	0	1	1	1	1	0	1	1	0	0	0	0	0	1	0	1	fish
swan	0	1	1	0	1	1	0	0	1	1	0	0	1	0	0	0	0	1	0	1	bird
termite	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	insect
toad	0	0	1	0	0	1	0	1	1	1	0	0	0	1	0	0	0	0	0	0	amphibian
tortoise	0	0	1	0	0	0	0	0	1	1	0	0	0	1	0	0	0	1	0	1	reptile
tuatara	0	0	1	0	0	0	1	1	1	1	0	0	0	1	0	0	0	1	0	0	reptile

continued on next page

Table B.1: The *zoo* demo data set *(continued from previous page)*

												legs								
Name	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	breathes	venomous	fins	2	4	5	6	8	tail	domestic	catsize	species
tuna	0	0	1	0	0	1	1	1	1	0	0	1	0	0	0	0	0	1	0	1	fish
vampire	1	0	0	1	1	0	0	1	1	1	0	0	1	0	0	0	0	1	0	0	mammal
vole	1	0	0	1	0	0	0	1	1	1	0	0	0	1	0	0	0	1	0	0	mammal
vulture	0	1	1	0	1	0	1	0	1	1	0	0	1	0	0	0	0	1	0	1	bird
wallaby	1	0	0	1	0	0	0	1	1	1	0	0	1	0	0	0	0	1	0	1	mammal
wasp	1	0	1	0	1	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	insect
wolf	1	0	0	1	0	0	1	1	1	1	0	0	0	1	0	0	0	1	0	1	mammal
worm	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	invertebrate
wren	0	1	1	0	1	0	0	0	1	1	0	0	1	0	0	0	0	1	0	0	bird

Bibliography

[AHS00] Damminda Alahakoon, Saman K. Halgamuge, and Bala Srinivasan. Dynamic self-organizing maps with controlled growth for knowledge discovery. *IEEE Transactions on Neural Networks*, 11(3):601–614, May 2000.

[Bay95] Harald F. Bayer. *Über die Anwendung selbstorganisierender Karten*. Dissertation, Universität Stuttgart, Institut für parallele und verteilte Höchstleistungsrechner, Germany, Ocotber 1995.

[BM93] Justine Blackmore and Risto Miikkulainen. Incremental grid growing: Encoding high-dimensional structure into a two-dimensional feature map. In *Proc. ICNN'93, International Conference on Neural Networks*, volume I, pages 450–455, Piscataway, NJ, 1993. IEEE Service Center.

[BM95] Justine Blackmore and Risto Miikkulainen. Visualizing high-dimensional structure with the incremental grid growing neural network. In A. Prieditis and S. Russell, editors, *Machine Learning. Proceedings of the Twelfth International Conference on Machine Learning*, pages 55–63. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1995.

[BYRN99] Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[DMB02] Michael Dittenbach, Dieter Merkl, and Helmut Berger. What customers really want from tourism information systems but never dared to ask. In *Proceedings of the 5th International Conference on Electronic Commerce Research (ICECR-5)*, Montreal, Canada, October 23–27 2002.

[DMR00a] Michael Dittenbach, Dieter Merkl, and Andreas Rauber. The Growing Hierarchical Self-Organizing Map. In S. Amari, C. L.

Giles, M. Gori, and V. Puri, editors, *Proc of the International Joint Conference on Neural Networks (IJCNN 2000)*, volume VI, pages 15 – 19, Como, Italy, July 24. – 27. 2000. IEEE Computer Society.

[DMR00b] Michael Dittenbach, Dieter Merkl, and Andreas Rauber. Using Growing Hierarchical Self-Organizing Maps for Document Classification. In *Proc of the European Symposium on Artificial Neural Networks (ESANN 2000)*, pages 7–12, Bruges, Belgium, April 26. – 28. 2000. D-Facto Publications.

[Fri92] Bernd Fritzke. Growing cell structures—a self-organizing network in *k* dimensions. In I. Aleksander and J. Taylor, editors, *Artificial Neural Networks, 2*, volume II, pages 1051–1056, Amsterdam, Netherlands, 1992. North-Holland.

[Fri95a] Bernd Fritzke. Growing grid - a self-organizing network with constant neighborhood range and adaptation strength. *Neural Processing Letters*, 2(5):9–13, 1995.

[Fri95b] Bernd Fritzke. A growing neural gas network learns topologies. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 625–632. MIT Press, Cambridge MA, 1995.

[Fri96] Bernd Fritzke. Growing self-organizing maps—why? In Michel Verleysen, editor, *Proc. ESANN'96, European Symp. on Artificial Neural Networks*, pages 61–72, Bruges, Belgium, 1996. D facto conference services.

[HA01] Victoria J. Hodge and Jim Austin. Hierarchical growing cell structures: Treegcs. *Knowledge and Data Engineering*, 13(2):207–218, 2001.

[He01] Hui Shao He. Analyziny the topology of high-dimensional data using the adapative hierarchical imcremental grid growing. Diplomarbeit, Technische Universität Wien, Austria, 2001.

[KHKL96] Teuvo Kohonen, Jussi Hynninen, Jari Kangas, and Jorma Laaksonen. SOM_PAK: The Self-Organizing Map program package. Report A31, Helsinki University of Technology, Laboratory of Computer and Information Science, January 1996.

[KKK98] Samuel Kaski, Jari Kangas, and Teuvo Kohonen. Bibliography of self-organizing map (SOM) papers 1981-1997. *Neural Computing Surveys*, 1(3&4):1–176, 1998.

[KKL$^+$00] Teuvo Kohonen, Samuel Kaski, Krista Lagus, Jarkko Salojärvi, Jukka Honkela, Vesa Paatero, , and Antti Saarela. Self-organization of a massive document collection. *IEEE Transactions on Neural Networks*, 11:574–85, 2000.

[KM98] Monika Köhle and Dieter Merkl. Experiments in gait pattern classification with neural networks of adaptive architecture. In L. Niklasson, M. Bodén, and T. Ziemke, editors, *Proceedings of ICANN98, the 8th International Conference on Artificial Neural Networks*, volume 1, pages 293–298. Springer, London, September 2-4 1998.

[Koh82] Teuvo Kohonen. Self-organizing formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69, 1982.

[Koh90] Teuvo Kohonen. The self-organizing map. In *Proceedings of the IEEE*, volume 78, pages 1464–1480, September 1990.

[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.

[Koh98] Teuvo Kohonen. Self-organizing map. *Neurocomputing*, 21(1):1–6, 1998.

[Mer97] Dieter Merkl. Exploration of text collections with hierarchical feature maps. In *Research and Development in Information Retrieval*, pages 186–195, 1997.

[Mer98a] Dieter Merkl. Self-organizing maps and software reuse. In Witold Pedrycz, W. Pedrycz, and J. F. Peters, editors, *Computational Intelligence in Software Engineering*, pages 65–95. World Scientific Publishing Co., Inc., River Edge, NJ, 1998.

[Mer98b] Dieter Merkl. Text classification with self-organizing maps: Some lessons learned. *Neurocomputing*, 21(1-32), 1998.

[MHDR03] Dieter Merkl, Shao Hui He, Michael Dittenbach, and Andreas Rauber. Adaptive hierarchical incremental grid growing: An architecture for high-dimensional data visualization. In *Proceedings of the 4th Workshop on Self-Organizing Maps*, Advances in Self-Organizing Maps, pages 293–298, Kitakyushu, Japan, September 11-14 2003.

[Mii90] Risto Miikkulainen. Script recognition with hierarchical feature maps. *Connection Science*, 2(1&2):83–101, 1990.

[MR97] Dieter Merkl and Andreas Rauber. Alternative ways for cluster visualization in self-organizing maps. In *Proceedings of WSOM'97, Workshop on Self-Organizing Maps, Espoo, Finland, June 4-6*, pages 106–111. Helsinki University of Technology, Neural Networks Research Centre, Espoo, Finland, 1997.

[MS91] Thomas Martinetz and Klaus Schulten. A "Neural-Gas" network learns topologies. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, *Proc. International Conference on Artificial Neural Networks* (Espoo, Finland), volume I, pages 397–402, Amsterdam, Netherlands, 1991. North-Holland.

[OKK03] Merja Oja, Samuel Kaski, and Teuvo Kohonen. Bibliography of self-organizing map (SOM) papers: 1998-2001 addendum. *Neural Computing Surveys*, 3:1–156, 2003.

[RB99] Andreas Rauber and Harald Bina. A metaphor graphics based representation of digital libraries on the world wide web: Using the libViewer to make metadata visible. In *DEXA Workshop 1999*, pages 286–290, 1999.

[RDM00] Andreas Rauber, Michael Dittenbach, and Dieter Merkl. Automatically detecting and organizing documents into topic hierarchies: A neural-network based approach to bookshelf creation and arrangement. In J. Borbinha and T. Baker, editors, *Proceedings of the 4. European Conference on Research and Advanced Technologies for Digital Libraries (ECDL2000)*, number LNCS 1923 in Lecture Notes in Computer Science, pages 348–351, Lisboa, Portugal, September 18. - 20. 2000. Springer.

[RM99a] Andreas Rauber and Dieter Merkl. Somlib: A digital library system based on neural networks. In *Proceedings of the 4th ACM*

Conference on Digital Libraries (DL'99), Berkeley, CA, August 11 - 14 1999. ACM.

[RM99b] Andreas Rauber and Dieter Merkl. The SOMLib digital library system. In *European Conference on Digital Libraries*, pages 323–342, 1999.

[RM01] Andreas Rauber and Dieter Merkl. Automatic labeling of self-organizing maps for information retrieval. *Journal of Systems Research and Information Systems (JSRIS)*, 10(10):23–45, December 2001.

[RMD02] Andreas Rauber, Dieter Merkl, and Michael Dittenbach. The growing hierarchical self-organizing map: Exploratory analysis of high-dimensional data. *IEEE Transactions on Neural Networks*, 13(6):1331–1341, November 2002.

[Sal89] Gerald Salton. *Automatic text processing – The Transformation, Analysis, and Retrieval of Information by Computer.* Addison-Wesley Longman Publishing Co., Inc., 1989.

[Spe96] Heike Speckmann. *Dem Denken abgeschaut: Neuronale Netze im praktischen Einsatz.* Friedr. Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig/Wiesbaden, 1996.