PERGAMON

Neural Networks 14 (2001) 505–525

Neural Networks

www.elsevier.com/locate/neunet

Contributed article

S-TREE: self-organizing trees for data clustering and online vector quantization

Marcos M. Campos, Gail A. Carpenter*

Center for Adaptive Systems, Department of Cognitive and Neural Systems, Boston University, 677 Beacon Street, Boston, MA 02215, USA

Received 6 December 2000; accepted 6 December 2000

Abstract

This paper introduces S-TREE (Self-Organizing Tree), a family of models that use unsupervised learning to construct hierarchical representations of data and online tree-structured vector quantizers. The S-TREE1 model, which features a new tree-building algorithm, can be implemented with various cost functions. An alternative implementation, S-TREE2, which uses a new double-path search procedure, is also developed. The performance of the S-TREE algorithms is illustrated with data clustering and vector quantization examples, including a Gauss–Markov source benchmark and an image compression application. S-TREE performance on these tasks is compared with the standard tree-structured vector quantizer (TSVQ) and the generalized Lloyd algorithm (GLA). The image reconstruction quality with S-TREE2 approaches that of GLA while taking less than 10% of computer time. S-TREE1 and S-TREE2 also compare favorably with the standard TSVQ in both the time needed to create the codebook and the quality of image reconstruction.

Keywords: Hierarchical clustering; Online vector quantization; Competitive learning; Online learning; Neural trees; Neural networks; Image reconstruction; Image compression

1. Introduction: clustering and decision trees

Data clustering is a technique used by both artificial and biological systems for diverse tasks such as vision and speech processing, data transmission and storage, and classification. Clustering can be defined as partitioning a dataset into subsets, or clusters, where the number of subsets and the grouping criteria depend on the application. Some applications seek 'natural' groups, while others try to represent hierarchical structure in the data (hierarchical clustering). Other goals include summarizing the data while preserving essential information as completely as possible. Fig. 1 illustrates hierarchical clustering and data summarization for a simple dataset with four natural clusters. A review of the clustering problem can be found in Duda and Hart (1973).

In situations where knowledge of the data distribution is available, a Bayesian or maximum likelihood approach may solve the clustering problem by estimating parameters of a distribution (Duda & Hart, 1973). When this knowledge is not available, clustering can be cast as an optimization problem by specifying a suitable cost function to be minimized. A common choice of cost function is the sum of squared distances from points in a cluster to the cluster's center. There are many procedures in the literature for choosing cost functions for clustering problems. Some of the most prominent are: the ISODATA algorithm (Ball & Hall, 1967), the *K*-mean algorithm (MacQueen, 1967), the generalized Lloyd vector quantization algorithm (Linde, Buzo & Gray, 1980), and fuzzy clustering methods (Dunn, 1974; Bezdek, 1980). These procedures share a number of limitations, including sensitivity to initial conditions and poor performance with datasets that contain overlapping clusters or variability in cluster shapes, densities, and sizes. These are also unstructured clustering methods, with no structural constraint imposed on the solution. Because unstructured methods require an exhaustive search for the nearest cluster, this approach typically becomes impractical for large feature spaces or many clusters.

In order to overcome the computational burden associated with unconstrained clustering, structural constraints such as lattices and trees have been proposed (see Gersho & Gray, 1992, for a review). In particular, tree-structured clustering methods have become popular in the vector quantization literature. *Binary trees* construct prototype vectors (weight vectors) at each node (Fig. 2), and nodes are traversed according to a nearest-neighbor algorithm and a given distance measure (Fig. 3). For each node, starting at

* Corresponding author. Tel.: +1-617-353-9483; fax: +1-617-353-7755.
E-mail address: gail@cns.bu.edu (G.A. Carpenter).

0893-6080/01/$ - see front matter
PII: S0893-6080(01)00020-X

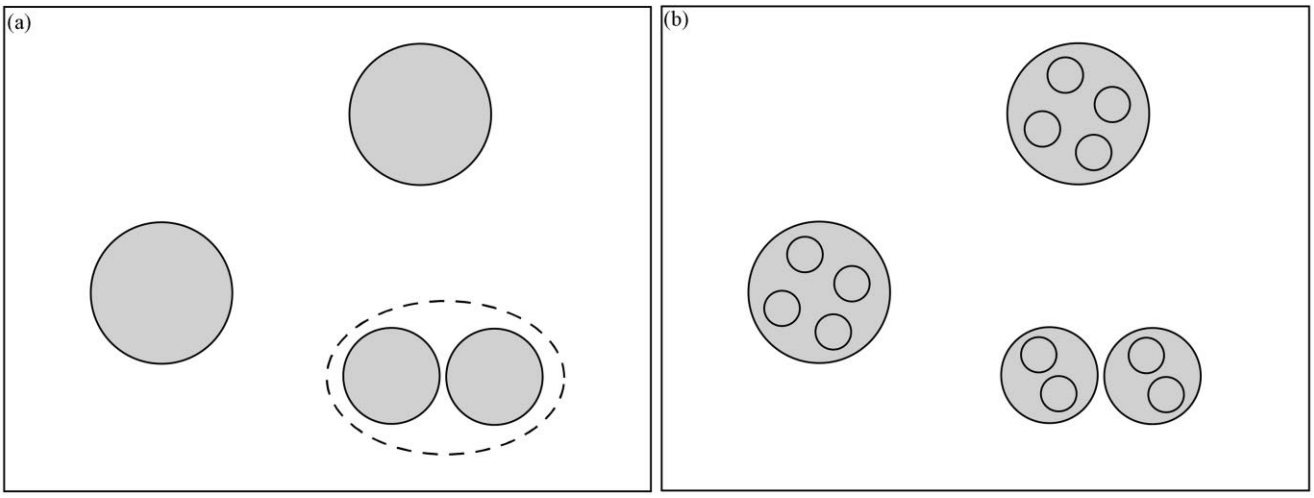


Fig. 1. In the unsupervised clustering problem, with no class labels, data points can be grouped according to the relationships they have among themselves. In this example the gray regions represent areas where data points are located in four natural clusters. (a) A hierarchical clustering application would identify three large clusters, and subdivide one of them (dashed line) into two clusters. (b) A typical solution for a data summarization application trying to group the data into 12 compact clusters (small circles).

the root node, an input vector is compared to the prototypes of the two child nodes of the current node. The child node with the nearest neighbor, or closest prototype, to the input vector is selected. The algorithm repeats the same procedure with the new selected node until a leaf (terminal) node is selected.

Because of their local decision-making procedures, tree-structured clustering methods are globally suboptimal, and the algorithm might not select the leaf closest to the input. However, tree-structured algorithms are fast, scale well (in processing time) with the number of feature dimensions and clusters, and can capture hierarchical structures in the data.

A *balanced tree* is grown one level at a time with all nodes in a level split at once. *Unbalanced trees* can be obtained either by growing a balanced tree and then pruning using the generalized BFOS algorithm (Breiman, Freidman, Olshen, & Stone, 1984), or by incrementally growing an unbalanced tree directly one node at a time (Riskin & Gray, 1991). Although unbalanced trees take longer to build, they are more flexible, and in general yield better results in vector quantization and clustering applications than balanced trees.

This paper introduces S-TREE (*Self-Organizing Tree*), a family of models that construct hierarchical representations of data. S-TREE models solve the clustering problem by imposing tree-structured constraints on the solution. The S-TREE1 model, which features a new tree building algorithm, can be implemented online and used in conjunction with various cost functions. An alternative implementation, S-TREE2, which uses a new double-path search procedure, is also developed. S-TREE2 implements an online procedure which approximates an optimal (unstructured) clustering solution while imposing a tree-structured constraint. Because of their online nature, S-TREE models have smaller memory requirements than traditional offline methods. They are also fast, relatively insensitive to the initialization of cluster centers, and, in the case of S-TREE2, approach the performance of unconstrained methods while requiring a fraction of the computer time of those methods.

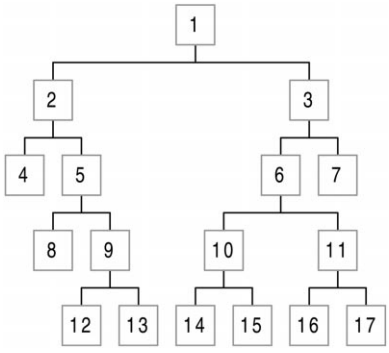


Fig. 2. Binary tree with 17 nodes. Nodes 4, 7, 8, 12, 13, 14, 15, 16, and 17 are leaf (terminal nodes). The remaining nodes are inner nodes. Node 1 is the root node.

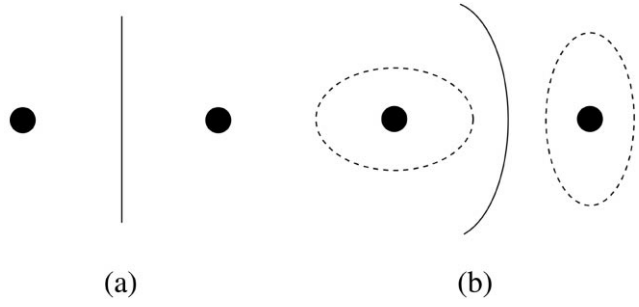


Fig. 3. The shape of the boundary, in input space, between two nodes depends upon the distance measure used. (a) Euclidean distance. (b) Weighted Euclidean distance using the inverse of the variance along each dimension as the weighting factor for each node. The dotted lines represent, for each node, the variance along the two dimensions.

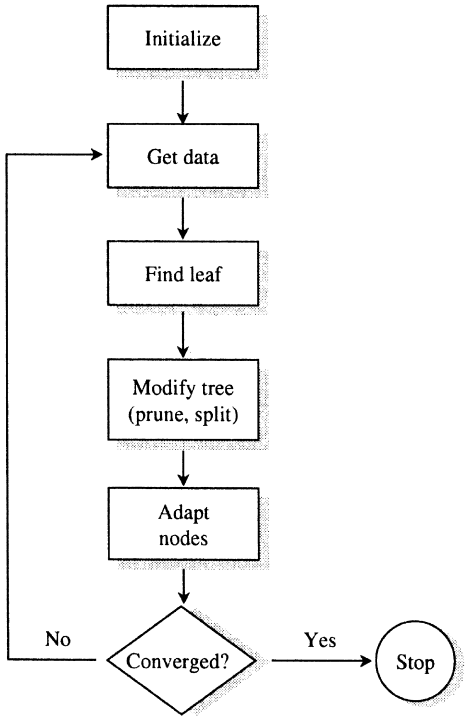


Fig. 4. Main steps of the S-TREE algorithms.

The paper is organized in two parts. Sections 6–8 describe the S-TREE1 and S-TREE2 algorithms and illustrate applications to clustering. Sections 6–8 discuss online vector quantization, with algorithm performance tested on a Gauss–Markov source benchmark and an image compression application.

2. The S-TREE1 algorithm

The S-TREE family of tree-structured clustering algorithms adapt their weight vectors via online incremental learning. Fig. 4 illustrates the main steps of the algorithm, which is specified in Appendix A and available on the web (http://cns.bu.edu/~gail/stree). S-TREE divides the input space into a nested set of regions and assigns a prototype weight vector to the data that fall into each region. This nested structure implements a tree. Each node j in the tree has an associated weight vector $\mathbf{w}_j$, a counter N_j (the number of times the node has been updated), and a cost measure e_j. The algorithm also uses a splitting threshold E to track the average cost associated with the winning leaf nodes.

The tree is initialized to a single root node. With each input vector $\mathbf{A}$ the tree is traversed via single-path search (S-TREE1) or double-path search (S-TREE2) until a leaf node is reached. S-TREE1 searches in the traditional fashion, at each internal node comparing the input vector to the prototypes of the two child nodes and selecting the child node whose weight vector $\mathbf{w}_j$ is closest to $\mathbf{A}$. After a leaf node has been found, the algorithm performs a test to decide whether it should modify the tree structure by splitting a node, if the distortion at the winning leaf node is too great; and also by pruning extra nodes, if the tree has reached its maximum size. Following the tree modification step, the weight vectors of the nodes in the path connecting the root node to the winning leaf are adapted to reflect the current input vector. The system checks convergence by calculating the total distortion C across a window of T inputs. Training stops when C remains nearly unchanged from one window to the next. Otherwise, a new input is read and the process is repeated.

During testing, the tree is traversed until a leaf node is found, and the input is assigned to the cluster labeled by that leaf. The associated cost for that input vector is computed using the weight vector of the winning leaf.

2.1. Adapting nodes

During training with an input vector $\mathbf{A}$, the accumulated cost e_j, the counter N_j, and the weight vector $\mathbf{w}_j$ are updated for each node j in the path connecting the root node, where $j = 1$, to the winning leaf, where $j = J$, according to:

$$\Delta e_j = \epsilon \tag{1}$$

$$\Delta N_j = 1 \tag{2}$$

$$\Delta \mathbf{w}_j = \left(\mathbf{A} - \mathbf{w}_j\right)/N_j \tag{3}$$

In Eq. (1), ϵ is the value of the cost measure for the current input vector, which in most applications tracks the square distance from $\mathbf{A}$ to the winning weight vector $\mathbf{w}_J$ (Section 2.4). The splitting threshold E is also updated according to:

$$\Delta E = \beta_1(e_J - E)$$

Splitting and pruning may occur only when the cost e_J of the winning leaf is greater than E. If this condition is met, E is also increased to γE.

S-TREE uses a competitive learning approach to update the tree weight vectors (3), with only one node at each level updated for a given input vector. In particular, at most one sibling in each pair is adapted for a given input. As a result, the weight vectors of each sibling pair tend to align themselves along the first principal component of the data assigned to their parent node. In the case of the sum-of-squared-distances cost function, this alignment implicitly defines a partition of the data by a hyperplane perpendicular to the principal component (Fig. 5), which usually yields good results in vector quantization tasks (Fränti, Kaukoranta, & Nevalainen, 1997; Landelius, 1993; Wu & Zhang, 1991). S-TREE approximates a principal component partition without needing to store a covariance matrix or compute eigenvectors, as is required by the related approaches.

2.2. Growing the tree

S-TREE begins with a single root node and grows a tree

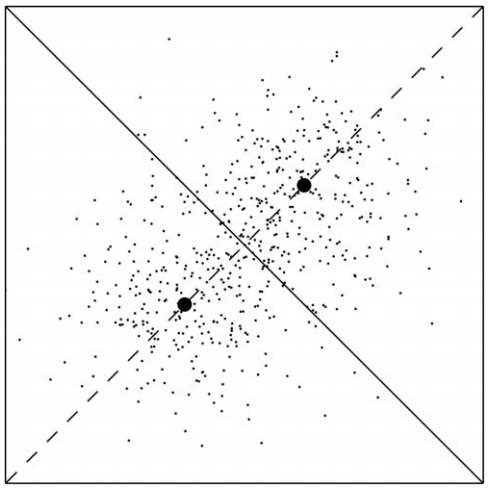


Fig. 5. Sibling nodes partition the space by a hyperplane (solid line) perpendicular to the direction of the first principal component of the data (dashed line). The filled circles (●) represent node weight vectors.

by adding either no nodes or two nodes per input, until a maximum number of nodes (U) is reached. Thereafter, the system prunes two nodes before adding each new pair (Section 2.3). The algorithm adds a new pair, or *splits*, when the cost e_J for the winning leaf is greater than the splitting threshold E. The two new child nodes are initialized as follows:

- The left child weight vector is set to $\mathbf{w}_J$ and the right child weight vector to $(1 + \delta)\ \mathbf{w}_J$, where δ is a small positive constant
- The counter N_j for each child is set to 1
- The cost variable e_j for each child is set to $e_J/2$

After a split, the maximum index u of the tree nodes is increased by 2.

In contrast to other decision tree methods (e.g. Cosman, Perlmutter, & Perlmutter, 1995; Held & Buhmann, 1998; Hoffmann & Buhmann, 1995; Landelius, 1993; Riskin & Gray, 1991), the S-TREE splitting procedure does not require a priori specification of how often nodes should be split. It also does not need a full search among all leaves to determine which one to split.

2.3. Pruning

S-TREE grows the tree in a greedy fashion. At every splitting step it tries to split the node with the largest accumulated distortion, but, because S-TREE is an online algorithm, the choice is not necessarily optimal. A pruning mechanism reduces the effect of bad splits.

S-TREE pruning is implemented as a complementary process to splitting. If the cost of the winning leaf J is found to be too large and if the number of nodes in the tree already equals the maximum U, then pruning is engaged. The idea behind pruning is to remove nodes from regions with the *least* cost to make room for new nodes in regions where the cost is still high.

For each input for which $e_J > E$, S-TREE finds the leaf m with the smallest e_j. If the cost e_m is sufficiently small compared to e_J (that is, if $e_m \leq \Gamma e_J$, where $\Gamma < 1$ is a pruning threshold) then m and one nearby node are removed from the low-cost region, and two new children are added near the high-cost region represented by J. There are three cases to consider for pruning.

- Type I: Node m's sibling is not a leaf (Fig. 6a).
- Type II: Node m's sibling is leaf J (Fig. 6b).
- Type III: Node m's sibling is a leaf other than J (Fig. 6c).

For Type I, m and its parent are removed from the tree, m's sibling takes the place of its parent, and J is split. For Type II, both m and J are removed from the tree, and their parent is split. For Type III, m and its sibling are removed from the tree, J is split, and the value of e_j for m's parent is divided by 2. This last step is needed to give the parent node a chance to adapt its cost value to reflect the new structure of the tree before it becomes a likely candidate for splitting, which could otherwise send the tree into a local cycle of splitting and pruning at the same node. In all cases the cost of each new child node is set to $e_j/2$.

2.4. Cost functions

The S-TREE construction discussed in Sections 2.1–2.3 could use a variety of cost functions, depending on the application. For example, some applications seek to partition the input data in such a way that the distribution of the weight vectors $\mathbf{w}_j$ approximates the distribution of the data points $\mathbf{A}$. Ideally, then, all clusters would have the same number of data points. This goal is pursued by methods such as the Self-Organizing Map (Kohonen, 1988) and Neural Gas (Martinetz & Schulten, 1991). One way to implement this goal in S-TREE is to specify that all leaf nodes should win the competition with the same frequency. This can be accomplished using the number of times N_j that a node is updated as the accumulated cost measure e_j, setting $\epsilon = 1$ in Eq. (1). As a result, a node that wins the competition frequently would have a large accumulated cost and become a good candidate for splitting.

In other applications such as vector quantization, a common goal is to minimize the sum of squared distances:

$$D = \sum_j \sum_{\mathbf{A} \in \Lambda_j} \|\mathbf{A} - \mathbf{w}_j\|^2 \tag{4}$$

where Λ_j is the set of inputs $\mathbf{A}$ assigned to leaf j. The strategy in this case is to split, at each split opportunity, the node that contributes the most to the total distortion D. One approach would be to set the cost ϵ to $\|\mathbf{A} - \mathbf{w}_j\|^2$. However, this formula would not work well in an online setting for the following reason. Early in training, when the weights are changing rapidly, $\|\mathbf{A} - \mathbf{w}_j\|^2$ is on average

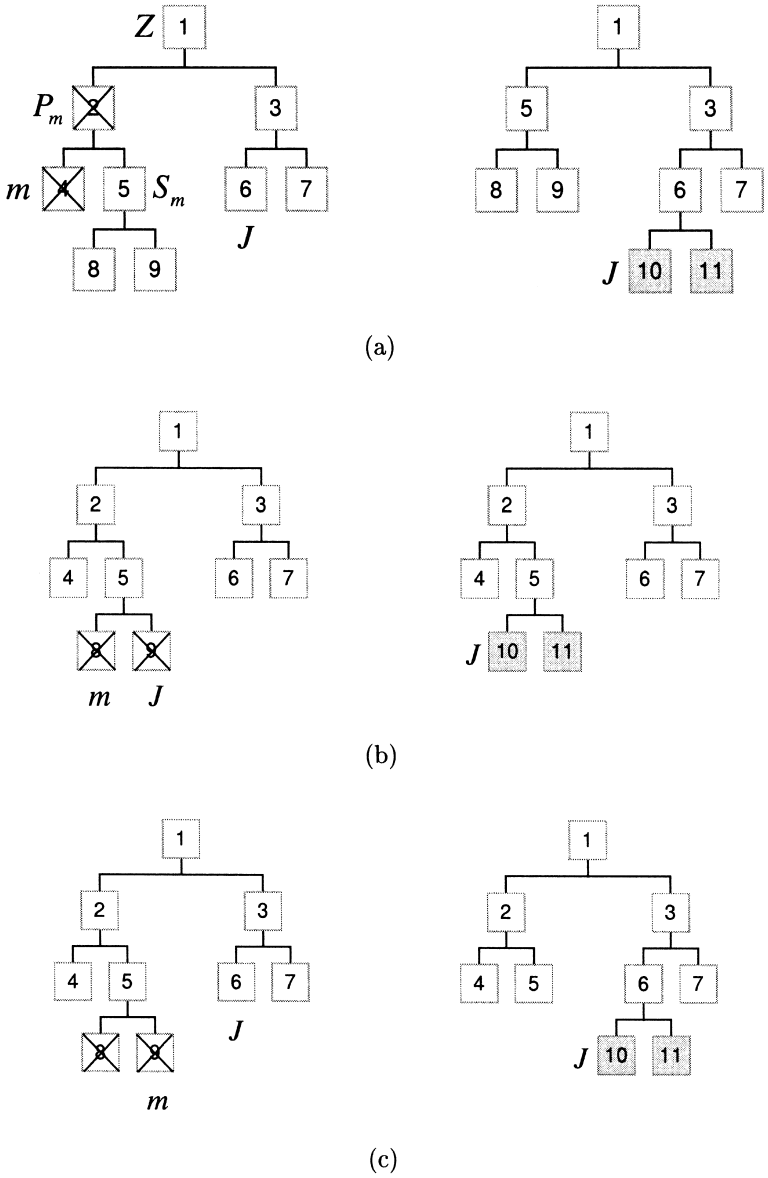


Fig. 6. Pruning cases with $U = 9$. (a) Type I: m's sibling is not a leaf. (b) Type II: m and J are siblings. (c) Type III: m's sibling is a leaf other than J. In all cases $J = 10$ and $u = 11$ after splitting. J denotes the winning leaf, m is the leaf with the smallest cost e_j, × marks deleted nodes, and gray boxes indicate nodes added by splitting.

much larger than after the weights have converged. As a result, setting ϵ to $\|\mathbf{A} - \mathbf{w}_j\|^2$ would lead e_j to be dominated by the large early error. This early factor would make the model more sensitive to initial conditions, and would also require a longer interval between splits to allow the weights to stabilize and to allow e_j to reflect the actual cost associated with each leaf node.

The solution used in S-TREE is to transform the cost from an absolute value to a relative one. Accordingly, ϵ is computed as

$$\epsilon = \epsilon_0/\bar{\epsilon}_0 \tag{5}$$

where ϵ_0 is $\|\mathbf{A} - \mathbf{w}_j\|^2$ and $\bar{\epsilon}_0$ is a fast-moving average of ϵ_0 computed using

$$\Delta\bar{\epsilon}_0 = \beta_2(\epsilon_0 - \bar{\epsilon}_0)$$

When the weights are changing rapidly, ϵ_0 is large but so is $\bar{\epsilon}_0$. Later, when the weights converge and ϵ_0 becomes smaller so does $\bar{\epsilon}_0$. This relative cost measure allows the sum e_j to reflect more readily the true cost associated with each node. Nodes with large e_j have bigger contributions to D than those with smaller e_j and are good candidates for splitting.

2.5. Convergence criterion

In batch processing a commonly used convergence

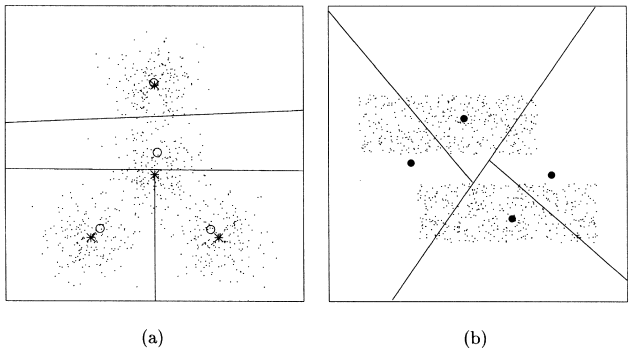


Fig. 7. Two S-TREE1 simulation examples illustrating how the tree-structured constraint may prevent the correct partition of the input space. (a) An example with a mixture of four isotropic Gaussian components. Note how the leaf weight vectors (○) have not converged to the correct cluster centers (∗). Lines represent decision boundaries. (b) A non-Gaussian data distribution has led to an uneven partition of the space, with the left-most and right-most leaf nodes (●) having much larger accumulated costs than the remaining two leaf nodes.

criterion is

$$\frac{|C_{\tau-1} - C_\tau|}{C_{\tau-1}} < \eta \tag{6}$$

where $C_{\tau-1}$ and C_τ measure system performance on the whole training set for epochs $\tau - 1$ and τ respectively, and η is a small constant.

For an online method this criterion needs to be modified, since the size of the training set is not specified in advance. S-TREE uses a window of a fixed size and computes the performance of the algorithm over consecutive windows of the training data. To compensate for fluctuations that can occur for small windows, a moving average of the performance on consecutive windows is used to check for convergence. Taking $\bar{C}_\tau$ to be the smoothed moving average performance on window τ, S-TREE's online convergence criterion is defined by

$$\frac{|\bar{C}_{\tau-1} - \bar{C}_\tau|}{\bar{C}_{\tau-1}} < \eta$$

where $\bar{C}_\tau = \bar{C}_{\tau-1} + \beta_3(C - \bar{C}_{\tau-1})$ and C is the performance on window τ.

2.6. Limitations of the S-TREE1 algorithm

S-TREE1, like other tree-structured clustering algorithms (Gersho & Gray, 1992; Held & Buhmann, 1998; Hoffmann & Buhmann, 1995), is suboptimal in the sense that the leaf node selected by the algorithm is not necessarily the one closest to the input vector. This occurs because branching at the higher levels of the tree biases the search, which may cause data points to be assigned to wrong clusters or weight vectors not to correspond to the cluster centers. Fig. 7 illustrates the impact of this structural bias for two simple clustering problems where S-TREE1 did not learn the correct centers.

3. S-TREE2: double-path search

The goal of S-TREE2 is to minimize the bias introduced by the tree-structured constraint. This version of the algorithm uses two paths to search for the leaf closest to a given input (Fig. 8). During training, for each input vector, the algorithm first selects the root node as the initial winning node. If the root node has no children, then the root is selected as the winning leaf and the search stops. Otherwise, the input vector is compared with the weight vectors of the children of two current winning nodes, and the two child nodes with weight vectors closest to the input vector are selected. If a node has no children, then the node itself is used in the competition to determine the next two closest weight vectors. The algorithm repeats the same procedure

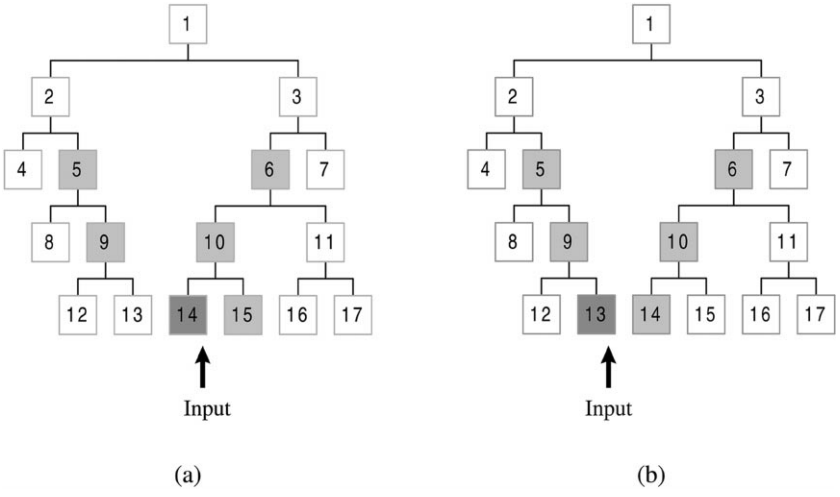


Fig. 8. Two examples using an S-TREE2 double-path search. Light gray boxes show the winning nodes for each level of the tree. Dark gray boxes indicate the final winning leaf. The arrow shows the position of the input relative to leaf weight vectors. (a) Both winning leaves (14, 15) are on the same side of the tree. (b) Winning leaves (13, 14) are on different sides of the tree.

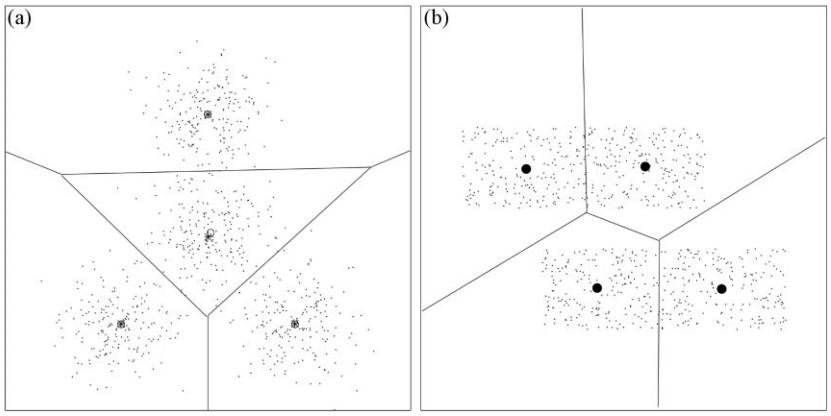


Fig. 9. Improved solutions with S-TREE2, compared with Fig. 7. In (a), ○ represents leaf weight vectors, ∗ represents the actual Gaussian centers, and lines represent decision boundaries. In (b), filled circles (●) represent leaf weight vectors.

with the newly selected nodes until both winners are leaves. Which one of the two leaves is chosen as the final winner depends on the size of the tree. After the tree has reached its maximum size (U), the selected leaf with weight vector closer to the input is chosen. If the tree is still growing, and if the two leaves are at the same depth, then the one with the closer weight vector is again chosen, but if the paths from the root node have different lengths, then the leaf with the shorter path is chosen. This constraint on the selection of the winning leaf in a growing tree enforces a type of load balancing among nodes at each level, preventing nodes that split early from seizing too many inputs. After an overall winning leaf has been chosen, the algorithm proceeds in the same fashion as the single-path search version.

In S-TREE2, the double-path search approximates the unconstrained solution, as the system adjusts the boundaries of the inner nodes to reflect the distribution of data around leaf weight vectors. After training, S-TREE2 approximates the Voronoi partition, unlike most other tree-structured clustering algorithms (Fig. 9). The change in search strategy in S-TREE2 thus adds significant power, at minor computation cost.

Table 1
S-TREE2 parameters for clustering simulations

Parameter	Value
All simulations	
E_0	50
β_1	0.01
β_2	0.075
β_3	0.2
γ	1.1
Γ	0.35
δ	0.0001
η	0.01
Eight clouds (2-D and 3-D)	
U	15
T	640
Sixteen clouds	
U	31
T	1200

Fig. 10 presents a step-by-step example of how S-TREE2 grows a tree. The dataset is a 2-D Gaussian mixture with eight isotropic components. The diagrams show the tree just before pruning and splitting take place at each step. As training proceeds, the boundaries between the tree leaves approach the Voronoi partition for the leaf distribution. The figure also shows how leaf weight vectors move towards the centroid of the region they control. In Fig. 10(g), Type III pruning is engaged to remove underutilized leaves (near the center) and to split a leaf (lower right) accounting for too many inputs. Fig. 10(h) shows the final configuration of the tree. With the exception of some minor defects, the boundaries approach the Voronoi partition for the leaf nodes, and the associated tree structure also reflects the hierarchical structure in the data.

4. Clustering examples

This section compares S-TREE2 performance with other tree-structured and unstructured clustering methods. In Figs. 11–13 the data were generated by randomly sampling 2-D (Figs. 11 and 13) and 3-D (Fig. 12) Gaussian mixture distributions with isotropic components. In Fig. 14 the data were generated by randomly sampling 16 Gaussian distributions with different shapes and densities. In all examples, each mixture component contributed 400 samples to the dataset. The parameters used in the simulations are listed in Table 1. The window size T was set to about 20% of the sample size. In applications, values for T are typically between 5 and 20% of the sample size; large datasets can use smaller values for T.

Fig. 11(a) shows that the Tree-Structured Vector Quantizer (TSVQ) algorithm (Gersho & Gray, 1992) may not do a good job of discriminating mixture components, with several leaf weight vectors missing the cluster centers. S-TREE2, on the other hand, is capable of overcoming the tree structure bias and correctly discriminating the eight mixture components (Fig. 11(b)). The decision boundaries for

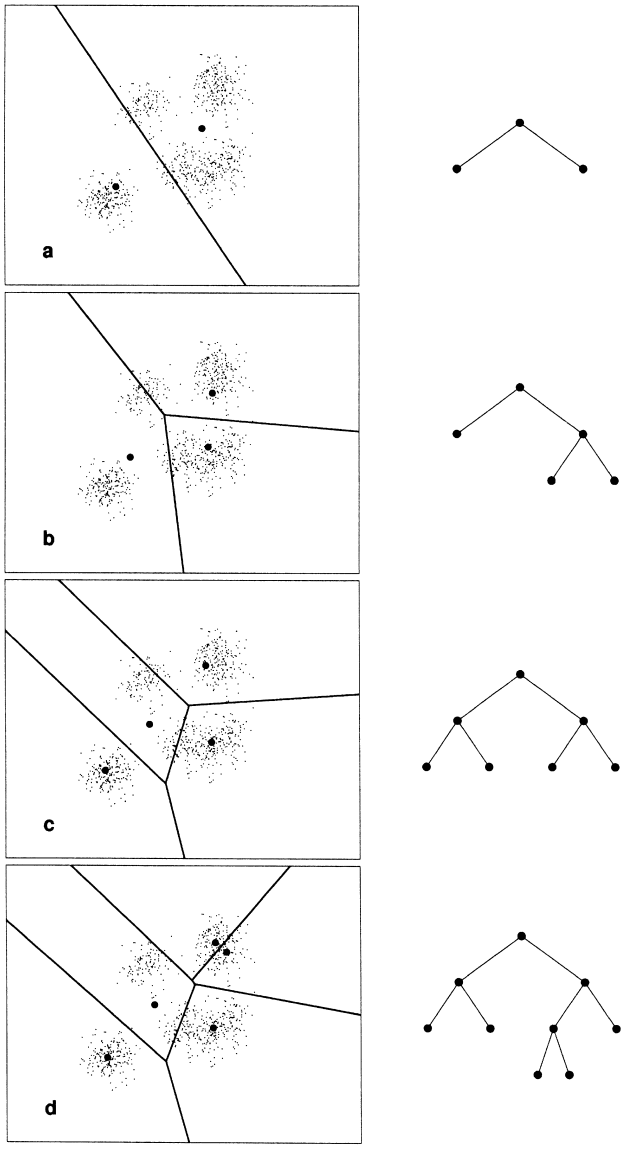


Fig. 10. (a)– (h) S-TREE2 solution for a Gaussian mixture with eight components. Each figure illustrates the tree state just before pruning and splitting takes place. Leaf weight vectors are indicated by ● and lines represent decision boundaries. (g) shows the tree before a pruning step (h) shifts leaf weight vectors toward the lower right.

S-TREE2 also approach the Voronoi partition for the same distribution of leaf weight vectors. Fig. 12 shows similar results for a 3-D example.

For the example in Fig. 13, S-TREE2 is compared with the unstructured *K*-means algorithm, as implemented by the SPSS statistical package. Even though S-TREE2 imposes a tree-structured constraint on the solution, it achieves better results than *K*-means in this example. Fig. 14 shows similar results for a mixture of Gaussians with different shapes and orientations.

Detailed comparisons between S-TREE2, TSVQ, and the unstructured generalized Lloyd algorithm (GLA) (Linde et al., 1980) are given in Section 7, including analysis of their performance on an image compression task.

5. Cluster validity and tree size

What is the proper number of clusters in a given data sample? The answer to this question depends upon the goal of the clustering task. Some applications seek to find 'natural' clusters in data, and their subclusters. Other applications seek to group, or vector-quantize, the data.

In the case of grouping, an algorithm may actually be imposing, rather than finding, a certain structure in the data. If the data are uniformly distributed, the concept of clusters does not make sense. Nevertheless, it may still be useful to group the data in smaller bunches.

The premise of natural clusters is that the dataset has some internal structure which can be used to summarize it. For example, if data points are distributed in a Gaussian cloud, the mean and the variance accurately summarize the entire data distribution. How can one find natural clusters in a dataset? How can one extract a hierarchical structure if present? In a tree-structured clustering algorithm these questions are closely related, since the identification of hierarchical structures is equivalent to recursively finding the natural clusters in a dataset. So, the key question becomes: when should a cluster be further divided?

Duda and Hart (1973) suggest a procedure for deciding upon cluster division which can be readily applied to S-TREE, has some statistical backing, and is not computationally expensive. The approach is based on the observation that, although the sum of costs (mean squared distances) after partitioning a cluster in two is always smaller than the parent's cost, the reduction in cost is greatest with true subclusters (Fig. 15). This observation, combined with some simplifying assumptions (see Duda & Hart, 1973, Chapter 6, for details), allows the construction of the following test for rejecting the null hypothesis, that there are no subclusters in the data. That is: assume there are subclusters at the *p*-percent significance level if

$$\frac{E_2}{E_1} < 1 - \frac{2}{\pi M} - \alpha\sqrt{\frac{2(1 - 8/\pi^2 M)}{NM}} \tag{7}$$

where E_1 is the cost for the parent node, E_2 is the sum of costs for the children, M is the number of dimensions of the input vector, N is the number of data points in the sample assigned to the parent node, and α is determined by

$$p = 100\int_{\alpha}^{\infty} \frac{1}{2\pi} e^{-u^2/2} du = 100(1 - \mathrm{erf}(\alpha))$$

where erf(.) is the standard *error function*.

Using this test, the following procedure can be implemented, for S-TREE, to prune spurious clusters: after training is finished check whether inequality (7) is satisfied for each inner node with two leaf children. If a node does not satisfy (7), then its children represent spurious clusters and can be pruned. Repeat this procedure until all inner nodes with two children satisfy (7). The simulations reported in this paper do not use this test.

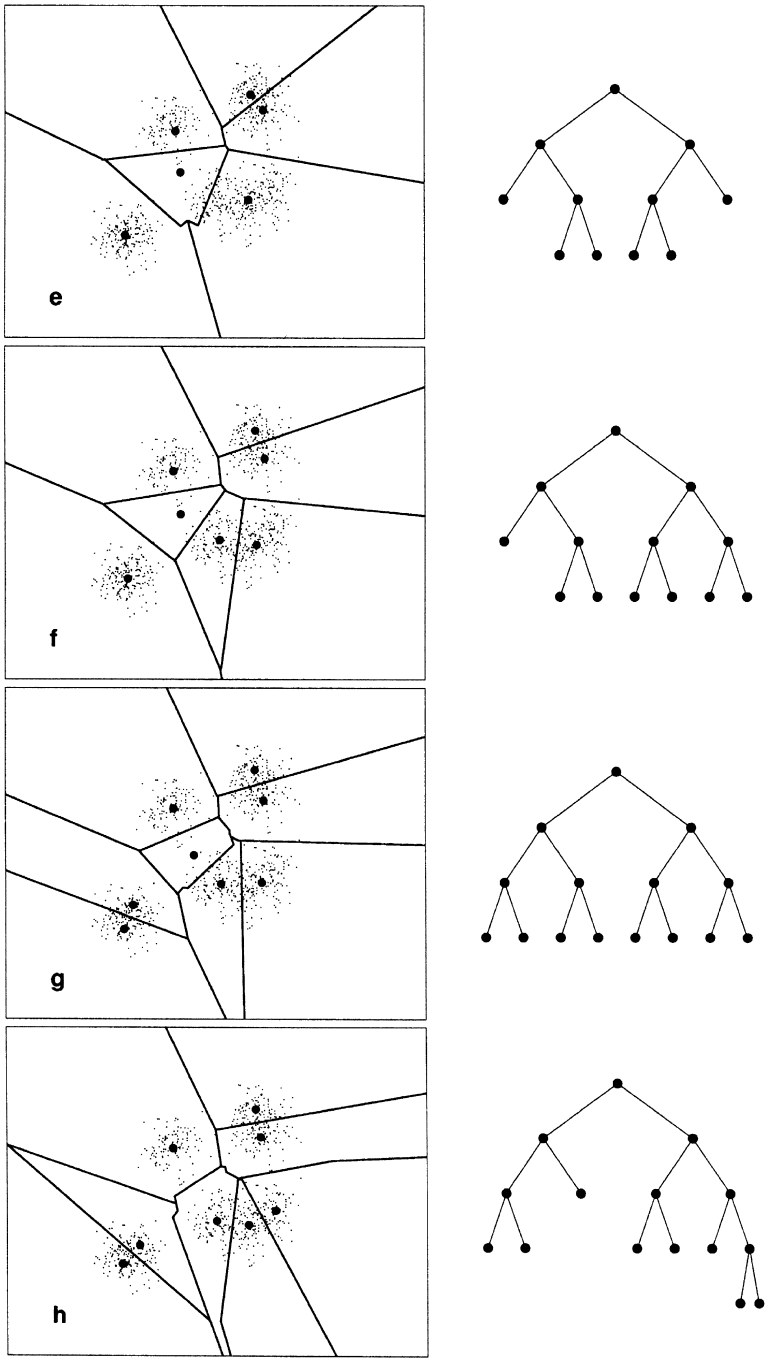


Fig. 10. (*continued*)

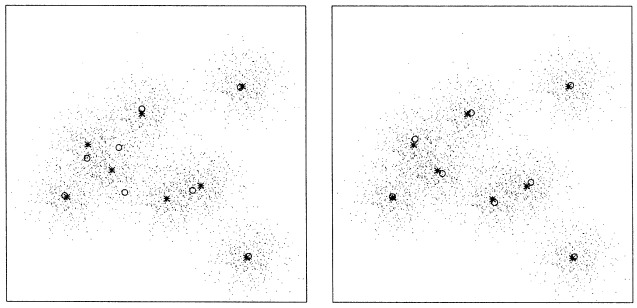
6. Vector quantization

Vector quantization is a special case of clustering. It is mainly used for data compression, to represent images and information. Applications of vector quantization include speech and image transmission.

Fig. 16 illustrates a general data compression system. The system has two components: an *encoder* and a *decoder*. The encoder converts the original data into a compressed representation that has a smaller size in bits than the original data. The decoder uses the compressed data to reconstruct the original data. The reconstructed data may be either identical to the original data (lossless compression systems) or a close match (lossy compression systems).

Vector quantization is a lossy compression technique that uses a codebook for encoding and decoding data. Vector quantization techniques are aimed at creating small codebooks capable of encoding and decoding with the smallest possible difference between original and reconstructed data.

The search procedure for vector quantization (VQ) methods



(a) TSVQ (b) S-TREE2

Fig. 11. An example with a mixture of eight isotropic Gaussian components. (a) TSVQ solution with sum of squared errors (SSE) = 8.6; (b) S-TREE2 solution with SSE = 6.5. ○ represents leaf weight vectors and ∗ the actual Gaussian cluster centers.

may be either unconstrained (Equitz, 1989; Linde et al., 1980) or constrained. Constrained search procedures include tree-structured (TSVQ) (Buzo, Gray, Gray, & Markel, 1980; Makhoul, Roucos, & Gish, 1985) and lattice (Conway & Sloane, 1985; Gersho, 1979) methods. In the unconstrained (full search) case, all weight vectors are codewords, and the system searches through all the entries in a codebook to find which one best represents the data. In the constrained case, a subset of the weight vectors are used as codewords (e.g. in TSVQ the leaf nodes are used as codewords), and only some codewords are searched.

In the past 20 years, many new developments have aimed at increasing the speed of vector quantizers. These include splitting (Linde et al., 1980), single-node splitting (Makhoul et al., 1985), fine-coarse VQ (Moayeri, Neuhoff, & Stark, 1991), subspace-distortion method (Chan & Po, 1992; Po & Chan, 1990, 1994), pairwise nearest-neighbor (PNN) algorithm (Equitz, 1989), principal component-based splitting (Wu & Zhang, 1991), maximum descent (MD) algorithm (Chan & Ma, 1994), and fast tree-structured encoding (Katsavounidis, Kuo, & Zhang, 1996). A limitation of these methods is that they create codebooks offline, requiring all the data for training the system to remain in memory throughout training. For large databases, this places heavy demands on the system.

6.1. Competitive learning for online vector quantization

Recently there has been a growing interest in competitive learning neural network approaches to vector quantization (Ahalt, Krishnamurty, Chen, & Melton, 1990; Amerijckx, Verleysen, Thissen, & Legat, 1998; Bruske & Sommer, 1995; Buhmann & Kuhnel, 1993; Butler & Jiang, 1996; Choi & Park, 1994; Chung & Lee, 1994; Fritzke, 1995;

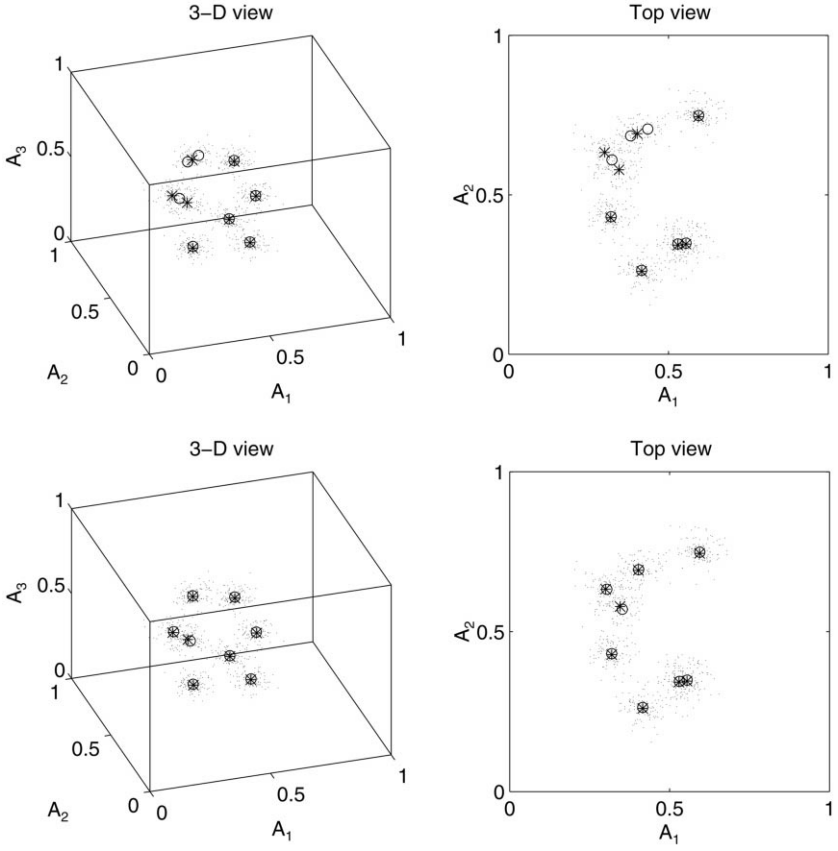
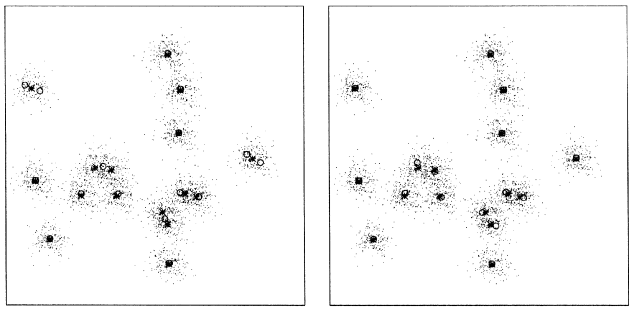
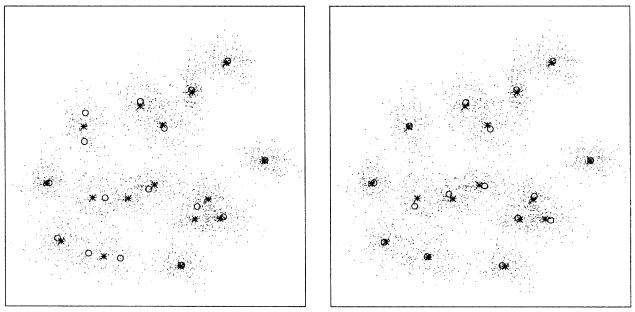


Fig. 12. An example with a 3-D mixture of eight isotropic Gaussian components. (a) TSVQ solution with SSE = 12.3. (b) S-TREE2 solution with SSE = 11.8. ○ represents leaf weight vectors and ∗ the actual Gaussian cluster centers.



(a) K-means (b) S-TREE2

Fig. 13. An example with a Gaussian mixture with 16 isotropic components. (a) K-means solution with $K = 16$ and SSE = 5.0. (b) S-TREE2 solution with SSE = 4.7. ○ represents leaf weight vectors and ∗ the actual Gaussian cluster centers.



(a) K-means (b) S-TREE2

Fig. 14. Example with a Gaussian mixture with 16 anisotropic components. (a) K-means solution with $K = 16$ and SSE = 35.8. (b) S-TREE2 solution with SSE = 34.7. ○ represents leaf weight vectors and ∗ the actual Gaussian cluster centers.

Hoffmann & Buhmann, 1996; Kohonen, 1988; Lee & Peterson, 1990; Li, Tang, Suen, Fang, & Jennings, 1992; Martinetz & Schulten, 1991; Racz & Klotz, 1991; Ueda & Nakano, 1994). These are online methods for codebook generation, and do not require all the data for training the system to be kept in memory throughout training.

The majority of competitive learning neural network methods use full search. The few tree-structured vector quantizers among them either compromise speed by requiring the update of all nodes during training (Held & Buhmann, 1998), or are not generally stable and have not been tested on large problems (Li et al., 1992; Racz & Klotz, 1991).

During training, competitive learning neural network approaches to vector quantization update the weight vectors according to the general equation:

$$\Delta \mathbf{w}_j = \alpha h_j(\mathbf{A} - \mathbf{w}_j),$$

where h_j is a node-specific learning rate and α is a global learning rate which decreases over time. For *hard*, or *winner-take-all*, competitive learning systems (e.g. Ahalt et al., 1990), $h_j = 1$ for the nearest-neighbor weight vector J and zero otherwise. For *soft*, or *distributed*, competitive learning systems (e.g. Chung & Lee, 1994; Kohonen, 1988; Martinetz & Schulten, 1991), h_j is non-zero at more than one node in the early stages of training, and slowly approaches the hard competitive learning case over time.

Many competitive learning methods attempt to minimize the sum of squared distances:

$$D = \sum_j \sum_{\mathbf{A} \in \Lambda_j} \|\mathbf{A} - \mathbf{w}_j\|^2, \tag{8}$$

where Λ_j is the set of inputs mapped to codeword $\mathbf{w}_j$, while also imposing the constraint that codewords have an equal probability of being selected (Choi & Park, 1994; Chung & Lee, 1994; Kohonen, 1988; Martinetz & Schulten, 1991). In recent years, approaches that attempt to equalize the distortion associated with each codeword have been proposed (Butler & Jiang, 1996; Ueda & Nakano, 1994). As illustrated in Fig. 17, smaller total distortion can be achieved using an equal distortion constraint.

S-TREE implements a hard competitive learning approach at each level of the tree, with a single winner per level. This constraint, combined with the tree-structured search for the best codeword, reduces the number of codewords searched and updated during training. As a result, S-TREE achieves faster codebook generation, encoding, and decoding than full search competitive learning approaches.

S-TREE can be applied to vector quantization with either a probability equalization or a distortion equalization goal. Distortion equalization is implemented by computing ϵ according to Eq. (5). Probability equalization is implemented using $\epsilon = 1$ to update the nodes in the path from the root node to the winning leaf. Fig. 17 illustrates results for both goals on a simple example.

6.2. Data compression measures

The *compression ratio* is one measure of quality of a data compression system. It is defined by

$$r = \frac{\text{size of original data in bits}}{\text{size of compressed data in bits}}$$

For example, a compression ratio $r = 2$ means that the compressed data require half the storage space of the original data. The higher the value of r the better the compression system.

Another useful figure of merit is the *compression rate* R (bit/sample) achieved by a vector quantization system:

$$R = \frac{\lceil \log_2 K \rceil}{M}$$

where $\lceil x \rceil$ is the smallest integer greater than or equal to x, K is the number of codewords, and M is the number of input dimensions. A quantizer with three codewords and 16 input dimensions produces a compression rate of $R = 2/16 = 1/8$ bit/sample. For images, compression rate measures bit per pixel (bpp). The lower the value of R the better the compression system.

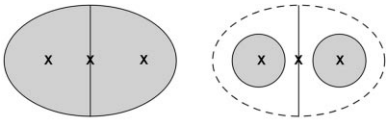


Fig. 15. Dividing a cluster in half through the centroid (×) creates a smaller total cost. (a) The large cluster has no subclusters. The sum of squared distances is still reduced, because the distance from a data point to the two new centroids is on average smaller than the distance to the parent centroid. (b) The cluster has two subclusters. In this case, the decrease in the sum of squared distances is greater than in (a).

The quality of the reconstructed data can be measured using the *peak-signal-to-noise ratio* (PSNR) in dB:

$$\mathrm{PSNR} = 10\log_{10}\frac{\sigma^2}{\mathrm{MSE}} \tag{9}$$

where σ^2 is the variance of the original data and MSE is the reconstruction mean squared error. For gray-scale images with 256 (8-bit) gray levels, PSNR is defined as

$$\mathrm{PSNR} = 10\log_{10}\frac{256^2}{\mathrm{MSE}} \tag{10}$$

The examples in Section 7 use PSNR to measure the quality of the different data compression algorithms.

7. Vector quantization examples

This section compares S-TREE performance with performance of the tree-structured vector quantizer using the splitting method (TSVQ) (Linde et al., 1980; Makhoul et al., 1985) and of the generalized Lloyd algorithm (GLA) (Linde et al., 1980), on problems of vector quantization of Gauss–Markov sources and image compression. The TSVQ and GLA simulations were performed using programs developed by the University of Washington Data Compression Laboratory. The TSVQ program was used in the balanced tree mode, which allows the number of codewords generated by the program to be specified independently of

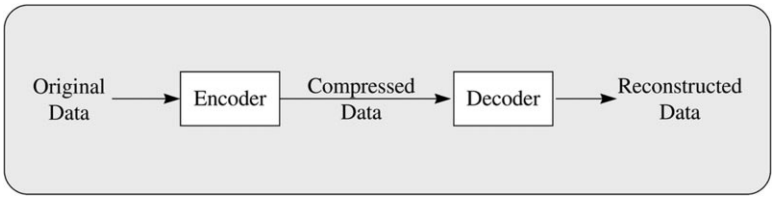


Fig. 16. A general data compression system.

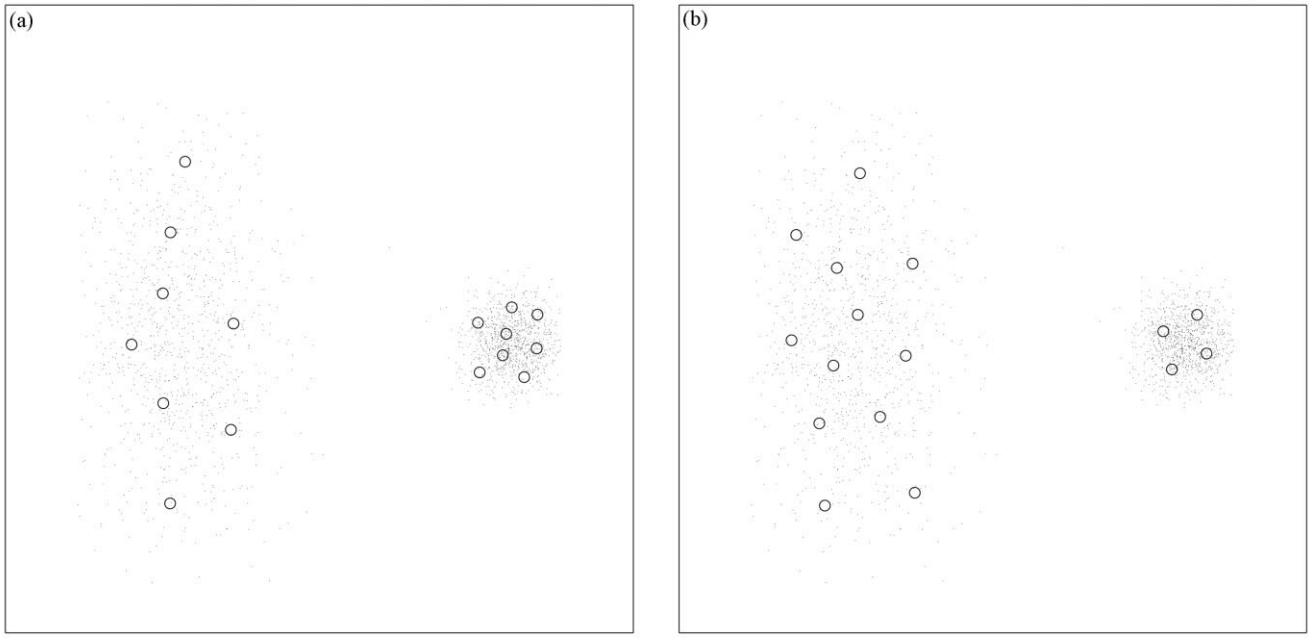


Fig. 17. Equal probability versus equal distortion constraints for vector quantization using S-TREE2. Each cloud has 1000 points, but the two have different standard deviations. (a) With an equal probability constraint, an equal number of codewords is assigned to each cloud and SSE = 8.4. (b) With equal distortion, SSE is reduced to 6.7. ○ marks the position of the 16 codewords in input space.

Table 2
S-TREE2 parameters for vector quantization simulations

Parameter	Value
All simulations	
E_0	50
β_1	0.02
β_2	0.075
β_3	0.2
γ	1.5
Γ	0.4
δ	0.0001
η	0.01
Gauss–Markov source	
T	1200
Image compression	
T	6000

the data. The GLA program was used in unconstrained search mode. In order to make training time comparisons meaningful, both programs were modified to have all the data in memory before the start of training. This eliminated the impact of disk access time on the training time results.

In order to obtain average performance values for the algorithms, the training process was repeated 20 times for each example, each instance using a different random ordering of the data. The simulation parameters used by S-TREE are listed in Table 2.

7.1. Gauss–Markov sources

Vector quantizers were first tested on the classical Gauss–Markov source benchmark, with construction following Gersho and Gray (1992). Training sets were processed with input dimensions $M = 1, 2, \ldots, 7$, with each training set consisting of 60,000 input vectors. Data points were created using sequences of values from the following random process:

$$X_{t+1} = 0.9X_t + u_t$$

where u_t is a zero-mean, unit-variance Gaussian random variable. Each sequence was converted to vectors with the appropriate dimensions. For example, for a training set with two-dimensional input vectors, a sequence of length $n = 120{,}000$ was created $\{X_1, X_2, \ldots, X_n\}$ and then converted to a set of 60,000 two-dimensional vectors $\{(X_1, X_2), (X_3, X_4), \ldots, (X_{n-1}, X_n)\}$. Training sets were encoded with codebooks of size 2^M where M is the number of dimensions of the input vector. Performance was measured using PSNR (9).

Simulation results for the Gauss–Markov task are summarized in Table 3 and illustrated in Figs. 18 and 19. In most cases, especially in high dimensions, S-TREE1 outperformed TSVQ in signal-to-noise ratio while requiring less training time. S-TREE2, as expected, showed even better signal-to-noise performance, approaching that of the full search GLA. This comparison is further illustrated in

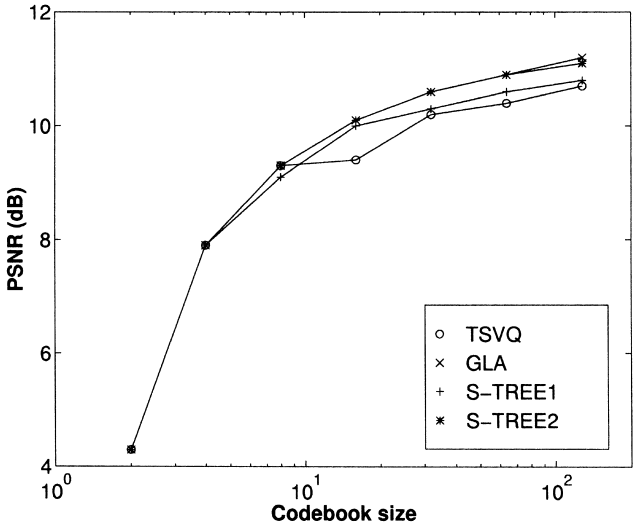


Fig. 18. PSNR (dB) for the Gauss–Markov task. PSNR is measured on the training set.

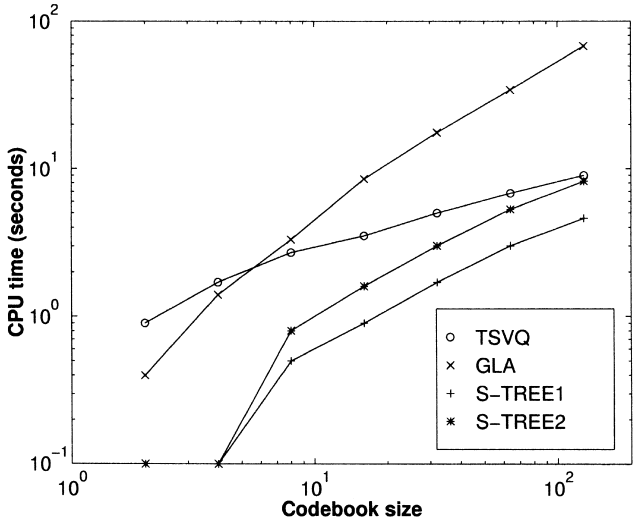


Fig. 19. Training time (s) for the Gauss–Markov task.

Table 4, which shows that the performance (PSNR) gap between GLA and TSVQ, introduced by the tree-structured bias, is almost completely recovered by S-TREE2. The ability of the double-path search to overcome the bias decreases with codebook size since, for large trees, the likelihood increases that the nearest codeword is in one of the paths not explored.

S-TREE2 also requires significantly less time than TSVQ for smaller codebook sizes. However, as the codebook size grows, S-TREE2 training time approaches that of TSVQ. In fact, S-TREE2 training time will eventually become larger than that of TSVQ, but it will always be smaller than that of GLA. Note that most of the savings in training time achieved by TSVQ requires batch processing. Because S-TREE is an online algorithm it has to traverse the tree starting at the root node to find the nearest neighbor leaf, for each

Table 3
PSNR and training time for the Gauss–Markov task. Numbers reflect average results over 20 random orderings of the training data. Best results are in boldface

Input dimension	Codebook size	PSNR (dB)			
		TSVQ	GLA	S-TREE1	S-TREE2
1	2	**4.3** ± 0.00	**4.3** ± 0.00	4.3 ± 0.05	4.3 ± 0.05
2	4	**7.9** ± 0.00	**7.9** ± 0.00	7.9 ± 0.01	7.9 ± 0.16
3	8	**9.3** ± 0.00	**9.3** ± 0.03	9.1 ± 0.20	9.3 ± 0.07
4	16	9.4 ± 0.00	**10.1** ± 0.02	10.0 ± 0.06	10.1 ± 0.04
5	32	10.2 ± 0.00	**10.6** ± 0.03	10.3 ± 0.05	**10.6** ± 0.03
6	64	10.4 ± 0.00	**10.9** ± 0.02	10.6 ± 0.04	**10.9** ± 0.02
7	128	10.7 ± 0.00	**11.2** ± 0.01	10.8 ± 0.05	11.1 ± 0.01
		Training time (s)			
1	2	0.9 ± 0.02	0.4 ± 0.02	**0.1** ± 0.02	**0.1** ± 0.02
2	4	1.7 ± 0.02	1.4 ± 0.02	0.1 ± 0.05	**0.1** ± 0.04
3	8	2.7 ± 0.04	**3.3** ± 0.15	**0.5** ± 0.14	0.8 ± 0.17
4	16	3.5 ± 0.03	8.5 ± 0.73	**0.9** ± 0.14	1.6 ± 0.21
5	32	5.0 ± 0.40	17.6 ± 0.94	**1.7** ± 0.14	3.0 ± 0.28
6	64	6.8 ± 0.05	34.4 ± 1.33	**3.0** ± 0.25	5.3 ± 0.28
7	128	9.0 ± 0.08	68.1 ± 2.57	**4.6** ± 0.17	8.2 ± 0.31

input vector. TSVQ, on the other hand, processes a dataset one level at a time, storing the assignments of inputs vectors to tree nodes. This avoids the need to traverse the tree starting at the root when a new level is trained. This strategy cannot be applied online.

In this benchmark, an online approach such as S-TREE, which needs to start the search from the root node for each input vector, is penalized twice. Because the codebook size *and* the number of input dimensions grow simultaneously, the computational requirements for large codebooks increase more quickly than if the number of dimensions were fixed.

7.2. Image compression

For the image compression example, a training set was prepared by taking 4×4 blocks from four 256×256 gray-scale (8-bit) images (Bird, Bridge, Camera, Goldhill) in

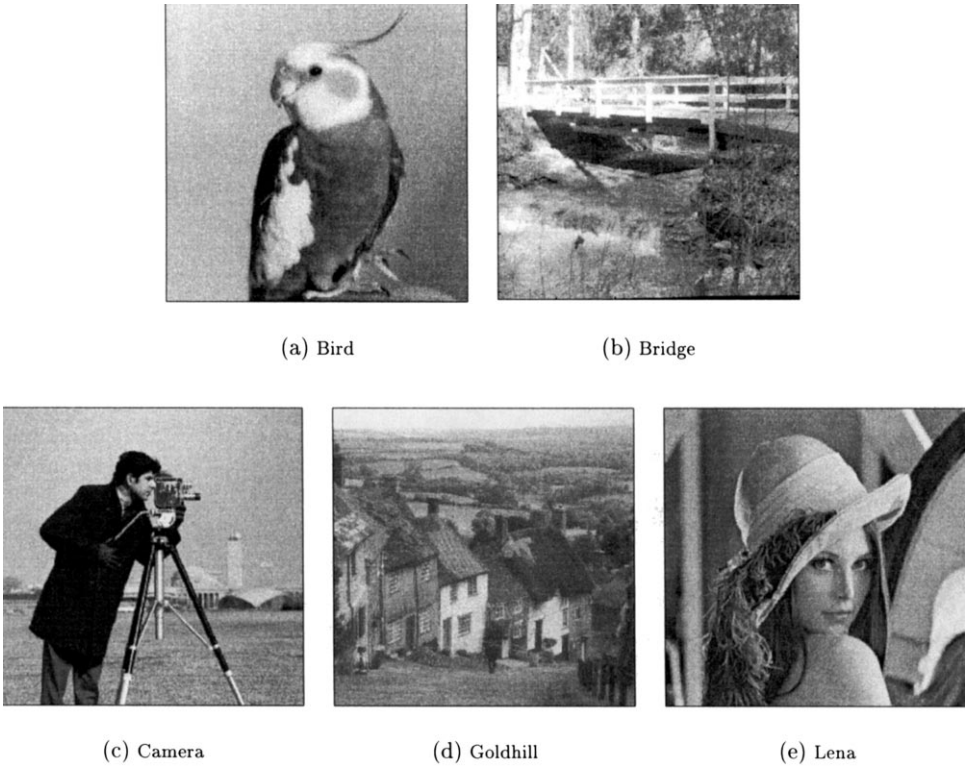


Fig. 20. 256×256 gray-scale (8-bit) images used in the image compression task. Images (a)– (d) were used as training set, and image (e) was used as testing set.

Table 4
Gain in dB for GLA and S-TREE2, and percentage of the performance gap recovered by S-TREE2 for the Gauss–Markov task. Gain is measured as the method's PSNR minus TSVQ's. % recovered equals S-TREE2 gain divided by GLA gain

Input dimension	Codebook size	Gain (dB)		% Recovered
		GLA	S-TREE2	
1	2	0.00	− 0.02	–
2	4	0.00	− 0.05	–
3	8	0.02	0.02	100
4	16	0.71	0.70	98
5	32	0.41	0.39	95
6	64	0.48	0.43	90
7	128	0.44	0.36	83

Fig. 20(a)– (d). These blocks were transformed into vectors, resulting in a training set with 16,384 16-dimensional vectors. A test set was prepared in a similar fashion using the 256 × 256 gray-scale (8-bit) Lena image (Fig. 20(e)). Quantizers with codebook sizes ranging from 2 to 512 were then trained using S-TREE1, S-TREE2, TSVQ, and GLA.

The results of the simulations are summarized in Table 5 and illustrated in Figs. 21 and 22. Sample reconstruction images are illustrated, for codebooks with 128 and 256 codewords, in Figs. 23 and 24, respectively. S-TREE1 outperformed TSVQ both in reconstruction quality (PSNR (10)) and training time. S-TREE2 performance was better than that of S-TREE1 in PSNR, but it required more time. In this application, because the input dimension is kept constant for different codebook sizes, S-TREE2 training time is below that of TSVQ even for large codebook sizes. Table 6 shows that S-TREE2 was capable of recovering much of the performance (PSNR) gap between GLA and TSVQ. As the codebook size increases, the size of the performance gap recovered by S-TREE2 decreases, as in Table 4.

Table 5
PSNR and training time for the image compression task. Numbers reflect average results over 20 random orderings of the training data. Best results are in boldface

Codebook size	PSNR (dB)			
TSVQ	GLA	S-TREE1	S-TREE2	
2	17.6 ± 0.00	17.6 ± 0.00	**17.7** ± 0.07	**17.7** ± 0.07
4	21.2 ± 0.00	21.2 ± 0.00	**21.3** ± 0.05	21.2 ± 0.12
8	**22.6** ± 0.00	**22.6** ± 0.00	22.4 ± 0.17	22.6 ± 0.09
16	23.2 ± 0.00	23.4 ± 0.04	23.2 ± 0.11	**23.5** ± 0.17
32	23.7 ± 0.00	**24.7** ± 0.07	24.2 ± 0.13	24.5 ± 0.07
64	24.8 ± 0.00	**25.5** ± 0.05	25.1 ± 0.10	25.3 ± 0.06
128	25.5 ± 0.00	**26.2** ± 0.05	25.8 ± 0.09	26.0 ± 0.06
256	26.1 ± 0.00	**26.9** ± 0.03	26.3 ± 0.07	26.6 ± 0.06
512	26.7 ± 0.00	**27.4** ± 0.03	26.8 ± 0.05	27.1 ± 0.05
	Training time (s)			
2	1.0 ± 0.02	0.9 ± 0.03	**0.1** ± 0.05	**0.1** ± 0.05
4	1.6 ± 0.03	1.6 ± 0.08	**0.3** ± 0.06	0.4 ± 0.09
8	2.0 ± 0.04	2.8 ± 0.03	**0.6** ± 0.12	0.8 ± 0.20
16	2.7 ± 0.03	5.0 ± 0.29	**0.9** ± 0.18	1.2 ± 0.23
32	3.7 ± 0.06	11.6 ± 0.71	**1.3** ± 0.28	1.8 ± 0.31
64	4.5 ± 0.07	18.7 ± 0.86	**1.9** ± 0.37	2.6 ± 0.35
128	5.5 ± 0.06	32.8 ± 1.59	**2.6** ± 0.29	3.6 ± 0.55
256	6.9 ± 0.07	59.1 ± 2.65	**3.3** ± 0.43	4.9 ± 0.80
512	8.8 ± 0.07	105.3 ± 2.40	**4.3** ± 0.39	6.3 ± 0.80

8. Related work

Incremental tree-structured methods for clustering have received a great deal of attention in the past few years. Some are online methods (Choi & Park, 1994; Held & Buhmann, 1998; Li et al., 1992; Racz & Klotz, 1991); others are offline (Chang & Chen, 1997; Hoffmann & Buhmann, 1995; Landelius, 1993; Miller & Rose, 1994, 1996; Xuan & Adali, 1995). Typically, tree-based approaches suffer from the

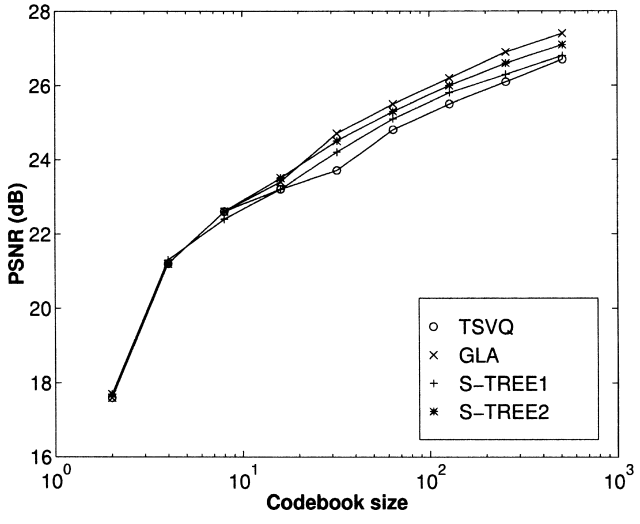


Fig. 21. PSNR (dB) for the image compression task. PSNR is measured on a test image ('Lena') not included in the training set.

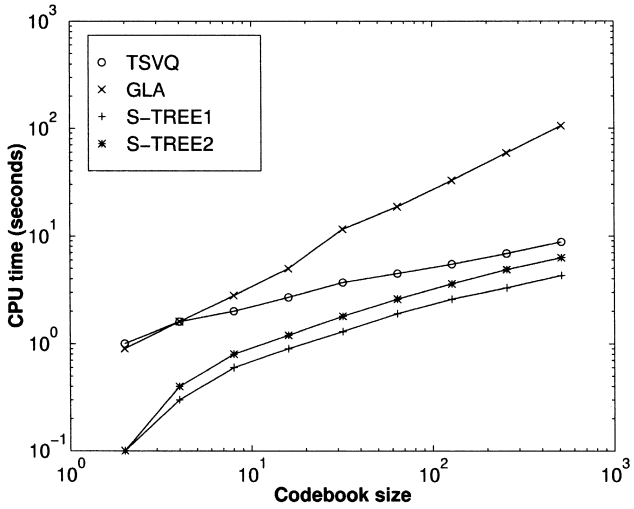


Fig. 22. Training time (s) for the image compression task.

(a) TSVQ, 25.5 dB (b) GLA, 26.2 dB

(c) S-TREE1, 25.8 db (d) S-TREE2, 26.0 dB

(e) Original

Fig. 23. Reconstructed Lena image for different algorithms and 128 codewords. Original encoded at 8 bpp.

bias created by imposing a tree-structured constraint on the solution of the clustering problem. S-TREE2, with its double-path search, minimizes this bias.

Xuan and Adali (1995) proposed the learning tree-structured vector quantization algorithm (LTSVQ). This is an offline algorithm similar to TSVQ, the difference being the use of a sequential competitive learning rule for updating the codewords instead of the batch rule used by TSVQ. As a result, LTSVQ presents the same limitations of TSVQ. Because S-TREE1 and S-TREE2 train all the levels of the tree simultaneously, they are capable of learning a codebook faster than TSVQ (Figs. 19 and 22), which trains one level at a time.

Landelius (1993) proposed a tree-structured algorithm that partitions the space at the centroid of data along the principal component. This is exactly what S-TREE1 approximates iteratively. While the method of Landelius

(a) TSVQ, 26.1 dB (b) GLA, 26.9 dB

(c) S-TREE1, 26.3 db (d) S-TREE2, 26.6 dB

(e) Original

Fig. 24. Reconstructed Lena image for different algorithms and 256 codewords. Original encoded at 8 bpp.

requires the storage of covariance matrices as well as computing the eigenvectors of these matrices, S-TREE1 does not. There is no equivalent to S-TREE2 and its improved search in that work.

Li et al. (1992) and Racz and Klotz (1991) used trees with variable numbers of children per node and threshold-based splitting rules. This has the potential of overcoming the bias introduced by tree structures with a fixed number of children per node. There is little information on how these methods perform on more difficult tasks. However, a comparative study (Butchart, Davey, & Adams, 1996) indicates that both approaches are sensitive to parameter settings, have problems dealing with noisy data, and seem to be affected by the tree-structure bias for trees with many levels. S-TREE1 is not very sensitive to the first two problems, and S-TREE2 minimizes the bias. SCONN (Choi & Park, 1994), which also uses a

Table 6
Gain in dB for GLA and S-TREE2, and percentage of the performance gap recovered by S-TREE2 for the image compression task. Gain is measured as the method's PSNR minus TSVQ's. % recovered equals S-TREE2 gain divided by GLA gain

Codebook	Gain (dB)		% recovered
size	GLA	S-TREE2	
2	0.01	0.063	>100
4	0.00	−0.022	–
8	0.01	0.044	>100
16	0.15	0.287	>100
32	0.97	0.779	81
64	0.71	0.551	78
128	0.68	0.474	69
256	0.75	0.473	63
512	0.78	0.463	60

distance threshold to control the creation of new nodes, has shown good performance on a few benchmarks. Although this algorithm is cast as a hierarchical structure, SCONN uses an unconstrained search for selecting the nearest neighbor leaf. As a result, it does not scale well with the number of input dimensions and codebook size.

Some recent methods have introduced interesting new ways of determining the tree topology based on information theoretical principles. For example, the number of leaves may be intrinsically determined by a complexity cost that penalizes complex tree structures (Held & Buhmann, 1998) or by specifying an annealing temperature (Hoffmann & Buhmann, 1995; Miller & Rose, 1994, 1996). However, these approaches require the processing of all the nodes in the tree for each training observation. This is especially serious in the annealing algorithms: while the effective size of the tree for each annealing temperature might be small, the actual size of the processed tree can be much larger. At high temperatures, many of the nodes in the tree have the 'same' codewords, which yields a small effective tree. As the temperature cools, these nodes gradually differentiate and the effective tree grows. Throughout this process, computations are performed for the whole tree, not only the effective tree. This computational load penalizes speed, and raises the question of how well these approaches scale with database and codebook size, and with the number of input dimensions. The same drawback is shared by a similar approach (Jordan & Jacobs, 1994) for growing trees for regression applications.

Clustering by melting (Wong, 1993) also proposes a new way of detecting clusters in data. The algorithm is agglomerative: it starts with each data point as a separated cluster and then gradually merges nearby clusters into a single cluster. This procedure is repeated until a single cluster is obtained. The natural clusters and hierarchical structures in the data are identified afterwards based on the analysis of bifurcations in scale space. This method can in principle deal with cluster variability in size, shape, and density. However, it is an offline approach and is computationally expensive.

TSOM (Koikkalainen, 1994; Koikkalainen & Oja, 1990) is a tree-structured clustering method that enforces topology preservation in the codebook. That is, similar codewords have nearby indices in the codebook. TSOM minimizes the tree-structure bias by searching, for each level of the tree, the neighbors of the best unit in the tree at that level. However, TSOM can generate only balanced trees and clusters with approximately equal numbers of data points. This is inadequate for many applications, including vector quantization.

The use of multiple searching paths in a tree has also been proposed by Chang, Chen, and Wang (1992)and further developed in the closest-coupled tree-structured vector quantization (CCTSVQ) (Chang & Chen, 1997). CCTSVQ has a number of drawbacks when compared with S-TREE2. CCTSVQ relies on the TSVQ algorithm to generate its codebook in an offline fashion. It also requires extra storage to keep a pointer for each node in the tree. This pointer stores the index of the node in the same level with the closest codeword to the codeword stored in the node owning the pointer. The multipath search in CCTSVQ can compensate for some of the bias introduced by the tree structure. However, it cannot compensate for bad placement of leaf codewords due to TSVQ's inability to minimize the tree-structure bias during codebook generation. S-TREE2, as illustrated in the examples in this paper, uses the double-path search to improve the placement of leaf codewords.

9. Conclusion

S-TREE1 is a fast tree-structured clustering algorithm, with online creation and pruning of tree nodes. It partitions the space along the principal components of the data, and can be used with different cost functions and model selection criteria (e.g. maximum number of nodes or minimum acceptable error at the leaves).

An alternative version, S-TREE2, introduces a new multipath search procedure which is integrated with the tree building process. This multipath search approach allows S-TREE2 to overcome, in many cases, the bias introduced by the tree-structured constraint on the solution of the clustering problem. For deeper trees, the ability of the double-path search to overcome this bias decreases.

S-TREE algorithms can also be used to implement online tree-structured vector quantizers. Unlike other neural network tree-structured methods, S-TREE is fast and robust to parameter choices. These features make it a viable solution to real vector quantization tasks such as image compression. To illustrate this, the method is tested on a Gauss–Markov source benchmark and an image compression application. S-TREE performance on these tasks is compared with the standard TSVQ and GLA algorithms. S-TREE's image reconstruction quality approaches that of GLA while taking less than 10% of computer time (Table 5). S-TREE also compares favorably with the standard TSVQ in both the time needed to create the codebook and the quality of image reconstruction.

Table 7
S-TREE parameters and variables. Parameter ranges for simulations are indicated in brackets

Parameter	Description
E_0	Initial value for E [50]
β_1	Learning rate for E [0.01, 0.02]
β_2	Learning rate for average cost $\bar{\epsilon}_0$ [0.075]
β_3	Learning rate for $\bar{C}_\tau$ [0.2]
γ	Multiplicative offset used after the tree modification step [1.1, 1.5]
Γ	Pruning threshold [0.35, 0.4]
δ	Multiplicative offset for initializing new child weights after a split [0.0001]
η	Convergence threshold [0.01]
U	Maximum number of tree nodes [an odd integer]
T	Window size used with the online convergence criterion: typical values are 5–20% of the training set size
Variable	**Description**
$\mathbf{A}$	Input vector $(A_1 \cdots A_i \cdots A_M)$
$\mathbf{w}_j$	Weight vector (codeword) for jth tree node $(w_{1j} \cdots w_{ij} \cdots w_{Mj})$
e_j	Relative cost associated with the jth tree node
N_j	Number of times the jth tree node has been updated
P_j	Index of the parent of node j; for the root node $P_1 = 0$
S_j	Index of the sibling of node j
θ_j	Set of indices of node j's children
Ω	Set of indices of nodes in path from root to winning leaf
l_j	Tree depth of node j; this is defined as the number of nodes (not including the root node) in the path connecting j to the root node
ϵ_0	Distortion at the winning leaf node
$\bar{\epsilon}_0$	Average distortion of the winning leaf nodes
ϵ	Relative cost of the winning leaf (distortion/average distortion)
E	Splitting threshold
u	Maximum index of the tree nodes
τ	Counter for online convergence criterion
C	Total cost for current window
$\bar{C}_\tau$	Moving average of total cost at window τ

Besides clustering and vector quantization, S-TREE can also be used, with minor modifications, in classification, function approximation, probability density estimation, and curve and surface compression. These areas offer many opportunities for the application of incremental methods such as S-TREE.

Acknowledgements

This research was supported in part by the Office of Naval Research (ONR N00014-95-10409 and ONR N00014-95-0657).

Appendix A. S-TREE algorithms

This appendix presents a detailed description of the S-TREE1 (single-path search) and S-TREE2 (double-path search) algorithms. Table 7 defines parameters and variables. The implementation assumes squared distance distortion and distortion equalization goals, and limits the number of nodes to a prescribed maximum (U).

A.1. S-TREE1: single-path search

Main algorithm

Initialize:

(0) Initialize the tree with a single node: set $t = 1$, $u = 1$, $\mathbf{w}_j = \mathbf{0}$, $N_j = 0$, $e_j = 0$, $\bar{\epsilon}_0 = 0$, $E = E_0$, $\tau = 0$, $C = 0$, $P_1 = 0$

Get data;

(1) Get tth input vector $\mathbf{A}$

Find leaf:

(2) Find winning leaf J (via single-path search below)

Modify tree structure:

(3) If $e_J \leq E$ or $U = 1$ go to (7)

(4) If $u \geq U$ then prune (remove two nodes via pruning step below)

(5) Split (add two nodes via splitting step below)

(6) Multiply E by γ

Adapt nodes in path from root to winning leaf J:

(7) Compute distortion at winning leaf: $\epsilon_0 = \|\mathbf{A} - \mathbf{w}_J\|^2$

(8) Adjust $\bar{\epsilon}_0$ according to:

$$\Delta\bar{\epsilon}_0 = \begin{cases} \epsilon_0 & \text{if } t = 1 \\ \beta_2(\epsilon_0 - \bar{\epsilon}_0) & \text{otherwise} \end{cases}$$

(9) Compute relative cost: $\epsilon = \epsilon_0/\bar{\epsilon}_0$

(10) Adjust E according to: $\Delta E = \beta_1(e_J - E)$

(11) Set Ω to the index set of nodes in the path from the root to the winning leaf J

(12) Adjust e_j according to: $\Delta e_j = \begin{cases} \epsilon & \text{if } j \in \Omega \\ 0 & \text{otherwise} \end{cases}$

(13) Adjust N_j according to: $\Delta N_j = \begin{cases} 1 & \text{if } j \in \Omega \\ 0 & \text{otherwise} \end{cases}$
(14) Adjust $\mathbf{w}_j$ according to:

$$\Delta \mathbf{w}_j = \begin{cases} (\mathbf{A} - \mathbf{w}_j)/N_j & \text{if } j \in \Omega \\ \mathbf{0} & \text{otherwise} \end{cases}$$

Check convergence:
(15) Adjust C according to $\Delta C = \epsilon_0$
(16) Add 1 to t
(17) If t/T is not an integer go to (1)
(18) Add 1 to τ
(19) If $\tau = 1$, set $\bar{C}_\tau = C$ and go to (22)
(20) Compute $\bar{C}_\tau$ according to: $\bar{C}_\tau = \bar{C}_{\tau-1} + \beta_3(C - \bar{C}_{\tau-1})$
(21) If $|\bar{C}_{\tau-1} - \bar{C}_\tau|/|\bar{C}_{\tau-1} < \eta$, STOP
(22) Set $C = 0$ and go to (1)

Step (2): Single-path search (S-TREE1)
(2.1) Set $J = 1$ (the root node)
(2.2) If J is a leaf (node without children) go to (3)
(2.3) Let θ_J be the set of J's children
(2.4) Let $k = \text{argmin}_{j \in \theta_J} \|\mathbf{A} - \mathbf{w}_j\|$
(2.5) Set $J = k$ and go to (2.2)

Step (4): Pruning
(4.1) Let Y be the index set of leaf nodes
(4.2) Let $m = \text{argmin}_j(e_j),\ j \in Y$
(4.3) If $e_m/e_J > \Gamma$, go to (6)
(4.4) Type I: m's sibling is not a leaf
4.4.1 If S_m is a leaf go to (4.5)
4.4.2 Set $Z = P_{P_m}$
4.4.3 Delete m and P_m
4.4.4 Replace P_m with S_m
4.4.5 If $Z = 0$, for $q = 1, 2$ set

$$\theta_Z(q) = \begin{cases} S_m & \text{if } \theta_Z(q) = P_m \\ \theta_Z(q) & \text{otherwise} \end{cases}$$

4.4.6 Go to (5)
(4.5) Type II: node m's sibling is leaf J
4.5.1 If $S_m \neq J$ go to (4.6)
4.5.2 Delete nodes m and J
4.5.3 Set $\theta_{P_m} = \emptyset$
4.5.4 Set $J = P_J$
4.5.5 Go to (5)
(4.6) Type III: node m's sibling is a leaf other than J
4.6.1 Delete nodes m and S_m
4.6.2 Divide e_{P_m} by 2
4.6.3 Set $\theta_{P_m} = \emptyset$
4.6.4 Go to (5)

Step (5): Splitting
(5.1) Set $\theta_J(1) = u + 1$ and $\theta_J(2) = u + 2$
(5.2) Set $N_j = 1,\ j \in \theta_J$
(5.3) Set $e_j = e_J/2,\ j \in \theta_J$
(5.4) Set $w_{i\theta_J(1)} = w_{iJ}$
(5.5) Set $w_{i\theta_J(2)} = (1 + \delta)w_{iJ}$
(5.6) Increase u by 2
(5.7) Set $J = \theta_J(1)$

A.2. S-TREE2: double-path search

S-TREE2 is implemented by substituting the single-path search (step (2)) in the basic algorithm with the following algorithm.

Step (2): Double-path search
(2.1) Set $J = 1$ (root node)
(2.2) If $u = 1$ go to (3)
(2.3) Set J_1 and J_2, respectively, to the left and the right child of the root node
(2.4) If $\theta_{J_1} = \emptyset$ set $\Psi_1 = \{J_1\}$, otherwise set $\Psi_1 = \theta_{J_1}$
(2.5) If $\theta_{J_2} = \emptyset$ set $\Psi_2 = \{J_2\}$, otherwise set $\Psi_2 = \theta_{J_2}$
(2.6) Set $J = \begin{cases} J_1 = \arg\min_j \| \mathbf{A} - \mathbf{w}_j \|, j \in \{\Psi_1 \cup \Psi_2\} \\ J_2 = \arg\min_j \| \mathbf{A} - \mathbf{w}_j \|, j \in \{\Psi_1 \cup \Psi_2\} \text{ and } j \neq J_1 \end{cases}$
(2.7) If either J_1 or J_2 is not a leaf go to (2.4)
(2.8) Set l_{J_1} and l_{J_2} to the tree depths of the leaf nodes J_1 and J_2, respectively
(2.9) Set $J = \begin{cases} J_1 \text{ if } l_{J_1} \leq l_{J_2} \text{ or } u \geq U \\ J_2 \text{ otherwise} \end{cases}$

A.3. S-TREE testing

S-TREE1 and S-TREE2 use the following algorithm during testing.

(0) Set $C = 0$ and $t = 1$
(1) Get tth input vector $\mathbf{A}$
(2) Find winning leaf J (via single-path search for S-TREE1 or double-path search for S-TREE2)
(3) Compute distortion at winning leaf: $\epsilon_0 = \|\mathbf{A} - \mathbf{w}_J\|^2$
(4) Adjust C according to $\Delta C = \epsilon_0$
(5) If t is the last entry in the test set, STOP
(6) Add 1 to t
(7) Go to (1)

References

Ahalt, S. C., Krishnamurty, A. K., Chen, P., & Melton, D. E. (1990). Competitive learning algorithms for vector quantization. *Neural Networks*, *3* (3), 277–290.

Amerijckx, C., Velerysen, M., Thissen, P., & Legat, J. D. (1998). Image compression by self-organized Kohonen map. *IEEE Transactions on Neural Networks*, *9* (3), 503–507.

Ball, G., & Hall, D. (1967). A clustering technique for summarizing multivariate data. *Behavioral Science*, *12*, 153–155.

Bezdek, J. C. (1980). A convergence theorem for the fuzzy ISODATA clustering algorithms. *IEEE Transactions in Pattern Analysis and Machine Intelligence*, *PAMI-2*, 1–8.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*, Belmont, CA: Waldsworth.

Bruske, J., & Sommer, G. (1995). Dynamic cell structures. In G. Tesauro, D. Touretsky & T. Leen, *Advances in neural information processing systems 7* (pp. 497–504). Cambridge, MA: MIT Press.

Buhmann, J., & Kühnel, H. (1993). Vector quantization with complexity costs. *IEEE Transactions on Information Theory*, *39* (4), 1133–1145.

Butchart, K., Davey, N., & Adams, R. (1996). A comparative study of two self organizing and structurally adaptive dynamic neural tree networks. In J. G. Taylor, *Neural networks and their applications* (pp. 93–112). New York: John Wiley & Sons.

Butler, D., & Jiang, J. (1996). Distortion equalized fuzzy competitive learning for image data vector quantization. In IEEE Proceedings of ICASSP'96 (Vol. 6, pp. 3390–3394). New York: IEEE.

Buzo, A., Gray, Jr., A. H., Gray, R. M., & Markel, J. D. (1980). Speech coding based upon vector quantization. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *ASSP-28*, 562–574.

Chan, C. -K., & Ma, C. -K. (1994). A fast method of designing better codebooks for image vector quantization. *IEEE Transactions on Communications*, *42*, 237–242.

Chan, C. -K., & Po, L. -M. (1992). A complexity reduction technique for image vector quantization. *IEEE Transactions on Image Processing*, *1* (3), 312–321.

Chang, C. -C., & Chen, T. -S. (1997). New tree-structured vector quantization with closest-coupled multipath searching method. *Optical Engineering*, *36* (6), 1713–1720.

Chang, R. F., Chen, W. T., & Wang, J. S. (1992). Image sequence coding adaptive tree-structured vector quantization with multipath searching. *IEEE Proceedings—Part I*, *139* (1), 9–14.

Choi, D. -L., & Park, S. -H. (1994). Self-creating and organizing neural networks. *IEEE Transactions on Neural Networks*, *5* (4), 561–575.

Chung, F. L., & Lee, T. (1994). Fuzzy competitive learning. *Neural Networks*, *7* (3), 539–551.

Conway, J. H., & Sloane, N. J. A. (1985). Fast quantizing and decoding algorithms for lattice quantizers and codes. *IEEE Transactions in Information Theory*, *IT-28*, 227–232.

Cosman, P. C., Perlmutter, S. M., & Perlmutter, K. O. (1995). Tree-structured vector quantization with significance map for wavelet image coding. In J. A. Stores & M. Cohn, *Proceedings of the 1995 IEEE Data Compression Conference* (pp. 33–41). Snowbird, UT: IEEE Computer Society Press.

Duda, R. O., & Hart, P. E. (1973). *Pattern classification and scene analysis*, New York: Wiley-Interscience.

Dunn, J. C. (1974). A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters. *Journal of Cybernetics*, *3* (3), 32–57.

Equitz, W. J. (1989). A new vector quantization clustering algorithm. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *37* (10), 1568–1575.

Fränti, P., Kaukoranta, T., & Nevalainen, O. (1997). On the splitting method for vector quantization codebook generation. *Optical Engineering*, *36* (11), 3043–3051.

Fritzke, B. (1995). A growing neural gas network learns topologies. In G. Tesauro, D. S. Touretzky & T. K. Leen, *Advances in neural information processing systems 7* (pp. 625–632). Cambridge, MA: MIT Press.

Gersho, A. (1979). Asymptotically optimal block quantization. *IEEE Transactions on Information Theory*, *IT-25*, 373–380.

Gersho, A., & Gray, R. M. (1992). *Vector quantization and signal compression*, Boston, MA: Kluwer Academic Publishers.

Held, M., & Buhmann, J. M. (1998). Unsupervised on-line learning of decision trees for hierarchical data analysis. In M. I. Jordan & S. A. Solla, *Advances in neural information processing systems 10*. Cambridge, MA: MIT Press.

Hoffman, T., & Buhmann, J. M. (1995). Inferring hierarchical clustering structures by deterministic annealing. In F. Fogelman-Soulié & P. Gallinari, *Proceedings ICANN'95, International Conference on Artificial Neural Networks* (pp. 197–202), vol. II. Nanterre, France: EC2.

Hoffmann, T., & Buhmann, J. M. (1996). An annealed neural gas network for robust vector quantization. In C. von der Malsburg, W. von Seelen, J. C. Vorbruggen & B. Sendhoff, *Artificial neural networks—ICANN 96, 1996 International Conference Proceedings* (pp. 151–156), vol. 7. Berlin, Germany: Springer Verlag.

Jordan, M. I., & Jacobs, R. A. (1994). Hierarchical mixture of experts and the EM algorithm. *Neural Computation*, *6*, 181–214.

Katsavounidis, I., Kuo, C. C. J., & Zhang, Z. (1996). Fast tree-structured nearest neighbor encoding for vector quantization. *IEEE Transactions on Image Processing*, *5* (2), 398–404.

Kohonen, T. (1988). *Self-organization and associative memory*, (2nd edn). New York: Springer-Verlag.

Koikkalainen, P. (1994). Progress with the tree-structured self-organizing map. In A. G. Cohn, *11th European Conference on Artificial Intelligence* (pp. 211–215). New York: John Wiley & Sons.

Koikkalainen, P., & Oja, E. (1990). Self-organizing hierarchical feature maps. In Proceedings IJCNN-90, International Joint Conference on Neural Networks, Washington, DC (Vol. II, pp. 279–285). Piscataway, NJ: IEEE Service Center.

Landelius, T. (1993). Behavior representation by growing a learning tree. PhD dissertation, Linköping University, Sweden.

Lee, T. -C., & Peterson, A. M. (1990). Adaptive vector quantization using a self-development neural network. *IEEE Journal on Selected Areas in Communications*, *8* (8), 1458–1471.

Li, T., Tang, Y., Suen, S., Fang, L., & Jennings, A. (1992). A structurally adaptive neural tree for the recognition of large character set. In Proceedings of 11th IAPPR, International Conference on Pattern Recognition (Vol. 2, pp. 187–190). Los Alamitos, CA: IEEE Computer Society Press.

Linde, Y., Buzo, A., & Gray, R. M. (1980). An algorithm for vector quantizer design. *IEEE Transactions on Communications*, *COM-28*, 84–95.

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability (vol. 2, pp. 187–190). Los Alamitos, CA: IEEE Computer Society Press.

Makhoul, J., Roucos, S., & Gish, H. (1985). Vector quantization in speech coding. *Proceedings IEEE*, *73* (11), 1551–1587.

Martinetz, T., & Schulten, K. (1991). A 'neural-gas' network learns topologies. In T. Kohonen, K. Mäkisara, O. Simula & J. Kangas, *Proceedings International Conference on Artificial Neural Networks* (pp. 397–402), vol. I. Amsterdam, Netherlands: North-Holland.

Miller, D., & Rose, K. (1994). A non-greedy approach to tree-structured clustering. *Pattern Recognition Letters*, *15* (7), 683–690.

Miller, D., & Rose, K. (1996). Hierarchical, unsupervised learning with growing via phase transitions. *Neural Computation*, *8* (8), 425–450.

Moayeri, N., Neuhoff, D. L., & Stark, W. E. (1991). Fine-coarse vector quantization. *IEEE Transactions on Signal Processing*, *39* (7), 1503–1515.

Po, L. -M., & Chan, C. -K. (1990). Novel subspace distortion measurement for efficient implementation of image vector quantizer. *Electronics Letter*, *26*, 480–482.

Po, L. -M., & Chan, C. -K. (1994). Adaptive dimensionality reduction techniques for tree-structured vector quantization. *IEEE Transactions on Communications*, *42* (6), 2246–2257.

Racz, J., & Klotz, T. (1991). Knowledge representation by dynamic competitive learning techniques. In S. K. Rogers, *SPIE applications of artificial neural networks II* (pp. 778–783), vol. 1469. Bellingham, Washington: SPIE—The International Society for Optical Engineering.

Riskin, E. A., & Gray, R. M. (1991). A greedy tree growing algorithm for the design of variable rate vector quantizers. *IEEE Transactions on Signal Processing*, *39*, 2500–2507.

Ueda, N., & Nakano, R. (1994). A new competitive learning approach based on an equidistortion principle for designing optimal vector quantizers. *Neural Networks*, *7* (8), 1211–1227.

Wong, Y. (1993). Clustering data by melting. *Neural Computation*, *5*, 89–104.

Wu, X., & Zhang, K. (1991). A better tree-structured vector quantizer. In Proceedings of the 1991 Data Compression Conference (pp. 392–401). Snowbird, UT: IEEE Computer Society Press.

Xuan, J. & Adali, T. (1995). Learning tree-structured vector quantization for image compression. In Proceedings WCNN'95, World Congress on Neural Networks (Vol. I, pp. 756–759). Mahwah, NJ: Lawrence Erlbaum Associates.