Fast Vector Quantization with Topology Learning

Marcos M. Campos
Oracle
Data Mining Technologies
Burlington, USA
marcos.m.campos@oracle.com

Abstract – *A new vector quantization algorithm is introduced that is capable of learning the topology of the input distribution. The algorithm uses a tree-structured vector quantizer combined with topology learning to achieve fast performance and high accuracy. The approach can be applied to improve the performance of different types of tree-structured vector quantizers. This is illustrated with results for k-d trees and TSVQ on two high-dimensional datasets. The proposed method also allows the construction of topology-preserving graphs with one node per input row. The algorithm can be used for vector quantization, clustering, indexing, and link analysis.*

Key words – vector quantization, topology learning, constructive self-organizing maps, clustering, indexing, approximate nearest-neighbor search, unsupervised learning

1 Introduction

The problem of topology learning can be defined as: given some high-dimensional data distribution, find a topological structure that closely captures the topology of the data distribution [10]. This problem is closely related to the problem of learning a graph that captures the topological relationships in the data. The goal in topology learning contrasts with that of methods such as Self-Organizing Map (SOM) [13], Growing Cell Structure [9], Growing Hierarchical Self-Organizing Map [7], and ISOMAP [16] where the topology of the output space is fixed beforehand. These other methods are mainly concerned with dimensionality reduction. The mappings from the original space to the new space produced by projection methods frequently have topological defects. That is, neighboring points in input spaces may be mapped to far away points in the output or transformed space. Projection methods, however, are especially useful for representing multidimensional data in a form that can be visually inspected.

Learning a topological representation (graph) of a dataset can be used for vector quantization, clustering, link analysis, and indexing for nearest-neighbor and approximate nearest-neighbor searches. Several algorithms have been proposed for learning general topologies. These can be broadly classified into static (e.g., Neural Gas (NG) [15], and Optimally Topology Preserving Maps (OTPMS) [3]) and constructive architectures (e.g., Growing Neural Gas (GNG) [10], and SAM-SOM [6]). These algorithms can be seen as attempts to overcome the limitations in the SOM algorithm, including: fixed pre-defined output space topology (SOM uses a regular grid), poor scalability for large topologies, slow learning, and hard to tune parameters. All these methods create topological structures that are more flexible than SOM and thus better capture the topological relationships in the input data distribution. Constructive approaches speed up learning by leveraging hierarchical structures and growing the structure on demand. While most constructive methods use specialized data structures for speeding up learning, SAM-SOM proposes a different approach. It takes advantage of off-the-shelve hierarchical indexing methods to scale to large datasets and number of dimensions. This innovative proposal eliminates the need to develop specialized data structures for speeding up the search for the best matching unit (BMU), a key operation in topology learning algorithms.

Topology-learning algorithms usually attempt to learn the topology online. As a result, these algorithms require slow adaptation to the data. With few exceptions (e.g., GNG and SAM-SOM), online learning algorithms use multiple decaying parameters, which lead to relatively slow training. SAM-SOM is the only algorithm that attempts to learn a topological structure with a node for each input data vector. The algorithm use simple rules for creating and pruning connections. It is not clear, however, that these simple rules can approximate well the topology of input data distributions with uneven density and different dimensionalities in different areas of the input space.

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 data with the smallest possible difference between original and reconstructed data. Vector quantization can also be seen as a special case of clustering. As in clustering, many data records are mapped to a single codevector or cluster. Some applications of vector quantization include speech and image compression.

Vector quantizers for high dimensional vector spaces need a large codebook to achieve a small error rate. The Tree-Structured Vector Quantizer (TSVQ) [11] is a popular technique that scales well for large datasets and codebook sizes. Different versions of k-d trees have also been proposed for fast vector quantization [1]. k-d trees produce encoders with smaller memory footprint and faster encoding than TSVQ but, in general, they require larger codebooks for achieving the same level of compression of TSVQ.

As the size of the tree (codebook) grows the ability of approaches such as TSVQ and k-d trees to return the actual nearest neighbor to a input vector decreases. That is, the closest codevector (leaf centroid) to a given input may not be the one where the input is mapped to by the tree. The problem becomes more accentuated in axis-parallel approaches like k-d tree, where the partition imposed by the tree at each point is not well aligned with the data distribution principal directions. In general, tree-structured approaches trade speed for higher quantization error for a fixed codebook size when compared with full search approaches such as the LBG algorithm [11]. Some approaches have tried to minimize the impact of the error in the tree assignments by searching multiple paths at the same time [2, 4, 5] or by exploring a learned topological structure to search near-nodes for a better match [1, 7, 14]. Arya and Mount [1] have shown that the latter requires significantly less computation than the standard k-d tree approach for achieving the same level of error. Unfortunately, for a dataset with N input vectors, the RNG* algorithm used in [1] scales with $O(N^2)$, making it unsuitable for large datasets.

This paper proposes a new algorithm called a vector approximation graph (VA-graph) that leverages a tree based vector quantizer to quickly learn the topological structure of the data. It then uses the learned topology to enhance the performance of the vector quantizer. VA-graph can also learn graphs with as many nodes as the number of input vectors. Due to space constraints, the algorithm is presented in a batch version only. The paper is divided in 5 sections. Section 2 presents the tree construction step in VA-graph. Section 3 describes the topology learning capabilities. Section 4 reports the results for vector quantization experiments. Section 5 presents the conclusions and directions for future work.

2 Building the Tree

VA-graph combines a tree-structured vector quantizer with a fast topology-learning algorithm that relies on the tree-structured quantizer for its speed and scalability. The vector quantizer leverages the learned topology of the input data to achieve improved accuracy. The algorithm has three main pieces: tree construction, topology learning, and search or encoding. This section introduces the basic data partition scheme used in VA-graph for constructing the tree.

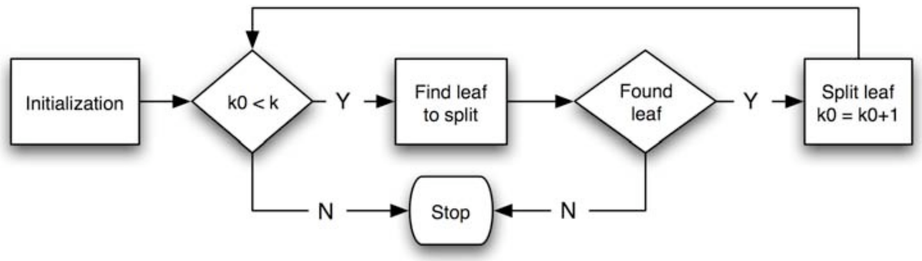

Figure 1: Tree building steps.

The algorithm builds a tree recursively as described in Figure 1. The algorithm is initialized with a single root node. The centroid of the root node is set to the mean of the data and the number of leaves (k_0) is set to 1. If the number of leaves is smaller than desired codebook size (k) then the eligible leaf node with the largest cost measure value is selected for splitting. Eligible leaf nodes are those with at least n data vectors assigned to them. If no leaf node is eligible for splitting the algorithm terminates. Some common cost measures are: the mean quantization error associated with the node and the number of input vectors assigned to the node. The experiments in this paper used the mean quantization error.
The leaf splitting step varies according to the type of vector quantizer implemented. TSVQ uses a 2-means approach for computing the centroids of the two child nodes and for assigning data to the child nodes. A k-d tree, on the other hand, uses axis parallel splits. In the k-d tree implemented here, the mean value of the node's component with the largest variance is used to split the node. After a leaf is split, k_0 is incremented by one.
The above tree building procedure is $O(mN\log_2 k)$, where N is the number of input vectors, m is the number of dimensions in an input vector, and k is the maximum or desired codebook size (number of leaves in the tree).

3 Learning the Topology

The topological structure of the input data distribution can be learned as a post-processing step to the tree building stage. The method proposed here can be seen as a hierarchical extension of OTPMS [3]. The OTPMS algorithm works as follows: Given a codebook of size k, constructed by a vector quantizer, for each input vector find the two nearest codevectors to the input and link them. OTPMS is optimum with respect to some of the most commonly used measures of topological quality [3]. However, it requires $O(Nkm)$ computations and does not scale well for large codebooks. If $k \sim N$ it requires $O(N^2m)$ processing. In order to mitigate this, it is proposed here the use of a hierarchical vector quantizer to help speed up the computation of the 2-NN

problem, that is, finding the two nearest codewords for each input vector. Although there are many different and efficient approaches to speed up nearest neighbor queries [1], for the experiments in this paper a simple strategy was used: For each row go down the tree using at most p paths and then connect the two leaves closest to the input vector. This procedure is $O(Nmp\log_2 k)$. For $k \sim N$ it becomes $O(Nmp\log_2 N)$. A value of p less than 10 seems to work well for reasonably large codebook sizes. In practice, p equal 4 or 5 produced good results for codebook sizes with a couple of thousand codevectors.
Applications such as link analysis and nearest neighbor queries may require the creation of topological structures where $k \sim N$. For these cases, there are not enough data per leaf node for OTPMS to capture the structure of the manifold containing the input vectors. This problem is shared by most topology learning algorithms. In order to address this we propose a strategy based on the following ideas: generate random data constrained to small volumes of the input space following the local shape of the manifold containing the data, and use the topology at higher layers of the tree as a guide for the topology at lower layers. The random sampling is used to learn local relationships in the spirit of OTPMS. If the sample is constrained to the ranges in the centroids of the nodes in the subtree of a given baseline graph node, it is expected that the sampling will be contained on a hyper-rectangle with dimensions adapted to the local shape of the manifold containing the data. The use of the topological connections at the higher layers of the tree as guides to the connectivity at lower levels can be seen as a smoothing or regularizing effect that compensates for the small sample size and help learn the overall structure. The following steps are proposed for extending the algorithm to cases where $k \sim N$:

1. Create baseline graph: Identify a level of quantization in the tree structure quantizer for which the nodes have enough support to capture well the topology of the data. This can be accomplished by selecting all nodes in the tree where $C_j < n$ and $C_{d(j)} \geq n$, where C_j is the number of inputs assigned to node j, $d(j)$ is one of the indices of node j two children, and n is a user defined parameter. Apply OTPMS to this set of nodes to construct a baseline graph. Utilize the tree structure for speeding up the 2-NN search. In this paper, the multi-path approach is used in all the experiments. This is a $O(Nmp\log_2 k_b)$ operation, where k_b is the number of nodes in the baseline graph. This step is quite robust to the value used for the parameter n. In practice n as low as 10 produced good results.

2. Link subtree: For each j node in the baseline graph generate r random vectors. The components of the random vectors should be between the minimum and the maximum values found for these components in the leaf nodes in the subtree rooted at the respective baseline graph node. Combine the r random vectors with the centroid values for the leaf nodes in the subtree. For each row in the combined set find the 2-NN leaf nodes s_1 and s_2 in the subtree and link them. Assign a weight of $1/\text{dist}(s_1, s_2)$ to the link, where $\text{dist}(a, b)$ is a distance function (usually the Euclidean metric can be used). This is a $O(k_b mpr\log_2 n)$ operation. For $r \sim n$ and $k_b \sim N/n$ then it becomes a $O(Nmp\log_2 n)$ operation.

3. Create long-range links: For each pair of nodes (u_1, u_2) connected by a link in the baseline graph and for each leaf s_1 in the subtree rooted in u_1, find the closest leaf node s_2 in the subtree rooted in u_2. If $1/\text{dist}(s_1, s_2)$ is greater than the smallest weight amongst the links containing either s_1 or s_2 then create a link between s_1 and s_2. If s_2 was already linked to a node in the subtree rooted at u_1 then keep the link with the smallest weight. This is a $O(0.5k_b nmpl\log_2 n)$ operation, where l is the average number of links for nodes in the baseline graph. For $k_b \sim N/n$ it becomes $O(0.5Nmpl\log_2 n)$

The approach proposed above takes three parameters: p, n, and r. In practice, setting $r = n$ works well and eliminates the need of a free parameter.

The ability of VA-graph to learn the topology of the input distribution using the algorithm described above is illustrated in Figures 2 and 3. k-d tree was the vector quantizer used in these examples. Figure 2 shows, for $k << N$, how the algorithm can learn a topological structure even when the dimensionality of the data distribution varies in different areas of the input space. The quality of the graph learned by VA-graph was assessed using Kiviluoto's topographic error [12]:

$$\varepsilon = \frac{1}{N}\sum_{i=1}^{N} u(\mathbf{x}_i), \tag{1}$$

where $u(\mathbf{x}_i) = 1$ if the first and second best matching units (codevectors) are not adjacent, otherwise zero. Both graphs in Figure 2 had almost no topographic errors. Figure 3 shows that VA-graph can construct a reasonable graph containing one leaf node per input vector ($k = N$) for a relatively sparse sample.

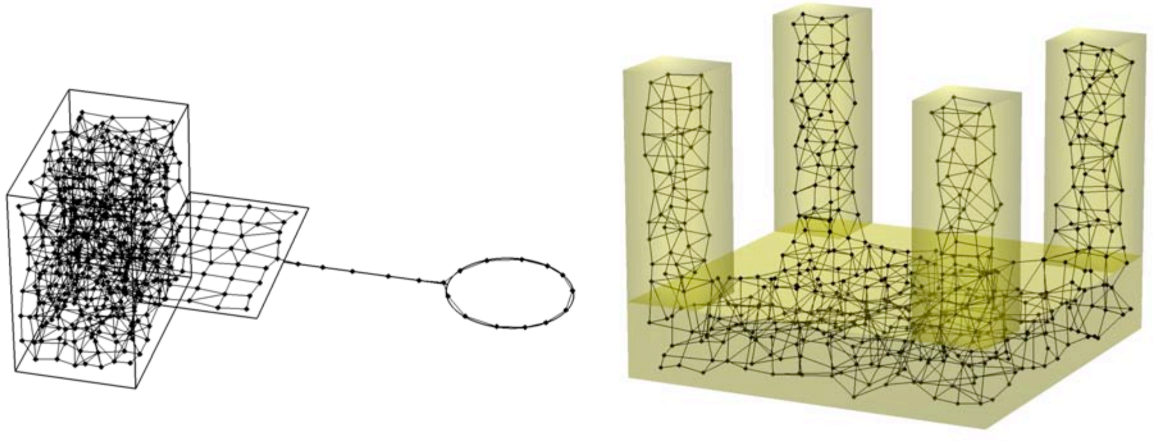

Figure 2: Left: VA-graph adapts to an input distribution with different dimensionalities in different areas of the input space, ε = 0.004. Right: Final topology configuration for another 3D dimensional distribution, ε = 0.007.

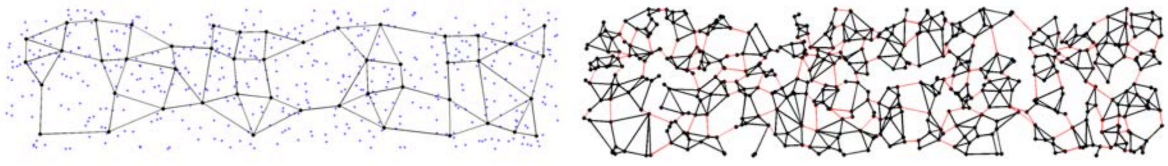

Fig. 3. Building a graph with one node per input vector. Left: Input distribution with 500 vectors and the baseline graph obtained with n = 15. Right: Final graph with one leaf node per input. The thick lines represent the links produced within each major group by the random rows created using the proposed method (r = 10). The thin lines are the links created across groups using the links in the baseline graph as guides.

The method described above can be easily extended to learn a hierarchical topological structure. By constraining the links to certain levels of the tree, it is possible to obtain a coarse to fine description of the structure of the data distribution. The weights on the links learned with the above approach can also be used for pruning the graph and for data exploration.

Once a topology has been learned it can be used to enhance the search for the BMU in the tree vector quantizer. The search process on the neighborhood graph follows the approach in [1]. Search is accomplished in two steps. First, the tree is traversed (single path) until a leaf is reached. Next, the nearest neighbors in the graph to the leaf node are visited to determined if a better codebook can be found. The best matching node is selected for further expansion, in which case its nearest neighbors in the graph are visited. The search through the graph is halted after a certain number of nodes have been visited or a given number of expansions s has taken place. The second stopping criterion is the one used in the experiment section of the paper. The processing for this approach is $O(Nm\log_2 N + Nms)$ for TSVQ and $O(N\log_2 N + Nms)$ for k-d tree.

4 Experiments

This section illustrates the performance of VA-graph on two vector quantization experiments. The experiments used the Letters and the Corel Image Features datasets from the UCI Machine Learning Archive[1] and UCI KDD Archive[2] respectively. For all the results below, the following labels were used: k-d tree represents a tree search for a k-d tree quantizer, TSVQ describes a tree search using a TSVQ quantizer, FS indicates a full search over the codebook produced by either tree quantizer (k-d tree or TSVQ), VA-g represents results for a topology learned using the multi-path algorithm described in Section 3 (p = 4, n = 10, r = 0, and s = 1), VA-gO indicates results using a topology learned with OTPMS. The full search case (FS) is the smallest quantization error possible for the codebook created by the vector quantizers in the experiments. The topology for the VA-gO results is the optimal topology that can be learned for a particular experiment given the codebook produced by the tree quantizer used in the experiment.
The Letters dataset has 20,986 rows and 17 attributes. Figure 4 shows the distortion (SSE, Sum of Squared Errors) for two sets of studies for different codebook sizes. In the first study, a k-d tree was used as the vector quantizer. In the second, TSVQ was used as the vector quantizer. The graphs clearly illustrate the benefit of using VA-graph with the topological information. In the experiments, the results for the approximated topology using the simple multi-path method are very close to those for the optimal topology. In addition, the performance of the quantizers leveraging the topological structure approaches that of the full search for both the k-d tree and TSVQ even for s = 1. Finally, although not reported in the graphs, the average number of extra nodes searched by VA-graph is at most 13 across all codebook sizes and vector quantizers in this set of experiments. To place this in perspective, for a codebook size of 256, VA-graph would search on average $13 + \log_2 256 = 21$ nodes versus 8 nodes for the basic vector quantizers and 256 for the full search case.
For the second experiment a subset with 20,000 rows and 89 attributes of the Corel dataset was used. The results, illustrated in Figure 5, follow the same trends seen for the Letters dataset. Again, the average number of extra nodes searched by VA-graph is at most 13 across all codebook sizes and vector quantizers used with the Corel dataset. Furthermore, for a codebook size of 1024, 97.8% (k-d tree case) and 99.7% (TSVQ case) of the nearest neighbor codevectors were within four hops from the leaf selected by the basic quantizers. For smaller codebook sizes, a higher percentage of the nearest neighbor codevectors were within fewer hops from the initial leaf node.

[1] http://www.ics.uci.edu/~mlearn/MLRepository.html
[2] http://kdd.ics.uci.edu/

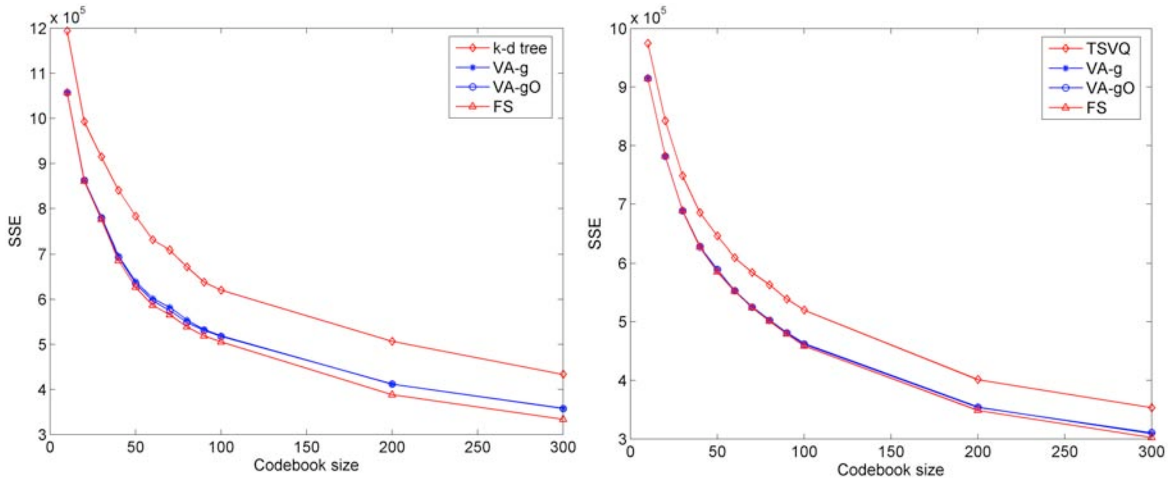

Figure 4: Sum of Squared Errors (SSE) as a function of codebook size for the Letters dataset. Left chart: k-d tree was used as the vector quantizer. Right chart: TSVQ was used as the vector quantizer.

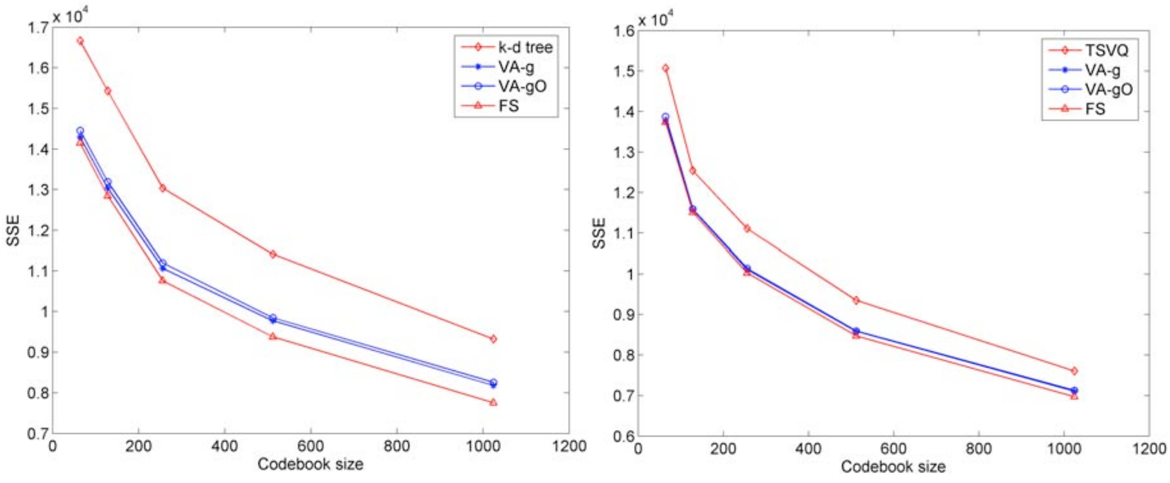

Figure 5: Sum of Squared Errors (SSE) as a function of codebook size for the Corel dataset. Left chart: k-d tree was used as the vector quantizer. Right chart: TSVQ was used as the vector quantizer.

It should be noted that it is possible to improve the performance of VA-graph by expanding the search in the graph to include more nodes ($s > 1$).

5 Conclusions

This paper introduced VA-graph, a new vector quantization algorithm that is capable of learning the topology of the input distribution. The approach presented here can be easily adapted for learning topologies using other hierarchical structures. The random sampling method discussed in Section 3 can be used to allow other topology learning algorithms to learn graphs with one node per input row.
VA-graph is also a promising algorithm for use in an indexing system. Scalar and vector quantization approaches have gained attention during the last years as an approach for indexing high dimensional vector spaces for exact and approximate similarity queries [8].

The current work can be expanded in a couple of directions, including: an online version of the algorithm, exploration of indexing applications, use of alternative nearest neighbor searching strategies in the tree [1, 2] in place of the multi-path approach, and applications to link analysis, data projection, and clustering.

References

1. S. Arya and D. M. Mount (1993), Algorithms for fast vector quantization. In J. A. Storer and M. Cohn, editors, *Proceedings of DCC 93: Data Compression Conference*, p. 381-390, IEEE Press.
2. S. Bader, F. Maire, and F. Wathne (2004), Fast indexing of codebook vectors using dynamic binary search trees with fat decision hyperplanes, *Studies in Fuzziness and Soft Computing*, **vol. 152**, Springer.
3. J. Bruske and G. Sommer (1997), Topology representing networks for intrinsic dimensionality estimation, *ICANN*, p. 595-600.
4. M. Campos and G. Carpenter (2001), S-TREE: Self-organizing trees for data clustering and online vector quantization, *Neural Networks,* **vol. 14**, p. 505-525.
5. C-C. Chang and T. S. Chen (1997), New trees-structured vector quantization with closest-coupled multipath searching method, *Optical Engineering*, **vol. 36**, p. 1713-1720.
6. E. Cuadros-Vargas and R. F. Romero (2002), A SAM-SOM family: Incorporating spatial access methods into constructive self-organizing maps. In *Proceedings IJCNN'02, Intl. Joint Conference on Neural Networks*, p.1172-1177, IEEE Press.
7. M. Dittenbach, A. Rauber, and D. Merkl (2001), Recent advances with the growing hierarchical self-organizing map. In N. Allinson, H. Yin, L Allinson, J. Slack, editors, *Advances in Self-Organizing Maps: Proceedings of the 3rd Workshop on Self-Organizing Maps*, Lincoln, England, Springer.
8. H. Ferhatosmanoglu and E. Tuncel (2000), Vector approximation based indexing for non-uniform high dimensional data sets. In *Proceedings of the Ninth International Conference on Information and Knowledge Management*, Virginia, United States, p. 202-209.
9. B. Fritzke (1994), Growing cell structures – A self-organizing network for unsupervised and supervised learning, *Neural Networks*, **vol. 7**, p.1441-1460.
10. B. Fritzke (1995), A growing neural gas network learns topologies. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems* 7, p. 625-632, MIT Press, Cambridge MA.
11. A. Gersho and R. Gray (1992), *Vector Quantization and Signal Processing,* Kluwer Academic Publisher.
12. K. Kiviluoto (1996), Topology preservation in self-organizing maps. In *Proceedings of ICNN'96, International Conference on Neural Networks*, **vol. 1**, p. 294-299, IEEE Neural Networks Council.
13. T. Kohonen (1982), Self-organized formation of topologically correct feature maps, *Biological Cybernetics*, **vol. 43**, p. 59-69.
14. P. Koikkalainen (1994), Progress with the tree-structured self-organizing map. In *Proceedings ECAI'94, 11th European Conference on Artificial Intelligence*, p. 211-215.
15. T. Martinez and K. Schulten (1993), A "neural gas" network learns topologies. In T. Kohonen, K. Makisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, **vol. 7(3)**, p. 507-522.
16. J. Tenenbaum, V. de Silva, and J. Langford (2000), A global geometric framework for nonlinear dimensionality reduction, *Science,* **vol. 22**, p. 2319-2323.