# A Self-Organizing Map for Adaptive Processing of Structured Data

Markus Hagenbuchner, Alessandro Sperduti, *Member, IEEE*, and Ah Chung Tsoi, *Senior Member, IEEE*

***Abstract*—Recent developments in the area of neural networks produced models capable of dealing with structured data. Here, we propose the first fully unsupervised model, namely an extension of traditional self-organizing maps (SOMs), for the processing of labeled directed acyclic graphs (DAGs). The extension is obtained by using the unfolding procedure adopted in recurrent and recursive neural networks, with the replicated neurons in the unfolded network comprising of a full SOM. This approach enables the discovery of similarities among objects including vectors consisting of numerical data. The capabilities of the model are analyzed in detail by utilizing a relatively large data set taken from an artificial benchmark problem involving visual patterns encoded as labeled DAGs. The experimental results demonstrate clearly that the proposed model is capable of exploiting both information conveyed in the labels attached to each node of the input DAGs and information encoded in the DAG topology.**

***Index Terms*—Clustering, data mining which involves novel types of data/knowledge, data reduction techniques, discovering similarities, innovative algorithms, processing labeled graphs, recurrent neural networks, recursive neural networks, self organizing maps (SOMs), vector quantization (VQ).**

## I. Introduction

MANY natural and artificial systems are more appropriately modeled using data structures. For example, an image can be segmented into various components, and each component can be considered in relation to one another in the form of a graph structure. This structured representation conveys much more information than a "flat" one, e.g., a vector of numerical features extracted from the image. The use of structured representations is convenient also for representing data from many other areas such as molecular chemistry, the world wide web, electronic documents, syntactic parsing trees, and others.

In recent years, supervised neural networks have been developed which are able to deal with structured data encoded as labeled directed acyclic graphs (DAGs). These models are called recursive neural networks and are fully described in [1]–[4]. The essential idea of recursive neural networks is to model each node of an input DAG by a multilayer perceptron, and then to process the DAG from its sink nodes toward the source node, using the structure of the DAG to connect the neurons from one node to another. The output of the neurons corresponding to the source node can then be obtained. A gradient descent method is used to learn the weights of the multilayer perceptrons. The models are simplified by assuming that all multilayer perceptrons in the DAG and across the set of training DAGs have the same parameters. This approach basically consists of an extension to DAGs of the traditional "unfolding" process adopted by recurrent neural networks for sequences [5], [6].

Theoretical properties of recursive neural networks have been studied [7]–[13] and applications are starting to emerge [14]–[17], especially in the field of chemistry [18]–[21], where chemical compounds are naturally represented in graphical form.

Supervised information, however, either may not be available or very expensive to obtain. Thus it is very important to develop models which are able to deal with structured data in an unsupervised fashion. In this paper, we will show how self-organizing maps (SOMs) [22] can be extended to treat complex objects represented by data structures. A SOM implements a discrete approximation of principal curves [23] by means of a set of neural units arranged into a topological map, usually of dimension two. The ability of the SOM to preserve topological features [24] can be exploited to perform a clustering of the data on the basis of a similarity criterion and also to induce a metric onto the input domain.

In this paper, we extend the standard SOM model [22], so as to allow the mapping of structured objects into a topological map.[1] This mapping can then be used to discover similarities among the input objects. The task performed by the proposed network, i.e., the development of a "topographic" map of the input structures and substructures in which the spatial locations of the neurons are indicative of statistical features of the structures is novel especially when considering labeled DAGs where the labels are real-valued vectors. Although SOMs have been defined to work on vector spaces, some extensions to sequences exist. The best known works include the temporal Kohonen map (TKM) [25], and the recurrent self-organizing map (RSOM) [26]. Both models use leaky integrators for the representation of the state (context). A different approach is adopted in the contextual self-organizing map (CSOM) [27], where recurrent connections between units in the map are used. The proposed extension to the domain of graphs produces an important generalization of the SOM model, while leaving the computation of context simple and efficient. As will be seen, the treatment of sequences and vectors, respectively, turn out to be special cases of our proposed general data structure model. In fact, a sequence can be seen as a labeled list (which is a special type

Manuscript received October 31, 2001; revised October 9, 2002.

M. Hagenbuchner is with the Faculty of Informatics, University of Wollongong, Wollongong NSW 2522, Australia (e-mail: markus@artificial-neural.net).

A. Sperduti is with the Dipartimento di Matematica Pura ed Applicata, Università di Padova, Padova 35122, Italy (e-mail: sperduti@math.unipd.it).

A. C. Tsoi is with the Office of Pro Vice-Chancellor, Information Technology and Communications, University of Wollongong, Australia (e-mail: act@artificial-neural.net).

Digital Object Identifier 10.1109/TNN.2003.810735

[1]We will refer to this extended version of SOM as SOM-SD, SOM for structured data.

of DAG), and a vector as a labeled graph consisting of a single node. The only other unsupervised neural-network approach to structured data we are aware of is described in [28] where the task is to find fixed-length vector representations for DAGs in an unsupervised manner using a maximum entropy approach.

The organization of this paper is as follows: Section II is an introduction to basic concepts in data structures and describes the notations used in this paper, while Section III addresses recursive neural networks. The basic idea underpinning our approach is explained in Section IV, while in Section IV-A, we discuss a slightly modified standard SOM, which is used as a component of our approach. The training algorithm for this SOM is described in Section IV-B. Section V presents a general algorithm for unsupervised learning of DAGs using the SOM technique. In Section VI, we show the results obtained by the algorithm proposed in this paper on an image processing problem where visual patterns are modeled by using DAGs. Finally, some conclusions are drawn in Section VII.

## II. Fundamental Concepts of Data Structures

We shall use lowercase bold letters to denote vectors, uppercase bold letters to denote matrices, calligraphic letters for representing graphs, and bold calligraphic letters for representing domains. We assume that instances in the learning domain are structured pieces of information described by annotated DAGs. In particular, we consider the following.[2]

1) Directed ordered acyclic graphs (DOAGs). A DOAG is a DAG $\mathcal{D}$ with vertex set $\mathrm{vert}(\mathcal{D})$ and edge set $\mathrm{egd}(\mathcal{D})$, where for each vertex $v \in \mathrm{vert}(D)$ a total order on the edges leaving from $v$ is defined.[3]
2) Directed positional acyclic graphs (DPAGs). DPAGs are a superclass of DOAGs in which it is assumed that for each vertex $v$, a bijection $P : \mathrm{egd}(\mathcal{D}) \rightarrow \mathbb{N}$ is defined on the edges leaving from $v$.

The *indegree* of node $v$ is the number of incoming edges to $v$, whereas the *outdegree* of $v$ is the number of outgoing edges from $v$.

We shall require the DAG (either DOAG or DPAG) to possess a supersource, i.e., a vertex $s \in \mathrm{vert}(\mathcal{D})$ such that every vertex in $\mathrm{vert}(\mathcal{D})$ can be reached by a directed path starting from $s$. If no supersource is present, a new node connected with all the nodes of the graph having null *indegree* can be added.

Given a DAG $\mathcal{D}$ and $v \in \mathrm{vert}(\mathcal{D})$, we denote by $ch[v]$ the set of children of $v$, and the $k$th child of $v$ by $ch_k[v]$.

The data structures that we consider are labeled DAGs, where labels are tuples of variables and are attached to vertices and possibly to edges. These variables express features attached to a given node. The label attached to node $v$ is coded by values in $\boldsymbol{\mathcal{U}} \subset \mathbb{R}^m$. When using connectionist models either numerical or categorical variables assume real-valued representations with values in $\boldsymbol{\mathcal{U}}$. Subscript notation will be used when referring to the labels attached to vertices in a data structure. Hence, $\boldsymbol{y}_v$ denotes the vector of variables labeling vertex $v \in \mathrm{vert}(\mathcal{D})$.

[2]The model described in this paper allows the processing of some special classes of directed cyclic graphs. We will not define these as our main aim is the processing of acyclic graphs such as directed positional acyclic graphs, and trees.

[3]For example, in the case of graphs representing logical terms, the order on outgoing edges is immediately induced by the order of the function arguments.

Note that DAGs with labeled edges can be reduced to DAGs having only labels on the nodes. A straightforward method for reducing structures with labeled edges to structures with unlabeled edges is to move each label attached to an edge leaving a given node $v$ to the label attached to node $v$.

Given a data structure $\mathcal{D}$, the DAG obtained by ignoring all node labels will be referred to as the *skeleton* of $\mathcal{D}$, denoted $\mathrm{skel}(\mathcal{D})$. Clearly, any two data structures can be distinguished because they have different skeletons, or, if they have the same skeleton, because they have different node labels.

In the following, we shall denote by $\#^{(c)}$ the class of DAGs with maximum outdegree $c$. A generic class of DAGs with bounded (but unspecified) outdegree, will simply be denoted by $\#$. The class of all data structures defined over the label universe domain $\boldsymbol{\mathcal{Y}}$ and skeleton in $\#^{(c)}$ will be denoted as $\boldsymbol{\mathcal{Y}}^{\#^{(c)}}$. The void DAG will be denoted by the special symbol $\xi$.

In Fig. 1, we have shown some examples of visual patterns represented as DOAGs. Each node of the graph represents a different colored component of the visual pattern, while edges are used to encode the concept of "connected with." The direction of the edges is decided on the basis of a procedure which scans the picture bottom-up and from left to right (see Section VI for more details).

## III. Supervised Learning: Recursive Neural Networks

In this section, we briefly describe how supervised learning can be performed by neural networks in structured domains. This description is useful, since it helps in understanding how unsupervised learning can be defined by using a computational framework which is essentially the same as that defined for recursive neural networks.

Recursive neural networks described in [29] are neural networks capable of performing mappings from a set of labeled graphs to a set of real vectors. Specifically, the class of functions which can be realized by a recursive neural network can be characterized as the class of functional graph transductions $\tau : \mathcal{I}^{\#} \rightarrow \mathbb{R}^k$, where $\mathcal{I} = \mathbb{R}^m$, which can be represented in the following form $\tau = g \circ \widehat{\tau}$, where $\widehat{\tau} : \mathcal{I}^{\#} \rightarrow \mathbb{R}^n$ is the *encoding* (or *state transition*) function and $g : \mathbb{R}^n \rightarrow \mathbb{R}^k$ is the *output* function. Specifically, given a DOAG $\mathcal{D}$, $\widehat{\tau}$ is defined recursively as

$$\widehat{\tau}(\mathcal{D}) = \begin{cases} \mathbf{0}\ (\text{the null vector in } \mathbb{R}^n), & \text{if } \mathcal{D} = \xi \\ \tau(\boldsymbol{y}_s, \widehat{\tau}(\mathcal{D}^{(1)}), \ldots, \widehat{\tau}(\mathcal{D}^{(c)})), & \text{otherwise} \end{cases} \tag{1}$$

where $\tau$ is defined as

$$\tau : \mathbb{R}^m \times \underbrace{\mathbb{R}^n \times \cdots \times \mathbb{R}^n}_{c\ \text{times}} \rightarrow \mathbb{R}^n \tag{2}$$

where $\mathbb{R}^m$ denotes the label space, while the remaining domains represent the encoded subgraphs spaces up to the maximum outdegree of the input domain $\mathcal{I}^{\#}$, $c$ is the maximum out-degree of DOAGs in $\mathcal{I}^{\#}$, $s = \mathrm{source}(\mathcal{D})$, $\boldsymbol{y}_s$ is the label attached to the super-source of $\mathcal{D}$, and $\mathcal{D}^{(1)}, \ldots, \mathcal{D}^{(c)}$ are the subgraphs pointed by $s$. A typical neural realization for $\tau$ is

$$\tau(\boldsymbol{u}_v, \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(c)}) = \boldsymbol{F}(\boldsymbol{W}\boldsymbol{u}_v + \sum_{j=1}^{c} \widehat{\boldsymbol{W}}_j \boldsymbol{x}^{(j)} + \boldsymbol{\theta}) \tag{3}$$

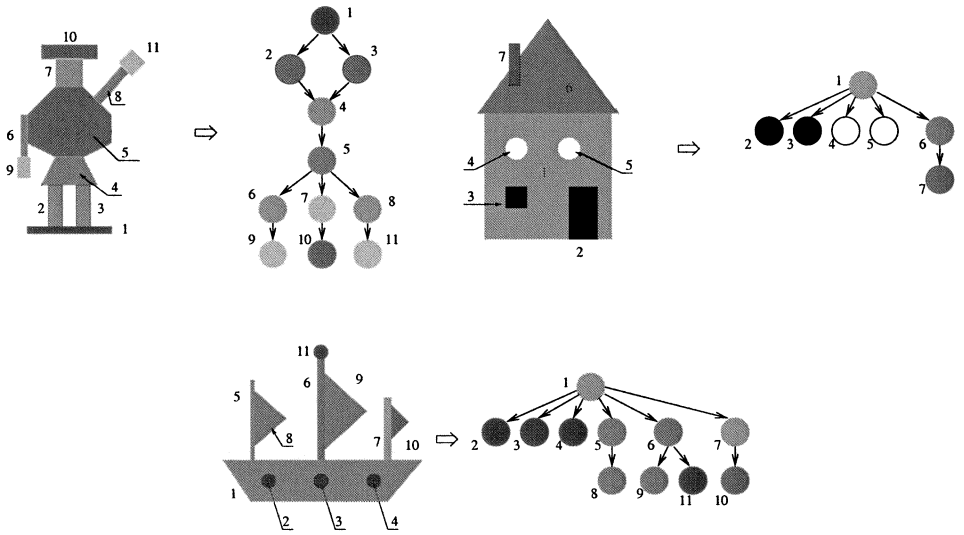

Fig. 1. Examples of DOAGs representing visual patterns. Labels are not shown.

where $\boldsymbol{F}_i(\boldsymbol{v}) = sgd(v_i)$ (sigmoidal function), $\boldsymbol{u}_v \in \mathbb{R}^m$ is a label, $\boldsymbol{\theta} \in \mathbb{R}^n$ is the bias vector, $\boldsymbol{W} \in \mathbb{R}^{m \times n}$ is the weight matrix associated with the label space, $\boldsymbol{x}^{(j)} \in \mathbb{R}^m$ are the vectorial codes obtained by the application of the encoding function $\widehat{\tau}$ to the subgraphs $\mathcal{D}^{(j)}$ (i.e., $\boldsymbol{x}^{(j)} = \widehat{\tau}(\mathcal{D}^{(j)})$), and $\widehat{\boldsymbol{W}}_j \in \mathbb{R}^{m \times m}$ is the weight matrix associated with the $j$th subgraph space.

The output function $g$ maps the state transition space to a $k$-dimensional output space and is generally realized by a feedforward network, i.e., it is realized as a linear combination of the outputs of the hidden layer neurons in the hidden layer of the output node.

Given a training set $T = \{(\mathcal{D}_i, t_i)\}_{i=1,\ldots,N}$, where for each data structure $\mathcal{D}_i$ a desired target value $t_i$ is associated, using (1), (3), and the neural network implementing the output function $g$, for each $\mathcal{D}_i$ a feedforward network can be generated and trained so as to match the corresponding desired target value $t_i$. Since the weights are shared among all the generated feedforward networks, the training converges to a set of weight values which reproduces the desired target value, to within a prescribed error tolerance, for each data structure in the training set.

The function $\tau$ maps from a higher dimensional space (i.e., $m + c \cdot n$) to a lower dimensional space (i.e., $n$). Thus, the role of $\tau$ consists of compressing the information about a node (i.e., the label, and the compressed information about the children of the node) in a vector of dimension $n$. This observation is fundamental to understand how (1) can be adapted to unsupervised learning within a SOM approach. In fact, the aim of the SOM learning algorithm is to learn a *feature map*

$$\mathcal{M} : \boldsymbol{\mathcal{I}} \rightarrow \boldsymbol{\mathcal{A}} \tag{4}$$

which given a vector in the spatially continuous input space $\boldsymbol{\mathcal{I}}$ returns a point in the spatially *discrete* output display space $\boldsymbol{\mathcal{A}}$. This is obtained in the SOM by associating each point in $\boldsymbol{\mathcal{A}}$ with a different neuron. Moreover, the output space $\boldsymbol{\mathcal{A}}$ topology is typically endowed by arranging this set of neurons as the computation nodes of a one- or two-dimensional lattice. Given an input vector $\boldsymbol{v}$, the SOM returns the coordinates within $\boldsymbol{\mathcal{A}}$ of the neuron with the closest weight vector. Thus, the set of neurons induce a partition of the input space $\boldsymbol{\mathcal{I}}$. In typical applications $\boldsymbol{\mathcal{I}} \equiv \mathbb{R}^m$, where $m \gg 2$, and $\boldsymbol{\mathcal{A}}$ is given by a two dimensional lattice of neurons. With this setting, high-dimensional input vectors are projected into the two-dimensional coordinates of the lattice, with the aim of preserving, as much as possible, the topological relationships among the input vectors, i.e., input vectors which are close to each other should maximally activate neighbor neurons in the lattice. The SOM is, thus, performing data reduction via a vector quantization approach,[4] and the coordinates of the winning neuron can be understood as the compressed version of the input vector.

In the next section, we show how a SOM, due to its data reduction ability, can be used to implement the $\tau$ function in an unsupervised learning framework.

## IV. Unsupervised Learning Within an SOM Framework

We are interested in generalizing (4) to deal with the case $\boldsymbol{\mathcal{I}} \equiv \boldsymbol{\mathcal{Y}}^{\#^{(c)}}$, i.e., the input space is a structured domain with labels in $\boldsymbol{\mathcal{Y}}$. In this case, we need to specify how the function

$$\mathcal{M}^{\#} : \boldsymbol{\mathcal{Y}}^{\#^{(c)}} \rightarrow \boldsymbol{\mathcal{A}} \tag{5}$$

is realized. One possibility is to redefine (1) as shown in (6) at the bottom of the page, where $s = \text{source}(\mathcal{D})$, $\mathcal{D}^{(1)}, \ldots, \mathcal{D}^{(c)}$ are the (eventually void) subgraphs pointed by the outgoing

[4]It can be shown that a SOM is actually related to a discrete version of the principal curves algorithm [23].

$$\mathcal{M}^{\#}(\mathcal{D}) = \begin{cases} nil_{\boldsymbol{\mathcal{A}}}, & \text{if } \mathcal{D} = \xi \\ \mathcal{M}_{\text{node}}\left(\boldsymbol{y}_s, \mathcal{M}^{\#}(\mathcal{D}^{(1)}), \ldots, \mathcal{M}^{\#}(\mathcal{D}^{(c)})\right), & \text{otherwise} \end{cases} \tag{6}$$

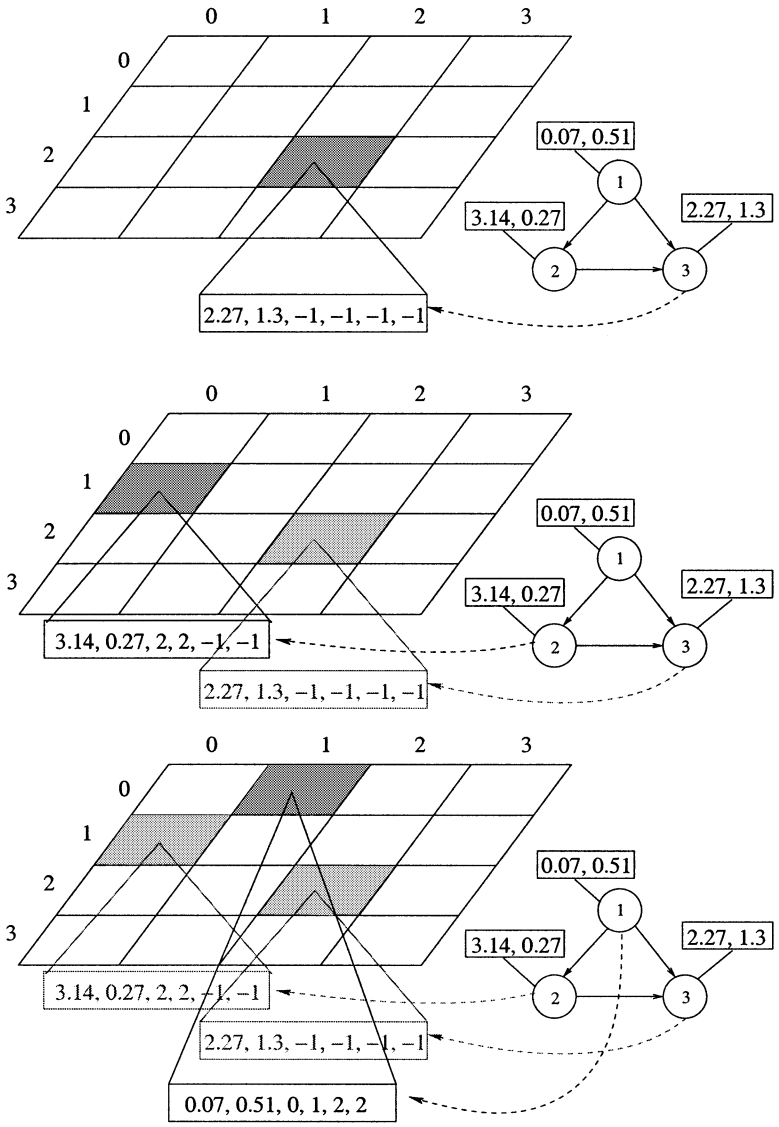


Fig. 2. Example of computation of $\mathcal{M}^{\#}$ for an input graph. The picture shows the multiple application of $\mathcal{M}_{\mathrm{node}}$ to the nodes of the graph. First of all, the leaf node 3 is presented to $\mathcal{M}_{\mathrm{node}}$ (top), where the null coordinates are represented by $(-1,-1)$. The winning neuron has coordinates (2,2). This information is used to define the input vector representing node 2. This vector is then presented to $\mathcal{M}_{\mathrm{node}}$ (middle) and the winning neuron is found to have coordinates (0,1). Using both this information and the previous one, the input vector for node 1 can be composed and presented to $\mathcal{M}_{\mathrm{node}}$ (bottom). This time, the winning neuron is found to have coordinates (1,0) and it is associated to the whole graph, i.e., $\mathcal{M}^{\#}(\mathcal{D}) = (1,0)$.

edges leaving from $s$, $nil_{\mathcal{A}}$ is a special coordinate vector into the discrete output space $\mathcal{A}$, and

$$\mathcal{M}_{\mathrm{node}} : \mathcal{Y} \times \underbrace{\mathcal{A} \times \cdots \times \mathcal{A}}_{c \text{ times}} \to \mathcal{A} \tag{7}$$

is a SOM, defined on a generic node, which takes as input the label of the node and the "encoding" of the subgraphs $\mathcal{D}^{(1)}, \ldots, \mathcal{D}^{(c)}$ according to the $\mathcal{M}^{\#}$ map. By "unfolding" the recursive definition in (6), it turns out that $\mathcal{M}^{\#}(\mathcal{D})$ can be computed by starting to apply $\mathcal{M}_{\mathrm{node}}$ to leaf nodes (i.e., nodes with null outdegree), and proceeding with the application of $\mathcal{M}_{\mathrm{node}}$ bottom-up from the frontier nodes (sink nodes) to the supersource of the graph $\mathcal{D}$.

Notice how $\mathcal{M}_{\mathrm{node}}$ in (6) is playing exactly the same role of $\tau$ in (1), with the difference that $\tau$ returns a real-valued vector representing a *reduced descriptor* of the node, while $\mathcal{M}_{\mathrm{node}}$ returns the coordinates of the winning neuron, which, due to the data reduction capability of the SOM, still constitutes a *reduced descriptor* of the node. An example of how the computation described above proceeds is shown in Fig. 2. In this case, $nil_{\mathcal{A}}$ is represented by the coordinates $(-1,-1)$. Note that a highlighted neuron in Fig. 2 refers to the best matching neuron for the given input vector.

### A. Model of $\mathcal{M}_{\mathrm{node}}$

In the previous section, we saw that the computation of $\mathcal{M}^{\#}$ can be recast as the recursive application of the $\mathcal{M}_{\mathrm{node}}$ SOM to the nodes compounding the input structure. Moreover, the recursive scheme for graph $\mathcal{D}$ follows the skeleton $\mathrm{skel}(\mathcal{D})$ of the graph. In this section, we give implementation details on $\mathcal{M}_{\mathrm{node}}$ SOM.

We assume that each label in $\mathcal{Y}$ is encoded in $\mathcal{U} \subset \mathbb{R}^m$. Thus, for each node $v$ in $\mathrm{vert}(\mathcal{D})$, we have a vector $\boldsymbol{u}_v$ of dimension $m$. Moreover, we realize the display output space $\mathcal{A}$ through a $q$

dimensional lattice of neurons.[5] We assume that each dimension of the $q$ dimensional lattice is quantized into integers, $n_i$, $i = 1, 2, \ldots, q$, i.e., $\boldsymbol{\mathcal{A}} \equiv [1 \ldots n_1] \times [1 \ldots n_2] \times \cdots \times [1 \ldots n_q]$. The total number of neurons is $\prod_{i=1}^{q} n_i$, and each "point" in the lattice can be represented by a $q$ dimensional coordinate vector $\boldsymbol{c}$. For example, if $q = 2$, and if we have $n_1$ neurons on the horizontal axis and $n_2$ neurons on the vertical axis, then the winning neuron is represented by the coordinate vector $\boldsymbol{c} \equiv (c_1, c_2) \in [1 \ldots n_1] \times [1 \ldots n_2]$ of the neuron which is most active in this two-dimensional lattice.

With the above assumptions, we have

$$\mathcal{M}_{\text{node}} : \mathbb{R}^m \times ([1 \ldots n_1] \times \cdots \times [1 \ldots n_q])^c \rightarrow [1 \ldots n_1] \times \cdots \times [1 \ldots n_q] \quad (8)$$

and the $m + cq$ dimensional input vector $\boldsymbol{v}$ to $\mathcal{M}_{\text{node}}$, representing the information about a generic node $v$, is defined as

$$\boldsymbol{v} = \left[\boldsymbol{u}_v \; \boldsymbol{x}_{ch_1[v]} \; \boldsymbol{x}_{ch_2[v]} \; \ldots \; \boldsymbol{x}_{ch_c[v]}\right] \quad (9)$$

where $\boldsymbol{x}_{ch_i[v]}$ is the coordinate vector of the winning neuron for the subgraph pointed by the $i$-th pointer of $v$. In addition, we have to specify how $nil_{\boldsymbol{\mathcal{A}}}$ is defined. $nil_{\boldsymbol{\mathcal{A}}}$ must be chosen to be a coordinate vector outside the range of valid coordinates such as $(-1, \ldots, -1)$ which is of dimension $q$.

Of course, each neuron with coordinates vector $\boldsymbol{c}_j$ in the $q$-dimensional lattice will have an associated vector weight $\boldsymbol{w}_{\boldsymbol{c}_j} \in \mathbb{R}^{m+cq}$.

Notice that, given a DAG $\mathcal{D}$, in order to compute $\mathcal{M}^{\#}(\mathcal{D})$, the SOM $\mathcal{M}_{\text{node}}$ must be recursively applied to the nodes of $\mathcal{D}$. One node can be processed only if all the subgraphs pointed by it have already been processed by $\mathcal{M}_{\text{node}}$. Thus, the computation can be parallelized on the graph, with the condition that the above constraint is not violated. A data flow model of computation fits completely this scenario. When considering a sequential model of computation, a node update scheduling constituted by any inverted topological order [30] for the nodes of the graph suffices to guarantee the correct computation of $\mathcal{M}^{\#}$.

Finally, it is observed that, not only is the SOM $\mathcal{M}_{\text{node}}$ formally representing single graph nodes, it is also "coding" information about the structures. This happens because of the structural information conveyed by the $\boldsymbol{x}_{ch_i[v]}$ used as part of the input vectors. Thus, some neurons of the map will be maximally active only for some leaf nodes, others will be maximally active only for some nodes which are roots of graphs, and so on.

### *B. Training Algorithm for $\mathcal{M}_{\text{node}}$*

The weights associated with each neuron in the $q$ dimensional lattice $\mathcal{M}_{\text{node}}$ can be trained using the following two-step process.

*Step 1—Competitive Step:* In this step, the neuron which is most similar to the input node $\boldsymbol{v}$ (defined as in (9)) is chosen. Specifically, the (winning) neuron, at iteration $t$, with the closest weight vector is selected as follows:

$$\boldsymbol{c}_{i^*}(t) = \arg\min_{\boldsymbol{c}_i} \|\Lambda(\boldsymbol{v}(t) - \boldsymbol{w}_{\boldsymbol{c}_i}(t))\| \quad (10)$$

where $\Lambda$ is a $(m+cq) \times (m+cq)$ diagonal matrix which is used to balance the importance of the label versus the importance of the pointers. In fact, the elements $\lambda_{1,1}, \ldots, \lambda_{m,m}$ are set to $\mu_1$, while the remaining elements $\lambda_{m+1,m+1}, \ldots, \lambda_{m+cq,m+cq}$ are set to $\mu_2$. Notice that if $\Lambda$ is equal to the identity matrix, then the standard SOM algorithm is obtained.

*Step 2—Cooperative Step:* The weight vector $\boldsymbol{w}_{\boldsymbol{c}_{i^*}}$, as well as the weight vector of neurons in the topological neighborhood of the winning neuron, are moved closer to the input vector

$$\boldsymbol{w}_{\boldsymbol{c}_r}(t+1) = \boldsymbol{w}_{\boldsymbol{c}_r}(t) + \eta(t) f(\Delta_{i^* r})(\boldsymbol{v}(t) - \boldsymbol{w}_{\boldsymbol{c}_r}(t)) \quad (11)$$

where the magnitude of the attraction is governed by the learning rate $\eta$ and by a neighborhood function $f(\Delta_{i^* r})$. $\Delta_{i^* r}$ is the topological distance between $\boldsymbol{c}_r$ and $\boldsymbol{c}_{i^*}$ in the lattice, i.e., $\Delta_{i^* r} = \|\boldsymbol{c}_r - \boldsymbol{c}_{i^*}\|$, and it controls the amount to which the weights of the neighboring neurons are updated. Typically, the neighborhood function $f(\cdot)$ takes the form of a Gaussian function

$$f(\Delta_{i^* r}) = \exp\left(-\frac{\Delta_{i^* r}^2}{2\sigma^2}\right) \quad (12)$$

where $\sigma$ is the spread which determines the *neighborhood size*. As the learning proceeds and new input vectors are given to the map, the learning rate gradually decreases to zero according to the specified learning rate function type. Along with the learning rate, the neighborhood radius decreases as well.[6]

## V. Training Algorithm for $\mathcal{M}^{\#}$

It is now quite clear how to perform the training of $\mathcal{M}^{\#}$ by using the definition and training algorithm for $\mathcal{M}_{\text{node}}$. Here is the training algorithm.

```
Training Algorithm for M#
input: Set of training DAGs T = {D_i}_{i=1,...,N},
 c maximum outdegree of DAGs in T, map
 M_node;
begin
randomly set the weights for M_node;
repeat
 randomly select D ∈ T with uniform dis-
 tribution;
 List(D)  ← an inverted topological order
 for vert(D);
 for v ← first(List(D)) to last(List(D))do
  train(M_node([u_v M#(ch_1[v]) M#(ch_2[v]) ···
  M#(ch_c[v])]));
end
```

The statement

$$train(\mathcal{M}_{\text{node}}([\boldsymbol{u}_v \, \mathcal{M}^{\#}(ch_1[v]) \times \mathcal{M}^{\#}(ch_2[v]) \cdots \mathcal{M}^{\#}(ch_c[v])]))$$

means that training corresponding to (10) and (11) is performed just once for the input vector constituted by the label vector $\boldsymbol{u}_v$ and the updated coordinates for the subgraphs rooted in $ch_i[v]$. The coordinates for the subgraphs rooted in $ch_i[v]$

[5] Often, in SOM, $q = 2$.

[6] Generally, the neighborhood radius in SOMs never decreases to zero. Otherwise, if the neighborhood size becomes zero, the algorithm reduces to vector quantization (VQ) and no longer has topological ordering properties [31].

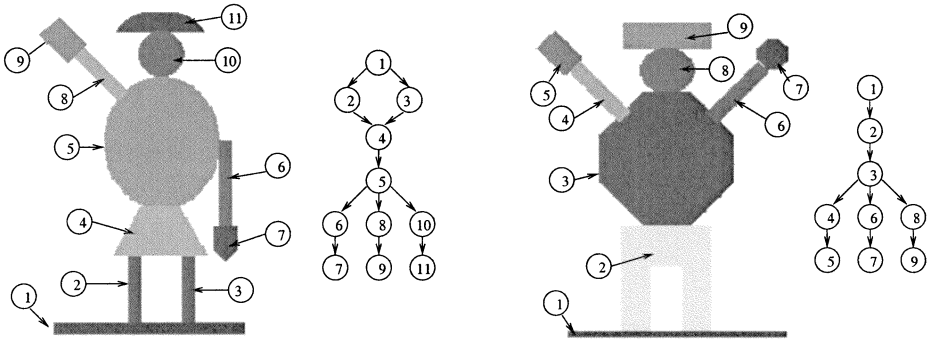


Fig. 3. Slection of artificially generated traffic policemen images and extracted graph structure. Data labels are not shown. Nodes are numbered to indicate which element they represent.

must be recomputed from scratch (i.e., $\mathcal{M}^{\#}(ch_i[v])$) at each iteration, since training changes the weights of the neurons and so the winning neurons may change as well. Notice that the adaptation for the neighborhood radius and the learning rate of $\mathcal{M}_{\rm node}$ follows the standard schedule.

The above algorithm is computationally quite expensive because of the updating of the coordinates for the subgraphs before performing a step of training. However, since the graphs in the training set must be presented several times, the following stochastic approximation to the algorithm can be defined.

```
Stochastic Training Algorithm for M#
input: Set of training DAGs T = {D_i}_{i=1,...,N},
 c maximum outdegree of DAGs in T, map
 M_node;
begin
randomly set the weights for M_node;
repeat
 randomly select D ∈ T with uniform dis-
 tribution;
 List(D) ← an inverted topological order
 for vert(D);
 for v ← first(List(D)) to last(List(D))do
  train(M_node([u_v x_ch1[v] x_ch2[v] ··· x_chc[v]]));
  x_v ← M_node([u_v x_ch1[v] x_ch2[v] ··· x_chc[v]]);
end
```

In this version, the coordinates for the (sub)graphs are stored in $\boldsymbol{x}_v$, once for each processing of graph $\mathcal{D}$, and then used when needed[7] for the training of $\mathcal{M}_{\rm node}$. Of course, the stored vector is an approximation of the true coordinate vector for the graph rooted in $v$, however, if the learning rate $\eta$ is small, this approximation may be acceptable.

## VI. Experiments

For the experiments, we used an extended version of the *Policemen Benchmark*, which was obtained from [32]. In essence, the *Policemen Benchmark* is a set of labeled DOAGs extracted from images produced by means of a context free attributed plex grammar. The dataset consists of visual patterns and associated graph structures from three different domains. A number of classes were defined for each domain. Structural representations had been obtained by a scan line algorithm where images are scanned from bottom to top, and from left to right. The first colored object found constitutes the root node. Objects directly connected to it form the offsprings. Applying this procedure recursively, all objects are considered and a graph representation is obtained. Each node in the graph receives a two-dimensional label stating the $\{x, y\}$ coordinate of the center of gravity of the corresponding object. Examples are illustrated in Figs. 3–5.

The result is a dataset that provides directed acyclic graphs with the following properties.

| | Outdegr. | Depth | | Num. nodes | | Num. |
|---|---|---|---|---|---|---|
| Data set | Max. | Min | Max | Min | Max | classes |
| Policemen | 3 | 4 | 5 | 9 | 11 | 2 |
| Houses | 5 | 2 | 3 | 4 | 7 | 8 |
| Ships | 6 | 1 | 2 | 3 | 13 | 2 |

Hence, policemen patterns produce deep narrow graphs, ships and houses have a flat and wide data structure. Some graph structures produced by the ships and houses are identical in structure such as house648 in Fig. 4 and ship1034 in Fig. 5. There is no graph structure derived from policemen images that is contained in the domain houses or ships. Thus, some patterns can be distinguished only through features encoded in the labels. For example, when considering policemen, the location of the arm is not encoded in the graph structure, but in the data label, while the graph structure is not affected.

The maximum outdegree of all nodes in the data set is six. As a result (with the dimension of the data label $\mathbf{u} = 2$) every input vector $\boldsymbol{v}$ is of dimension $2 + 2 \times 6$. The training (test) set contained 3750 (3750) graphs with a total of 29 864 (29 887) nodes. For the purpose of testing the network performance, 12 classes are defined as shown in Table I. These classes are defined so that some of them can only be distinguished by considering information provided through data labels such as class "a" and class "b." Some classes require structural information in order to be distinguished such as classes "h" and "k."

[7]Notice that the use of an inverted topological order guarantees that the updating of the coordinate vectors $\boldsymbol{x}_v$ is done before the use of $\boldsymbol{x}_v$ for training.

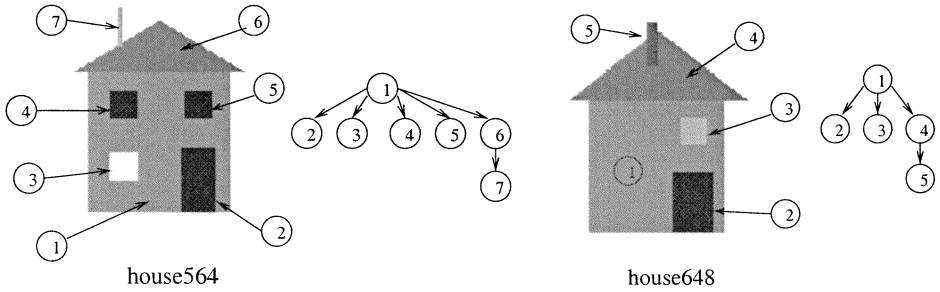


Fig. 4. Artificially generated houses and associated graph structure. Data labels are not shown. Nodes are numbered to indicate which element is represented.

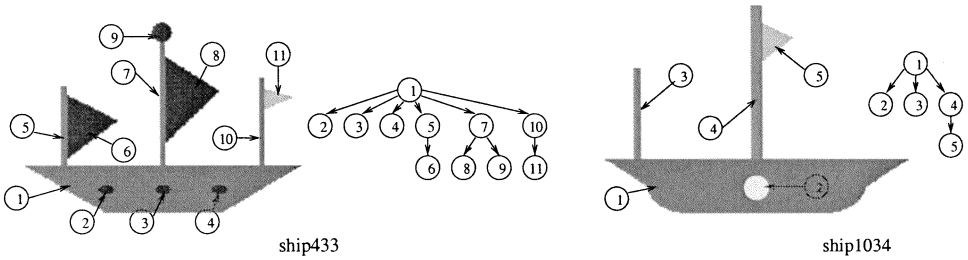


Fig. 5. Artificially generated images of ships. Data labels are not shown.

TABLE I
DEFINITION OF THE 12 CLASSES AND THEIR SYMBOLS AS USED LATER IN FIGS. 6–13. (LL) MEANS "LOWER LEFT," (UL) "UPPER LEFT," AND (UR) "UPPER RIGHT"

| Class | Description | Symbol | Samples in set | |
|---|---|---|---|---|
| | | | Train | Test |
| 'a' | Policemen with raised left arm | × | 645 | 670 |
| 'b' | Policemen with lowered left arm | ○ | 605 | 580 |
| 'c' | Ships featuring two masts | □ | 937 | 944 |
| 'd' | Ships with three masts | ◇ | 313 | 306 |
| 'e' | Houses without windows | + | 28 | 21 |
| 'f' | Houses with 1 window in (LL) corner | ✳ | 59 | 59 |
| 'g' | Houses with 1 window in (UR) corner | △ | 58 | 64 |
| 'h' | Houses with 1 window in (UL) corner | ■ | 172 | 195 |
| 'i' | Houses with 2 windows in (LL) and (UL) | ● | 53 | 71 |
| 'j' | Houses with 2 windows in (UL)) and (UR) | ▲ | 170 | 173 |
| 'k' | Houses with 2 windows in (LL) and (UR) | ▽ | 188 | 194 |
| 'l' | Houses with all three windows | ▼ | 522 | 473 |
| | | Total: | 3750 | 3750 |

Experiments were designed to demonstrate the typical behavior of SOM-SD networks in a given environment. Networks were trained on the same data set where class membership information was omitted during training. A software implementation of the SOM-SD model is available for download from http://www.artificial-neural.net/simulators.html.

### A. Results

This section discusses the behavior of sample SOM networks for data structures ($\mathcal{M}^{\#}$) when applied to the data set described earlier. Training, of course, is performed unsupervised. However, we utilize symbols to code the plots according to the original class membership of the data. The visualization of these classes gives indications on how well the network encodes structures and the data labels. All the results described in this section are obtained by SOM networks with a hexagonal topology and a Gaussian neighborhood function. Training was performed by utilizing the stochastic training algorithm introduced earlier.

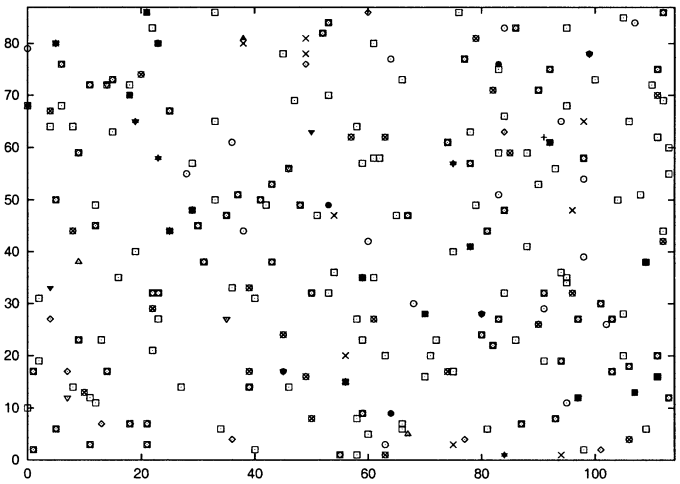

Fig. 6. Mapping of the root nodes on a randomly initialized network ($\mathcal{M}_{\mathrm{node}}$) of size $114 \times 87 \hat{=} 9918$ neurons (ratio 1:1.31). To avoid the cluttering of symbols, the mapping of nodes other than root nodes is not shown here.

Here, we present a network for $\mathcal{M}_{\mathrm{node}}$ with about one third as many neurons as nodes in the data set. For this, we generated a $114 \times 87$ output map, and initialized it with random values.[8] Fig. 6 shows the state of the initial network. Displayed are the locations of neurons that were activated by root nodes only. It can be observed that, at this initial state, the data is distributed randomly across the map. Note that there are less neurons marked than the number of root nodes in the dataset. This is because

[8]Random values are chosen from within a range of valid values which were obtained by scanning the data set once.

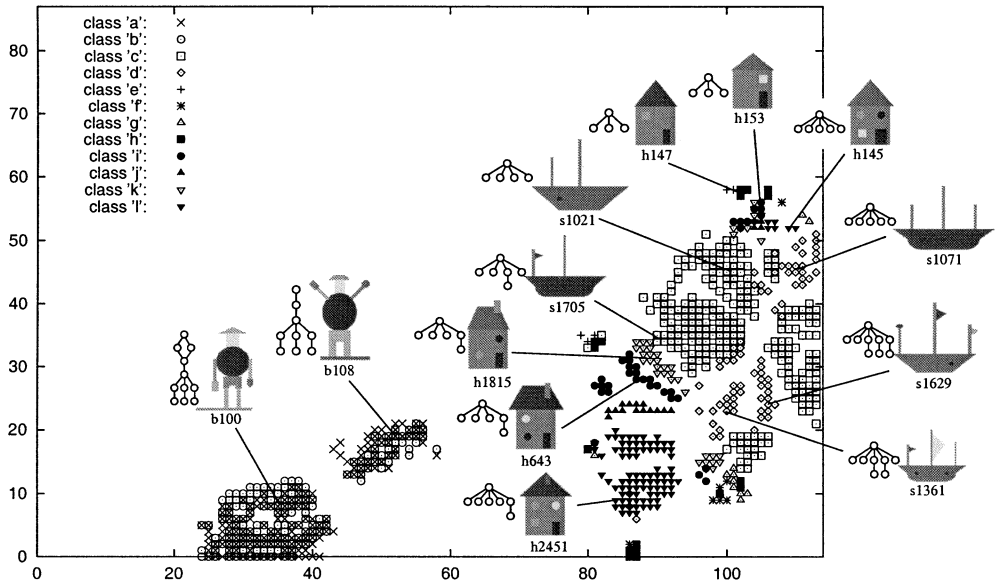


Fig. 7. Mapping of the root nodes after training a network of size $114 \times 87$ for a total of 350 iterations. A typical sample has been retrieved for many subclusters and are displayed accordingly. The graph representation of a sample is displayed to its left, and its identity is plotted underneath.

some neurons were activated by several root nodes. In the case where root nodes belonging to different classes activated by the same neuron, this is made visible through overlapping symbols (e.g., a cross within a square).

In a first attempt, the network was trained for a total of 350 iterations. The initial learning parameter $\eta(0)$ was set to 0.08, the initial neighborhood spread $\sigma$ was 60. During training, in order to assure convergence, the learning rate gradually decreased to zero while the neighborhood radius decreased linearly to one. The vector components in $\boldsymbol{v}$ were weighted with $\mu_1 = 1.0$ and $\mu_2 = 1.9$.[9] The resulting mapping of the root nodes after training is shown in Fig. 7.

It can be observed that during training, the mapping of root nodes has drifted into clusters, and that the network has found an ordering that discriminates the classes: houses, ships, and policemen, i.e., the network is able to encode structural information. A further observation is that policemen patterns are represented in two distinct clusters but not according to the two policemen classes defined earlier. Upon closer inspection we found that the cluster in the upper right corner is formed by all patterns featuring policemen that show two legs (wear a skirt or short pants) such as the pattern identified as b100, whereas the other cluster next to it holds all policemen wearing long pants (no legs visible) such as in pattern b108. The difference between these two instances is that graphs obtained from policemen wearing long pants have a root node with a single offspring, where the single offspring is the long pants. All other graphs featured two offsprings, one for each visible leg. Hence, the network seems to have difficulties to finely encode the information given in the data label. For example, policemen with raised arm and policemen with lowered arm, which can only be discriminated by looking at the label attached to the node representing the arm, are not mapped onto two distinct clusters. This is an issue which will be considered further later in this section.

Another interesting observation is that patterns belonging to the domain: ships are mapped onto a relatively large region whereas the mapping of houses is occurring in two regions, one of which is just above the region that mapped the ships, the other is below. We found that the houses mapped into the lower region featured a chimney so that the associated graph is of depth two. Houses mapped into the upper region did not feature a chimney so that associated graphs were of depth one. Interestingly, the same was true for the ships. Ships mapped closer to the upper end of the region did not feature flags so that the graphs representing those ships were also of depth one. Ships mapped closer to the lower end of the region always featured some flags and thus, associated graphs were of depth 2. Similarly, we found that houses and ships were mapped closer together if featuring the same out-degree such as illustrated through pattern h1815 and s1705, or pattern s1071 and h145.

In addition, it is found that most classes were mapped onto distinct clusters with little overlap between the clusters. Since some clusters represented patterns with identical structure such as the clusters associated with pattern h1815 and h643, the network has demonstrated that it is able to distinguish graphs by considering information stored in the data labels.

There is very little overlap between clusters formed by different classes with the exception for the graphs produced from policemen patterns. This finding suggests that it is probably more advisable to train the network with a modified set of weights $\mu_i$ so that the focus on the data label is strengthened. Experiments shown later in this section will find that the opposite is true. Nevertheless, the SOM-SD has produced an impressive first result given that the general performance of this network was near 92.03% on the training data, and 91.52% on the validation set.

[9]Input vectors are a composition of two components, the data label $\mathbf{u}$ and a coordinate vector $\mathbf{x}_{ch}[v]$. The contribution of these components to the Euclidean distance is better balanced using $\mu_1 = 1.0$ and $\mu_2 = 1.9$. These parameters were obtained by considering the dimensionality of the vector components, and the magnitude of its elements (averaged over the dataset).

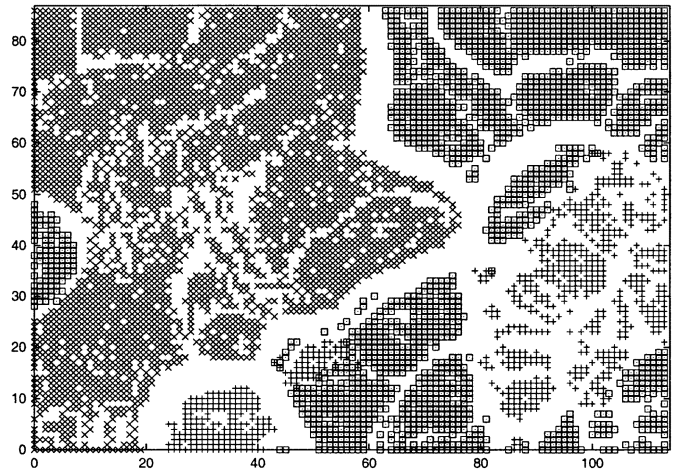


Fig. 8. Neurons as activated by nodes from the training set. Plus shaped symbols mark the location of neurons that represent root nodes, the squares are neurons activated by intermediate nodes. Codebook entries that won the competition for leaf nodes are shown as crosses.

From Fig. 7, we can assume that the empty space on the map is consumed by all nodes other than the root nodes. This assumption can be confirmed by considering Fig. 8. In this figure, we display neurons whose codebook vector resulted in the winner for at least one node in the input data set. Activated neurons are marked depending on the type of nodes that activated the neuron. The plus shaped symbols are neurons that were activated by the root nodes independently of the class to which they belong. Square shaped symbols indicate neurons that were associated with intermediate nodes, and neurons marked by crosses were activated by at least one leaf node.

It can be observed that there are areas for which neurons were not activated by any node in the data set. It is found that 3894 neurons (39.26%) are unused. We will find later, that the large number of unused neurons contribute to a good generalization performance. Of the 6024 neurons activated, 785 neurons are used by root nodes, 2022 by intermediate nodes, and 3227 by leaf nodes. Ten neurons were activated by at least one root and one intermediate node (overlap). There was no overlap between intermediate and leaf nodes or between leaf nodes and roots. In fact, in almost all other experiments we found only few neurons that were activated by two different types of nodes (e.g., intermediate and root nodes).

At this point, we know where the root and leaf nodes are located and that root nodes are ordered mainly according to a structural criterion. But what about intermediate nodes? One would expect that intermediate nodes located closer to a leaf node of a graph are also mapped to nearby codebook entries representing leaf nodes. Also, intermediate nodes nearer to the root are expected to be mapped near codebook entries representing root nodes. Hence, one would assume that the mapping of intermediate nodes drifts from clusters representing leaf nodes toward clusters holding root nodes depending on the relative position of the intermediate node within the graph. This assumption is confirmed through the sequence of plots shown in Fig. 9. This plot shows nodes obtained from policemen images according to their relative position within the graph. We considered plotting nodes associated with policemen patterns since they featured the deepest graph structure, and hence are better suited to demonstrate the ordering of nodes on the map. Nodes from other domains are not plotted to avoid cluttering of the data displayed.

The network displayed in Fig. 7 showed that the network was not always able to build distinct clusters according to information stored in the data label. The reason for this can be twofold. One reason might be that the network has been chosen too small (only 9918 neurons versus 29 808 different input nodes) so that there was not enough space to map data according to both the structure and the data label. Another reason may be the wrong choice of the weighting parameters $\mu_i$. The following two experiments are to bring more light into this. First, networks of different sizes were trained. The initial neighborhood radius was set to the maximum extension of a map while all other learning parameters remained unchanged. The result shown in Fig. 10 is that the network performance can increase significantly when more neurons are used. However, a performance level of 100% is never achieved. In addition, the level of performance increase is getting weaker as the network grows. As a result, a network of size $114 \times 87$ seems an appropriate choice when considering that the network performance increases only slightly for networks larger than this but at the cost of increased computational demand. Also, when choosing large networks, we lose the advantage of SOM to perform data compression on large amount of data. It is interesting to note that even for small networks not all neurons are activated by nodes in the training or test data set. Only for extremely small networks that featured just one neuron for every 300 nodes in the data set that all neurons were utilized.

The second set of experiments is to determine the influence of the weighting parameters $\mu_1$ and $\mu_2$ on the network performance. For this, the ratio $\mu_1/\mu_2$ was varied within $[0;\infty]$. The neighborhood spread $\sigma$ was set to 114, and other training parameters are left the same as those used in the initial experiment and the network size remained at $114 \times 87$. The result is illustrated in Fig. 11.

The two weighting values balanced the vector components at best at a ratio 1:1.9 as stated earlier. However, better results are obtained when stressing the influence of structural information ($\mu_2$) considerably stronger than the data label. The experiment shows that with $\mu_2$ chosen 100 times larger than $\mu_1$, the network performs at best. This result is not surprising when considering that vital information is often given in nodes far away from the root node (i.e., only the data label associated with leaf nodes distinguishes between policemen with a raised or lowered arm). Thus, it is essential, that information derived from a node's offsprings is passed to parent nodes accurately. This can be achieved by choosing large ratio of $\mu_2/\mu_1$.

An additional finding was that when choosing the optimal configuration, only 4234 neurons are activated by nodes in the training set. Hence, only 42.69% of neurons on the map are used compared to 51.07% when having $\mu_2 = 1.9$. This shows that the representation of the nodes is more compact, and explains why the generalization performance on the validation set is improved.

This experiment has shown another very interesting result. Note that we trained a network with $\mu_2$ set to zero. In this case the algorithm reduces itself to the standard SOM. Another network was trained with $\mu_1$ set to zero in which case only struc-

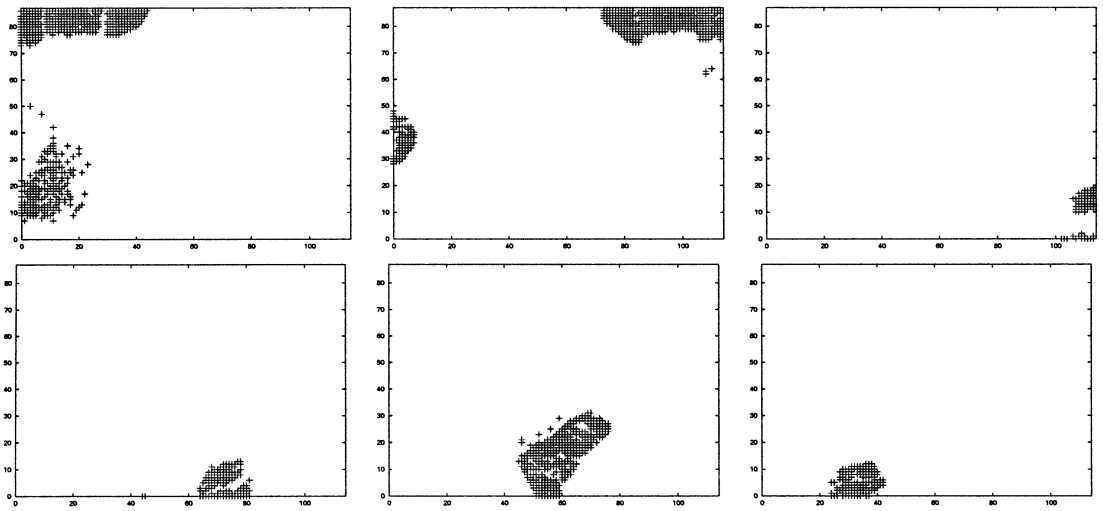

Fig. 9. Nodes from policemen patterns sorted by the distance from the leaf nodes. The upper left plot shows neurons as activated by the leaf nodes (distance 0), the next plot to the right are nodes at distance 1, and so on. The lower right plot shows the root nodes located at distance 5. Note that the network has been trained on the entire dataset and is the same shown as in Fig. 7 . Nodes belonging to a class other than policemen are not displayed here.

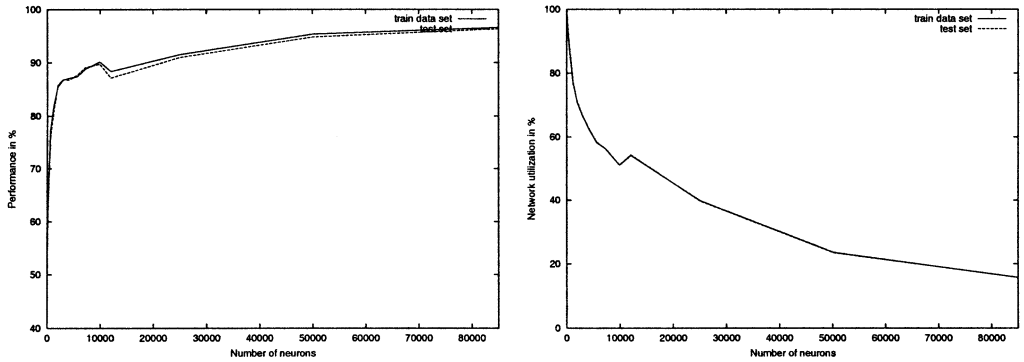


Fig. 10. Performance of the SOM-SD when varying the size of the network. The left plot illustrates the overall performance, the right plot gives the amount of neurons activated by at least one node.

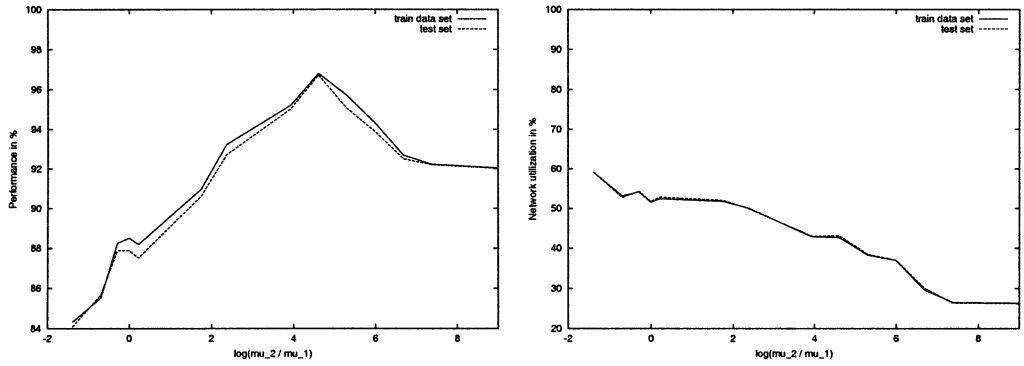


Fig. 11. Performance of the SOM-SD when varying $\mu_1$ and $\mu_2$. The network performance is shown on the left, the right plot illustrates the network utilization in percent. The logarithm is to the base $e$.

tural information is encoded. It was found, that information encoded in the data label contributes more strongly to a good network performance ($\sim 73\%$ with $\mu_2 = 0$) than pure structural information ($\sim 67\%$ with $\mu_1 = 0$). Also, when considering the network usage rates, it is found that data compression is growing with smaller $\mu_1/\mu_2$ ratio. This shows that $\mu_2$ controls the "focus" of the SOM on features while $\mu_1$ effects the "separation" of features. However, it has become evident that only the combination of both, numerical information presented in the data label and structural information, can produce well performing SOM networks. However, it is not possible to accurately predict the best combination of values for $\mu_1$ and $\mu_2$. With Fig. 12, we illustrate the mapping of root nodes when choosing $\mu_2 = 0$ (left plot), and $\mu_1 = 0$ (right plot). With $\mu_2 = 0$, the network has particular difficulties in distinguishing the classes derived from within each domain. The three domains themselves

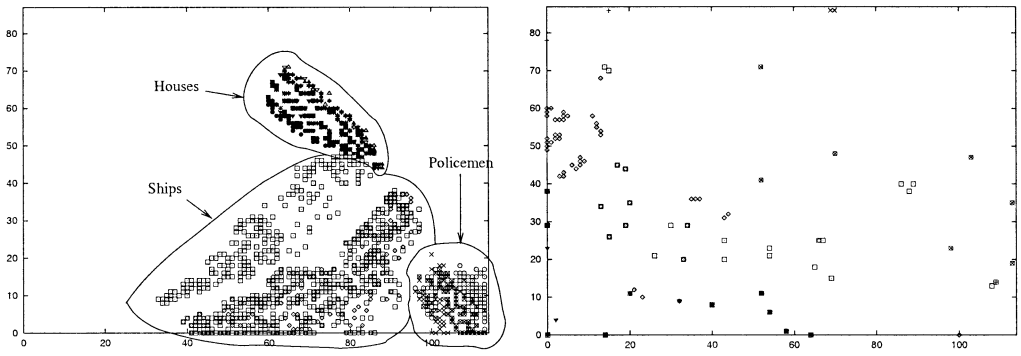


Fig. 12. Neurons as activated by root nodes when setting $\mu_2 = 0$ (left), and $\mu_1 = 0$ (right).

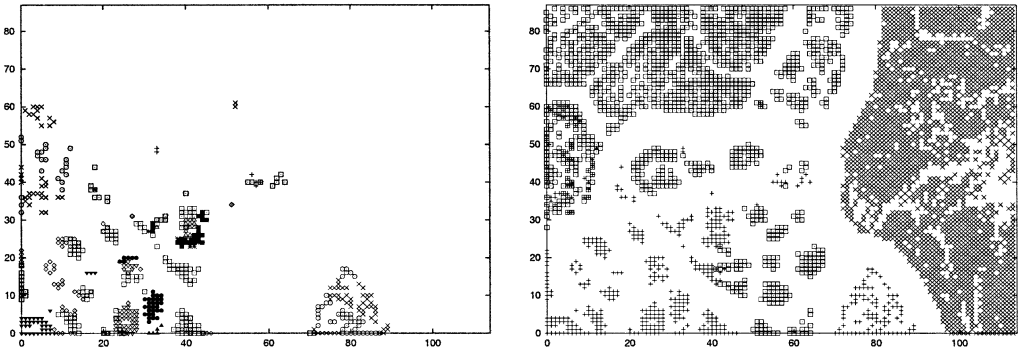

Fig. 13. Mapping of nodes when $\mu_2 \approx 100\mu_1$. This setting produced a well-performing network.

are well distinguished. This shows that classes within each domain can be distinguished only when considering the structural information as well. In contrast, the right plot in Fig. 12 shows the mapping of root nodes after training the network with $\mu_1 = 0$. With this, reasons for poor network performance become evident.

The mapping of nodes for the optimal set of $\mu_i$ is illustrated in Fig. 13. The left plot shows the mapping of nodes depending on their class memberships. It shows that there is very little overlap between nodes from different classes. The interesting observation is that the training algorithm has placed a considerable focus on structural information even when structural information itself does not help to separate the classes, e.g., the policemen classes. Here, the focus on the structure assures that vital information encoded in the data label associated with nodes away from a root node are passed back to the root accurately. The plot on the right shows the mapping of nodes depending on the type of a node. A plus shaped symbol corresponds to a root node, squares represent the mapping of intermediate nodes, and crosses show the location on which leaf nodes are mapped. It can be observed that relatively large areas are left blank. This resembles neurons which were not activated by any node in the training set, and contributes to a good generalization performance.

It has been demonstrated that SOM-SDs generalization performance is generally very good in that the performance on validation data is typically less than 1% behind the performance achieved on the training data. This gives no implications on whether this is due to data of low complexity, the sheer size of the data set, or a strength of the SOM-SD model. The following experiment varies the size of the data set to give a better insight. The quantity of training data is gradually reduced until only 61 graphs (1/64 of the original set) are left. The network is then tested against the test set where no pattern in the training set is present in the validation set. Training parameters and the size of the network are the same as the first experiment described in this section with the exception of $\sigma$ which was set to 114. The result is illustrated in Fig. 14. It is found that the network performs always around the 90% mark on the training data independent of how many patterns are present during training. In contrast, generalization performance decreases as soon as the traing set consists of less than $\sim$ 16 000 nodes. Also illustrated is the network utilization rate where the amount of neurons actually activated by any node from the training (test) set is plotted against the size of the data set. It is observed that just over 3% of neurons on the map are utilized after training a set with only 61 graphs, while the test set activates almost ten times as many neurons. This clearly shows that overfitting has occured, contributing to a poor generalization performance. With this result, it has been demonstrated that the dataset is sufficiently complex to require a large amount of training patterns to allow good generalization.

In the following, we consider the influence of the number of training iterations on the performance of a SOM-SD network. One would assume that due to the large size of the training data, only a relatively small number of iterations is required to achieve good results. To verify this, we trained some networks for just 50 iterations while others with as many as for 500 iterations. The result of these experiments is shown in Fig. 15. The figure shows that the network performance increases steadily with the

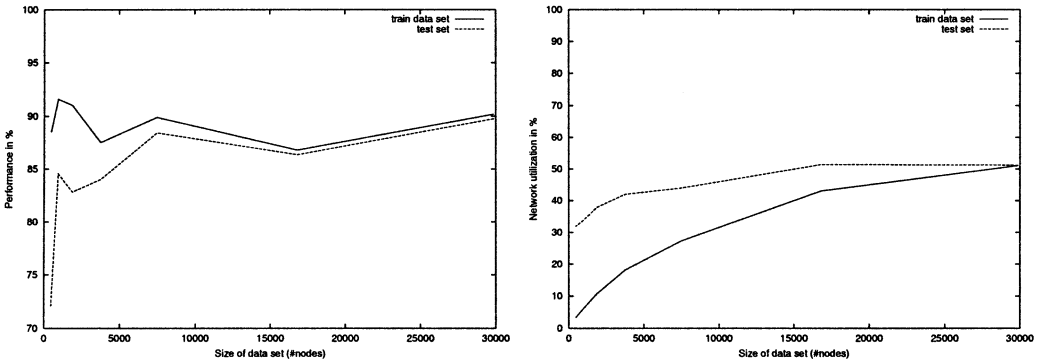


Fig. 14. Network performance versus size of the training set. The network performance is shown on the left plot. The right plot illustrates the amount of neurons activated by at least one node.

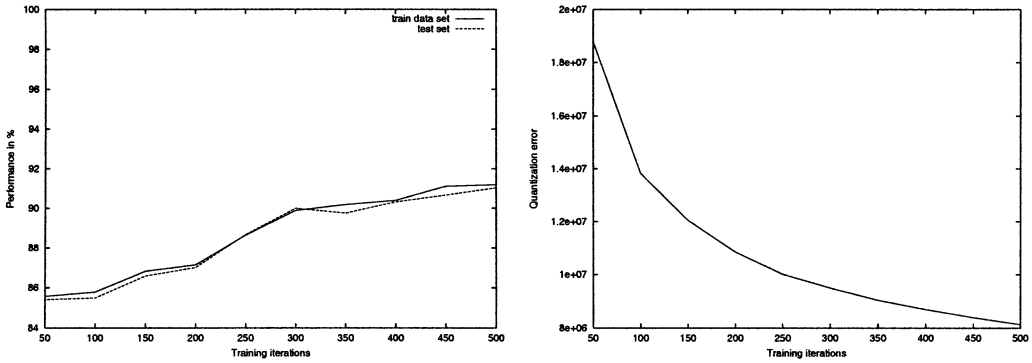


Fig. 15. Network performance versus the total number of iterations trained. The left plot shows the network performance while the right plot gives the quantization error.

number of training iterations. However, training the network for more than 350 iterations did not help to improve the performance much further. Instead, the generalization performance was observed to decrease due to the effects of "overfitting." Also shown in this figure is the quantization error as obtained at the last training iteration. The quantization error shown is the total sum over all nodes in the training set.[10] It is observed that while the network performance is at about 85% when training for only 50 iterations, the quantization error is more than twice as large when trained for 500 iterations where the increase in performance is just about 6%.

So far, most experiments were based on an initial learning parameter set to 0.08 and a neighborhood radius set to the maximum dimension of the SOM. During training, the learning parameter decreases linearly to zero while the neighborhood radius decreases linearly to one. Traditionally, the initial neighborhood radius is chosen large so as to reduce the chances of falling into a local minima situation [31]. To verify whether this is true for SOM-SD, we conducted the following experiments. Networks were trained with different initial radii, then the network performance was evaluated. From the results displayed in Fig. 16, it can be observed that SOM-SDs performance actually increases with smaller initial radius. This is in contrast with observations typically made on traditional SOM. An explanation to this behavior is that SOM-SD clusters input data into areas depending on the type of node (root, intermediate at different levels, leaf). This clustering has been observed to be established very early during training (within the first or second iteration). From then on, those clusters change in location only little whereas major updates are happening only within each cluster. Thus, it is beneficial for the network to have a small neighborhood function so that training can be focused on smaller areas of the network. Evidently, large radius does have a rather disturbing effect. This is confirmed by considering the plot showing the minimum quantization error as obtained after training. The quantization error actually increases the larger the initial radius.

The last parameter to be considered for experiments is the initial learning parameter. So far, all experiments were based on an initial learning rate starting at 0.08 which decreased linearly with the training iterations down to zero. Kohonen described that in the standard SOM case [22], too large and too small learning rates can lead to a poor network performance. There is no clear guideline which suggests good learning rates to any given learning problem. Hence, the following experiments are conceived to find the optimal learning rate for the benchmark problem under consideration. Networks of size $114 \times 87$ were trained with a range of different initial learning parameters $\eta(0)$. All other parameters were left the same as the first experiment described in this section. Results are illustrated in Fig. 17.

It is found, that the SOM-SD can be trained using a wide range of initial learning rates. Any initial learning rate larger than 0.32 and smaller than 1.5 can produce good results. Learning rates larger than two produced data overflow errors,

[10]This is the quantization error for the nodes of the structure as opposed to the quantization error for structures which is not shown.

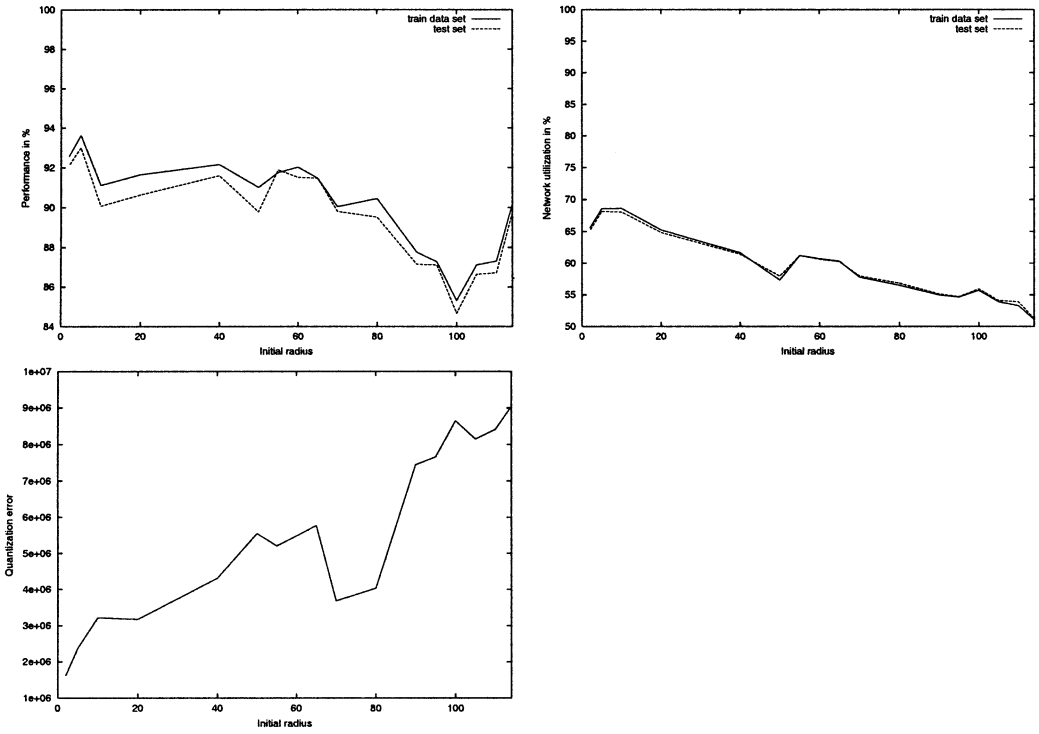


Fig. 16. Network performance versus initial neighborhood radii. The upper left plot visualizes the network performance. The amount of neurons utilized in the final mapping is shown in the upper right plot. The plot to the left illustrates the quantization error.

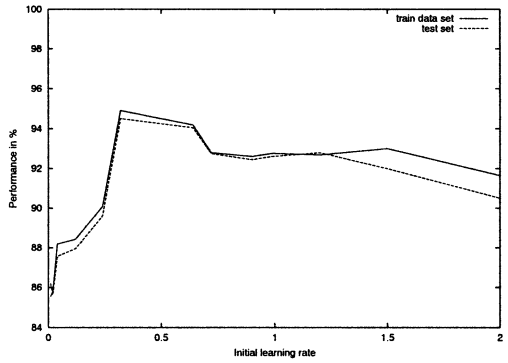


Fig. 17. Network performance versus the initial learning parameter. The plot to the right gives an overview over the network performance. The table to the left givens an indication to how $\eta(0)$ influences network utilization and performance. Note that for $\eta(0) \geq 3$ produced data overflow errors.

rates smaller than 0.32 were insufficient to make the SOM-SD to converge to high performance levels. The ability of this model to accept a large range of $\eta(0)$ together with the observation that a local minimum situation was never observed in any of the 106 training sessions conducted for this paper indicates that this model is more robust than the original SOM. This may partially be due to the use of the stochastic training method which adds some "noise" to the learning process if the data is structured.

By putting all results described in this section together, we find that for the given learning problem, and for a network of size $114 \times 87$ (number of neurons approximately 1/3 the number of nodes in the training set), the best set of parameters is $\sigma \approx 60$, $\eta(0) \in [0.32; 1.5]$, $\mu_2 \approx 100\mu_1$, and the number of training iterations is greater than 450.

A network trained with this set of parameters produced the following results: 97.63% performance on the training set, 97.33% on the validation set. 54.19% network utilization on the training set, and 54.40% utilization on the test set. The quantization error was $90.9 \times 10^6$. This indeed is the best result, no better performance was observed by any other experiments conducted so far. Nevertheless, there is still an error of about 2.5%. It was observed, that the greatest contributors to the network error are patterns belonging to classes that are represented by a small number of samples. Also some ships and policemen patterns contributed to the total error. It was found that these misclassifications were caused by the fact that the patterns actually shared an identical structural representation with some patterns from another class. Evidently, information provided by the data label was insufficient to produce a better mapping.

Obviously, for other problems these parameters will not necessarily be optimal. However, by showing this set of parameters, it is hoped that this gives a good indication of what to expect in other practical problems. Perhaps this set of parameters could be used as an initial set of parameters for other practical problems.

It is observed that the results for the test set are often very similar to those obtained from the training set. This is due to the number of training samples in the data set which has been chosen to be large in order to demonstrate the ability of SOM-SD to handle large amount of data efficiently. Hence, the training data cover the input space quite well. This is a good result when considering that some classes are represented poorly (e.g., class "e" is represented by just 28 training samples, whereas class "c" has 937 samples.

## VII. Conclusion

In this paper, we have described a new way of addressing unsupervised learning problems involving structured objects using an SOM technique. Specifically, we showed how an SOM network can be used recursively in order to learn a map defined on DAGs. The major innovation is to treat each leaf node as a standard input for an SOM, and the coordinates of the winning neuron as pointer within a parent node. In other words, we assume that the information transferred between the children and the parent is via the coordinates of the winning neuron in the child node.

We demonstrated the capabilities of the model by utilizing a relatively large data set taken from a recognized benchmark problem involving visual patterns encoded as labeled DAGs. The experimental results clearly show that the model is able to encode structural features in a coherent way. The information on the labels, on the other hand, can be encoded finely only if the map is sufficiently large and training parameters are chosen appropriately. In addition, it has been shown that the performance of SOM is improved through the incorporation of structural information. The fact, with SOM-SD, that smaller initial learning radius can be utilized opens the door for optimizations.

It has become clear that the model described here is in fact able to build a two-dimensional metric on the structured input space. The abilities of this model allow the application to new domains, involving data mining and knowledge discovery, which were not possible to directly attack in the past.

As a final remark on the computational efficiency of the proposed model, it must be stressed that recent techniques based on random projections [33], could be used in the future to speed up the selection of the winning neuron.

## References


[1] A. Sperduti, D. Majidi, and A. Starita, "Extended cascade-correlation for syntactic and structural pattern recognition," in *Advances in Structural and Syntactical Pattern Recognition*, P. Perner, P. Wang, and A. Rosenfeld, Eds. New York: Springer-Verlag, 1996, vol. 1121, Lecture Notes in Computer Science, pp. 90–99.

[2] C. Goller and A. Küchler, "Learning task-dependent distributed structure-representations by backpropagation through structure," *Proc. IEEE Int. Conf. Neural Networks*, pp. 347–352, 1996.

[3] A. Sperduti and A. Starita, "Supervised neural networks for the classification of structures," *IEEE Trans. Neural Networks*, vol. 8, pp. 714–735, May 1997.

[4] P. Frasconi, M. Gori, and A. Sperduti, "A general framework for adaptive processing of data structures," *IEEE Trans. Neural Networks*, vol. 9, p. 768, Sept. 1998.

[5] J. L. Elman, "Finding structure in time," Center for Research in Language, Univ. California at San Diego, La Jolla, CRL Tech. Rep. 8801, 1988.

[6] ——, "Finding structure in time," *Cogn. Sci.*, vol. 14, pp. 179–211, 1990.

[7] A. Sperduti, "On the computational power of recurrent neural networks for structures," *Neural Networks*, vol. 10, no. 3, pp. 395–400, 1997.

[8] P. Frasconi, M. Gori, and A. Sperduti, "On the efficient classification of data structures by neural networks," *Proc. Int. Joint Conf. Artificial Intelligence*, pp. 1066–1071, 1997.

[9] M. Gori, A. Kuchler, and A. Sperduti, "On the implementation of frontier-to-root tree automata in recursive neural networks," *IEEE Trans. Neural Networks*, vol. 10, pp. 1305–1314, Nov. 1999.

[10] B. Hammer, "On the learnability of recursive data," *Math. Contr. Signals Syst.*, vol. 12, pp. 62–79, 1999.

[11] P. Frasconi, M. Gori, and A. Sperduti, "Learning efficiently with neural networks: A theoretical comparison between structured and flat representations," *Proc. Eur. Conf. Artificial Intelligence*, pp. 301–305, 200.

[12] B. Hammer, "Generalization ability of folding networks," *IEEE Trans. Knowledge Data Eng.*, vol. 13, no. 2, pp. 196–206, 2001.

[13] R. C. Carrasco and M. L. Forcada, "Simple strategies to encode tree automata in sigmoid recursive neural networks," *IEEE Trans. Knowledge Data Eng.*, vol. 13, no. 2, pp. 148–156, 2001.

[14] C. Goller, "A Connectionist Approach for Learning Search-Control Heuristics for Automated Deduction Systems," Ph.D. dissertation, Dept. Computer Science, Tech. Univ. Munich, Munich, Germany, 1997.

[15] E. Francesconi, P. Frasconi, M. Gori, S. Marinai, J. Q. Sheng, G. Soda, and A. Sperduti, "Logo recognition by recursive neural networks," in *Graphics Recognition—Algorithms and Systems*, R. Kasturi and K. Tombre, Eds. New York: Springer-Verlag, 1997, vol. 1389, Lecture Notes in Computer Science, pp. 104–117.

[16] M. Diligenti, M. Gori, M. Maggini, and E. Martinelli, "Adaptive graphical pattern recognition: The joint role of structure and learning," in *Proc. Int. Conf. Advances Pattern Recognition*, Plymouth, UK, Nov. 1998, pp. 425–432.

[17] F. Costa, P. Frasconi, V. Lombardo, and G. Soda, "Toward incremental parsing of natural language using recursive neural networks," *Appl. Intell.*, to be published.

[18] T. Schmitt and C. Goller, *Relating Chemical Structure to Activity With the Structure Processing Neural Folding Architecture*, 1998.

[19] A. M. Bianucci, A. Micheli, A. Sperduti, and A. Starita, "Application of cascade correlation networks for structures to chemistry," *Appl. Intell.*, vol. 12, pp. 115–145, 2000.

[20] ——, "Analysis of the internal representations developed by neural networks for structures applied to quantitative structure-activity relationship studies of benzodiazepines," *J. Chem. Inform. Comput. Sci.*, vol. 41, no. 1, pp. 202–218, 2001.

[21] A. Micheli, A. Starita, A. Sperduti, and A. M. Bianucci, "Design of new biologically active molecules by recursive neural networks," *Proc. Int. Joint Conf. Neural Networks*, vol. 4, pp. 2732–2737, 2001.

[22] T. Kohonen, *Self-Organization and Associative Memory*, 3rd ed. New York: Springer-VERLAG, 1990.

[23] M. van Hulle, Ed., *Faithful Representations and Topographic Maps*. New York: Wiley, 2000.

[24] T. Villmann, R. Der, M. Herrmann, and T. M. Martinetz, "Topology preservation in selforganizing feature maps: Exact definition and measurement," *IEEE Trans. Neural Networks*, vol. 8, pp. 256–266, Mar. 1997.

[25] G. Chappell and J. Taylor, "The temporal Kohonen map," *Neural Networks*, vol. 6, pp. 441–445, 1993.

[26] M. Varsta, J. Del, R. Millan, and J. Heikkonen, "A recurrent self-organizing map for temporal sequence processing," in *Proc. 7th Int. Conf. Artificial Neural Networks*, 1997, pp. 421–426.

[27] T. Voegtlin, "Context quantization and contextual self-organizing maps," *Proc. Int. Joint Conf. Neural Networks*, vol. VI, pp. 20–25, 2000.

[28] C. Goller, M. Gori, and M. Maggini, "Feature extraction from data structures with unsupervised recursive neural networks," in *Proc. Int. Joint Conf. Neural Networks*, Washington, DC, July 1999, pp. 425–432.

[29] A. Sperduti, "A tutorial on neurocomputing of structures," in *Knowledge-Based Neurocomputing*, I. Cloete and J. M. Zurada, Eds. Cambridge, MA: MIT Press, 2000, pp. 117–152.

[30] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 1990.

[31] T. Kohonen, *Self-Organizing Maps*. Berlin, Germany: Springer, 1995, vol. 30, Springer Series in Information Sciences.

[32] M. Hagenbuchner, A. C. Tsoi, and A. Sperduti, "A supervised self-organizing map for structured data," in *Advances in Self-Organizing Maps*, N. Allinson, H. Yin, L. Allinson, and J. Slack, Eds., 2001, pp. 21–28.

[33] P. Indyk and R. Motwani, "Approximate nearest neighbors: Toward removing the curse of dimensionality," in *Proc. 30th Symp. Theory Computing*, 1998, pp. 604–613.


**Markus Hagenbuchner** was born in Germany. He received the Bachelor degree with honors in computer science and the Ph.D. degree in informatics from the University of Wollongong, Australia, in 1997 and 2002, respectively.

He was a Research Assistant with the Department of Artificial Neural Computing at the University of Ulm, Germany, from 1997 to 1998. His research interest include neural networks, processing and classification of structural data, and data mining. He is currently a Research Associate with the University of Wollongong.

**Alessandro Sperduti** (M'98) received the Laurea and Doctoral degrees in computer science from the University of Pisa, Pisa, Italy, in 1988 and 1993, respectively.

In 1993, he spent a period at the International Computer Science Institute, Berkeley, CA, supported by a postdoctoral fellowship. In 1994, he moved back to the Computer Science Department, University of Pisa, where he was Assistant Professor and Associate Professor. Currently, he is Full Professor with the Department of Pure and Applied Mathematics, University of Padova, Padova, Italy. His research interests include pattern recognition, image processing, neural networks, and hybrid systems. In the field of hybrid systems his work has focused on the integration of symbolic and connectionist systems. He is the author of 90 refereed papers mainly in the areas of neural networks, fuzzy systems, pattern recognition, and image processing.

Dr. Sperduti contributed to the organization of several workshops on this subject and he also served on the program committee of several conferences on neural networks and machine learning.He gave several tutorials within international schools and conferences, such as IJCAI '97, IJCAI '99, IJCAI '01. He was Guest Coeditor of the IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING for a special issue on connectionist models for learning in structured domains. He is a member of the Executive Board of the Italian Neural Network Society (SIREN) and the European Neural Networks Council.

**Ah Chung Tsoi** (S'70–M'72–SM'90) was born in Hong Kong. He received the Higher Diploma degree in electronic engineering from Hong Kong Technical College in 1969, and the M.Sc. degree in electronic control engineering, and Ph.D. degree in control engineering, both from University of Salford, Salford, U.K., in 1970 and 1972, respectively. He also received the B.D. degree from University of Otago in 1980.

Dr. Tsoi received the Outstanding Alumni Award, Hong Kong Polytechnic University in 2001.