Fast codebook searching in a SOM-based vector quantizer for image compression

Arijit Laha · Bhabatosh Chanda · Nikhil R. Pal

Abstract We propose a novel method for fast codebook searching in self-organizing map (SOM)-generated codebooks. This method performs a non-exhaustive search of the codebook to find a good match for an input vector. While performing an exhaustive search in a large codebook with high dimensional vectors, the encoder faces a significant computational barrier. Due to its topology preservation property, SOM holds a good promise of being utilized for fast codebook searching. This aspect of SOM remained largely unexploited till date. In this paper we first develop two separate strategies for fast codebook searching by exploiting the properties of SOM and then combine these strategies to develop the proposed method for improved overall performance. Though the method is general enough to be applied for any kind of signal domain, in the present paper we demonstrate its efficacy with spatial vector quantization of gray-scale images.

Keywords Vector quantization · Image compression · Self-organizing map (SOM) · Topology preservation · Fast codebook search

1 Introduction

With the advent of World Wide Web and proliferation of multimedia contents, data compression techniques have gained immense importance. Data compression has become an enabling technology for efficient storage and transmission of multimedia data. Vector quantization [8] is known to be an efficient method for data compression. The performance of a vector quantizer (VQ) depends on two factors, the quality of the codebook and the time required for codebook searching at the encoding stage. The self-organizing map introduced by Kohonen [15] can be used for constructing a good quality codebook. In this paper we propose a method which exploits the properties of SOM for fast (non-exhaustive) codebook searching without significant sacrifice in reproduction quality. Here we work with image data only, though the method can be applied to other kinds of data also.

A vector quantizer [8] $\mathcal{Q}$ of dimension k and size S can be defined as a mapping from data vectors (or "points") in k-dimensional Euclidean space, $\mathcal{R}^k$ into a finite subset $\mathcal{C}$ of $\mathcal{R}^k$. Thus,

$$\mathcal{Q} : \mathcal{R}^k \rightarrow \mathcal{C}$$

where $\mathcal{C} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_S\}$ is the set of S reproduction vectors, called a *codebook* of size S and each $\mathbf{y}_i \in \mathcal{C}$ is called a *codevector* or *codeword*. For each $\mathbf{y}_i$, $i \in \mathcal{I} = \{1, 2, \ldots, S\}$ is called the index of the codevector and $\mathcal{I}$ is the index set. Encoding a data vector $\mathbf{x} \in \mathcal{R}^k$ involves finding the index j of the codevector $\mathbf{y}_j \in \mathcal{C}$ such that $\|\mathbf{x} - \mathbf{y}_j\| \leq \|\mathbf{x} - \mathbf{y}_i\| \; \forall i \neq j$ and $i, j \in \mathcal{I}$. The decoder uses the index j to look-up the codebook and generates the reproduction vector $\mathbf{y}_j$ corresponding to $\mathbf{x}$. The distortion measure $d(\mathbf{x}, \mathbf{y}_j) = \|\mathbf{x} - \mathbf{y}_j\|$ represents the penalty of reproducing $\mathbf{x}$ with $\mathbf{y}_j$. If a VQ minimizes the average distortion, it is called the optimal VQ of size S.

The task of designing a VQ is essentially finding a set of codevectors for the encoder that partitions $\mathcal{R}^k$ into S disjoint regions or cells $R_i, i \in \mathcal{I}$, where $\mathcal{I} = \{1, 2, \ldots, S\}$ is the index set, such that,

$$R_i = \{\mathbf{x} \in \mathcal{R}^k : \mathcal{Q}(\mathbf{x}) = \mathbf{y}_i\}.$$

The decoder may be implemented as a simple lookup table that selects the reproduction vector corresponding to an index value produced by the encoder.

Vector quantization has been used for image compression successfully by many researchers [2,3,10–12,17,24,25]. The oldest as well as most commonly used method for codebook generation is the generalized Lloyd algorithm (GLA) [22], also known as *k-means* algorithm. GLA is an iterative gradient descent algorithm that tries to minimize an average squared error distortion measure. Design of an optimal VQ using GLA has been proposed and studied in [21]. Some of the other methods studied include fuzzy c-means algorithms [4] and several other fuzzy vector quantization techniques [12], stochastic relaxation techniques and simulated annealing by Zeger et al. [28].

Kohonen's SOM [15] is a neural network technique exhibiting several interesting properties and consequently has attracted attention of the researchers in the field of vector quantization [2,24,27]. SOM is easy to implement and has found numerous applications (for a list of references see [13,26]). A discussion of general features of SOM algorithm and their suitability with respect to design of a codebook can be found in [19]. In [6], Chang and Gray introduced an on-line technique for VQ design using stochastic gradient algorithm which can be considered a special case of the SOM algorithm and it is shown to perform slightly better than GLA. Nasrabadi and Feng [24] also used SOM for VQ design and demonstrated performance better than or similar to GLA. Yair et al. [27] used a combination of SOM and stochastic relaxation and obtained consistently better performance than GLA. Amerijckx et al. [2] used Kohonen's SOM algorithm to design a VQ for the coefficients of discrete cosine transform of the image blocks and further compressed the output of VQ encoder using entropy coding. In [19], the present authors used SOMs for designing VQ and proposed a novel surface-fitting method for refining the codebook for achieving better psychovisual quality of the reproduced image. However, all the above works use SOM for generating good codebooks. To the best of our knowledge, no attempt has been made till date to exploit the properties of SOM directly for designing fast codebook searching methods.

In Sect. 2 of the paper, we describe the codebook design scheme using the SOM. Section 3 covers the encoding methods in detail, we report the experimental results in Sect. 4 and Sect. 5 concludes the paper. Throughout this paper we use *peak signal to noise ratio* (PSNR) [8] as the performance measure of the VQ.

2 Codebook design

To generate the image (feature) vectors an $h \times w$ image is divided into $p \times q$ blocks. The pixel values in the $p \times q$ image block are presented in *row-major* fashion to form the feature vector $\mathbf{x}$. Thus, each block is represented by a vector $\mathbf{x} \in \mathcal{R}^k$, where $k = pq$ and each component x_i of $\mathbf{x}$ represents the value of (lexicographically ordered) i-th pixel in the block. Further, to enhance the reproduction quality, we use the mean-removed or residual vectors [8,19] in the computations.

The SOM is a two-layer *competitive learning* network, where the output nodes are arranged over a regular (usually square or hexagonal) display (also known as output) lattice. During the training, when a training vector is presented to the input layer, all the output nodes receive the same input through weighted connections. The node having the weight vector which matches best with the input vector, becomes the *winner node* for that input. Next, the winner node and its neighboring (according to their positions on the display lattice) nodes are updated to move closer to the input. The process is repeated several times over the whole training set. At the end of the training the distribution of the weight vectors of the SOM nodes resembles to a high degree to that of the training data set. This is known as the *density matching* property of SOM. This property enables the SOM algorithm to generate a high quality codebook. Further, the trained SOM exhibits the property of *topology preservation*. Due to topology preservation, the input vectors close to one another, usually have best matching weight vectors, i.e, winner nodes those are either the same or near-by over the display lattice.

Including the neighbors of the winner node in update process (also known as *neighborhood update*) is the most distinctive feature of SOM. It sets SOM apart from conventional clustering algorithms (e.g., k-means, fuzzy c-means) and other competitive learning networks (e.g., adaptive resonance theory (ART) based networks). The neighborhood update process causes the topology preservation and density matching properties of SOM while retaining the good clustering ability common in competitive learning networks. It can be thought as the emergence of a global (network-wide) ordering of the nodes through local learning (within the neighborhood). The trained SOM generates a topology preserving mapping of the input data onto the network lattice. This is the property of SOM that we exploit the most in the methods proposed in this paper. Further, density matching property enables the SOM nodes to capture the statistics of the training data. For a detailed general description of SOM algorithm see [15] and for the same in context of vector quantization see [19].

Here we design the codebook by training a 2-D SOM with a set of training vectors from a set of training images. As explained above, after training, the set of weight vectors

of SOM nodes form the basic codebook (or *super codebook*). Since each of the weight vectors is uniquely associated with a node in the display lattice, they can be indexed by 2-tuples of the form (x, y) representing the positions of the node in the SOM lattice.

3 The encoder

Apart from finding a set of good codevectors, the VQ performance depends crucially on the size of the codebook and the dimension of the vectors. The input space can be represented to a finer degree by increasing the codebook size. Further, the VQ works by directly exploiting the intra-block statistical dependencies. Therefore, a larger block enables the quantizer to exploit the statistical dependency existing in the data to a greater degree [8]. However, using large number of codevectors and high dimensional vectors introduces a serious performance bottleneck on the part of the encoder. Given any vector to be encoded, the encoder has to search the codebook for the best matching codevector. Thus, to make full search of a large codebook with high dimensional vectors the encoder hits a serious complexity barrier.

To circumvent this difficulty many techniques have been developed. These techniques apply various constraints on the structure of the codebook and use suitably altered encoding algorithm. These techniques can be divided into two major categories. The *memoryless* VQs perform encoding/decoding of each vector independently. Well known examples in this category include tree-Structured VQ (TSVQ), classified VQ, shape-gain VQ, multistage VQ, hierarchical VQ etc. Methods in the other category depend on the past information (thus having memory) for their operations. Prominent members include predictive VQ, recursive or feedback VQ, finite state VQ (FSVQ) etc. Comprehensive reviews of these methods can be found in [1,8,9]. All these techniques are aimed at reducing the codebook search complexity without a significant loss in reproduction quality. Among some of the recent works, in [20] Lai and Liaw developed a fast codebook searching algorithm based on projection and triangular inequality that uses the features of vectors to reject impossible codes. In [14], a method for designing predictive vector quantizer using deterministic annealing is proposed. An approach to design an optimal FSVQ is proposed in [7]. Interestingly, in this paper the codebook is built using the SOM algorithm. A novel method of designing TSVQ can be found in [5].

If SOM is used to generate the VQ codebook, it implicitly introduces some constraints through *neighborhood update* during the training resulting in some interesting properties including *topology preservation*. Due to the topology preservation property, the vectors close in the input space are represented by the weight vectors of the nodes those are close in the SOM lattice. We exploit this property of the SOM in the search method proposed here.

3.1 Searching the codebook

We design and analyze two strategies for searching the codebook in a restricted manner to reduce the search complexity. The first one uses the basic SOM generated codebook and exploits its topology preservation property. Some preliminary results for strategy 1 are reported in [18]. However, in case of **derailment** it performs an *exhaustive* search of the codebook, thus increasing the computational complexity. This shortcoming is overcome by combining it with the strategy 2 proposed here. The second strategy uses a smaller SOM along with the basic codebook. The smaller SOM is trained with the codevectors of the basic codebook and we shall call it Level2-SOM (L2-SOM). Then we propose a combined method that uses the L2-SOM and the basic codebook and utilizes the topology preservation property of the basic codebook, thus exploiting the best features of strategies 1 and 2. It should be noted that the strategies 1 and 2 are also independent methods for searching the codebook. Here we call them strategies simply to differentiate them from the final method that combine both of them. The methods are described below.

3.1.1 Strategy 1: Restricted window search over SOM lattice

This strategy is fashioned after finite state vector quantizers (FSVQ) [8]. FSVQs belong to a more general class of VQs known as *recursive* or *feedback* vector quantizers. Given an input sequence of vectors $\mathbf{x}_n$, $n = 1, 2, \cdots$, the encoder produces both a set of code indexes $u_n = 1, 2, \ldots$, and a sequence of *states* $\mathcal{S}_n = 1, 2, \ldots$, which determines the behavior of the encoder. A FSVQ is characterized by a finite set of K states $\mathcal{S} = \{1, 2, \ldots, K\}$ and state transition function $f(u, s)$. Given the current state s and last code index u, when presented with the next vector $\mathbf{x}$ of the input sequence, the FSVQ enters a new state determined by the state transition function $f(u, s)$. Associated with each state s is a smaller codebook $\mathcal{C}_s$, known as state codebook for state s. The search for the codevector for the new input $\mathbf{x}$ is now restricted to $\mathcal{C}_s$, which is much smaller than the full codebook, also known as super codebook, $\mathcal{C} = \bigcup_{s \in \mathcal{S}} \mathcal{C}_s$. Obviously the major challenge in designing a FSVQ involves the design of the state transition function and finding the codebooks corresponding to each state.

One of the simplest and popular techniques for finding a next-state function is called *conditional histogram design* [8]. This approach is based on the observation that each codeword is followed almost invariably by one of a very small subset of the available codevectors. This happens due to existence of

high degree of correlation between successive vectors in an input sequence. Thus the performance of the VQ should not degrade much if these small subsets form the state codebooks. So, the training sequence is used to determine the conditional probability $p(j \mid i)$ of generating the j-th codevector following the generation of i-th codevector. The state codebook for state i of size N_i consists of the N_i codevectors $\{\mathbf{y}_i\}$ with highest values of $p(j \mid i)$. However, with this design, especially for data outside the training sequence, the optimal codevector may lie outside the state codebook. Thus often a threshold of matching is used. If no codevector with match exceeding the threshold is found in the state codebook, the system is said to 'derail' [7]. In such a situation usually an exhaustive search over the super codebook is performed for re-initializing the search process.

In the codebook generated using SOM algorithm, the nodes are located in a regular 2-D lattice. Each node in turn is an image of a prototype or codevector in the lattice space. Thus, the 2-tuple describing the position of the corresponding node in the SOM lattice can uniquely identify a codevector. Further, due to topology preservation property of SOM, the nodes nearby on the lattice plane encode vectors with higher similarity than the nodes which are located far away on the lattice. In image compression (or more general signal compression) problems the successive source vectors in the input sequence are usually highly correlated. Thus, the best-matching codevectors representing them are also either same or very similar. This in turn is translated into the best-matching codevectors being represented by SOM nodes which are close to each other on the lattice. Thus, if the codevector for an input vector corresponds to a node c on the lattice, then the codevector for an adjacent block is likely to be within a small neighborhood N_c of c. In other words, in the FSVQ sense, the set of codevectors whose images are contained in the neighborhood N_c form the state codebook associated with $f(t-1, \mathcal{S}(t))$, where c is the codevector chosen for $(t-1)$-th source vector. It is to be noted that here *no extra computational effort is required to form the set of state codebooks*. They can be identified unambiguously due to the regular lattice structure of the SOM output space, and efficacy of their use is based on the topology preservation property of SOM network.

Now to formulate the search method we define a search window size $s_h \times s_w$ and a quality threshold T. For the first vector an exhaustive search is performed and the index in the form of (x_1, y_1) pair is generated. For the subsequent k-th vector $\mathbf{x}_k$ the search is performed among the nodes on the SOM lattice falling within the search window centered at the node (x_{k-1}, y_{k-1}) (we call it the previous index). Due to topology preservation property of SOM and the characteristic similarity between neighboring image blocks there is a high possibility of finding a good match within the window. If the best match within the window exceeds the threshold T, then the index (x_k, y_k) of the corresponding node is selected; otherwise, rest of the codebook is exhaustively searched to find the best match. Now (x_k, y_k) becomes the previous index for $(k+1)$-th vector. Thus we can identify the full codebook generated by the SOM as the super codebook of FSVQ, the current states as the predecessor index (x_{k-1}, y_{k-1}), and the state codebook as the set of codevectors within the SOM lattice window of size $s_h \times s_w$ centered at (x_{k-1}, y_{k-1}).

There are two issues to be taken care of. The first one concerns the case when the previous index is close to the boundary of the SOM lattice and the window cannot be centered at it. In such a case, we align the edge of the window with the edge of the lattice. The second issue relates to the image blocks at the beginning of a row of blocks, i.e., the blocks at the left edge of the image. Since the image blocks are encoded in a row-major fashion, for other blocks the previous index corresponds to the index for the immediate left block. For the leftmost blocks we use the indexes of the leftmost blocks in the previous row (i.e., the block at the top of current block) as the previous index.

There is also a design issue regarding the choice of the window size as well as the quality threshold T. For a given threshold T, smaller window sizes will reduce the window search time, but the instances of derailment will be higher. Again, for a fixed window size, the higher the threshold, the more is the instances of derailment. It is difficult to predict theoretically suitable choices of these parameters since that will require the knowledge about the distribution of the codebook vectors over the SOM lattice as well as the distribution of the image vectors being quantized. In the 'Experimental results' section we have presented some empirical studies regarding the choice of these parameters. Below we present the strategy 1 in an algorithmic form. Searching for the best-matching code for input vector $\mathbf{x}_i$ among a set of codevectors S is denoted as $(x_i, y_i) \leftarrow BestMatch(\mathbf{x}_i : S)$, where the output (x_i, y_i) is the SOM lattice position of the best matching code.

Algorithm: ***Strategy 1***

Input:

SOM codebook $\mathcal{C}$ of size $m \times n$
Search window size $s_h \times s_w$
Quality threshold T as PSNR value over image block
Sequence $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_N\}$ of image vectors to be quantized.

Begin

For $k \leftarrow 1$ to N
do
- if k = 1 then
  - $(x_k, y_k) \leftarrow BestMatch(\mathbf{x}_k : \mathcal{C})$;

- else
  - Compute the search window over the SOM lattice w. r. t. (x_{k-1}, y_{k-1});
    Let the set of codevectors within the window be $\mathcal{W}$
  - $(x_k, y_k) \leftarrow BestMatch(\mathbf{x}_k : \mathcal{W})$;
  - if $PSNR(\mathbf{y}_{(x_k,y_k)}, \mathbf{x}_k) > T$ then
    - $(\hat{x}_k, \hat{y}_k) \leftarrow BestMatch(\mathbf{x}_k : \mathcal{C} - \mathcal{W})$;
    - if $PSNR(\mathbf{y}_{(x_k,y_k)}, \mathbf{x}_k) > PSNR(\mathbf{y}_{(\hat{x}_k,\hat{y}_k)}, \mathbf{x}_k)$ then
      - $(x_k, y_k) \leftarrow (\hat{x}_k, \hat{y}_k)$;
- $k \leftarrow k + 1$;

End
Output:

Sequence of code indexes $\{(x_k, y_k) \mid 1 \leq k \leq N\}$

3.1.2 Strategy 2: Codebook searching with L2-SOM

The efficacy of strategy 1 depends on its rate of success in finding the successor code within the search window surrounding the predecessor code. Failing to do so requires exhaustive search of the codebook. Since the window size is significantly smaller than the size of the full SOM codebook, even a relatively small number of exhaustive searches can affect the overall search complexity significantly. The cases of derailments can be attributed to several factors. Firstly, though neighboring image blocks are more often than not highly correlated, there may be significant number of dissimilar neighboring blocks in the image. Secondly, the quality of the topology preservation achieved by the SOM degrades if there is significant difference between the dimensions of the data and the display lattice [23]. This is often termed in SOM literature as the *dimensional mismatch*. In this case, we train a 2-D SOM with a complex data of quite high dimension. Thirdly, as discussed earlier, the parameters of the algorithm, the *window size* and the *quality threshold* affect the frequency of derailments. What is more interesting, a significant fraction of the derailments could be false derailments. These cases happen when the best matching code is within the search window, but the matching quality does not satisfy the quality threshold. In that case, though *a full search is performed, the code found during the window search is finally selected*. This can happen frequently if we set the quality threshold reasonably high to ensure good reproduction quality.

From the above discussion, it becomes evident that if a *low-cost* method for addressing the issue of derailment can be found, the codebook search can be made considerably faster. To devise such a method we draw upon the ideas of classified VQ (CVQ) [8] from design perspective and on hierarchical SOM (HSOM) [16] for implementation. In a CVQ the codebook is partitioned into a set of classes. The set of codevectors belonging to a class has a high degree of similarity among its members. Each class has one representative codevector. Searching the codebook for an input vector $\mathbf{x}$ is performed as follows:

- Find the best-matching vector among the *set of class representatives*.
- Select the best-matching codevector from the *partition* corresponding to the best-matching class representative.

Generally in CVQ, various clustering methods are applied to build the partition (i.e., the classes) of the codebook. On the other hand, in neural network literature, the HSOM is used to speed-up the search for the winner node in a large SOM. Several SOMs are arranged in a hierarchical fashion. Each node in a higher level SOM corresponds to a subset of nodes in the larger next level SOM. The winner-search starts with finding the winner node in the highest level SOM and proceeds to the next level by confining the winner-search within the subset of nodes associated with the winner node in the previous layer. The higher layer SOM is often built in a bottom-up fashion by training it with the weight vectors of the next layer SOM.

Here we take a similar approach by training a smaller SOM, which we call the Level-2 SOM (L2-SOM) with the codevectors of the basic SOM codebook. The basic codebook is partitioned into sets $\mathcal{P}_i$s of best matching codevectors for each of the weight vectors of the L2-SOM. In other words, the set of weight vectors $\{\mathbf{w}_i\}$ of L2-SOM induce a partition $\{\mathcal{P}_i\}$ of basic SOM codebook $\mathcal{C}$ such that

$$\left\| d(\mathbf{y}_l - \mathbf{w}_i) \leq d(\mathbf{y}_l - \mathbf{w}_j) \right\| \forall \mathbf{w}_j \in \{\mathbf{w}_i\}, \quad \forall \mathbf{y}_l \in \mathcal{P}_i.$$

These two SOMs can be used independently for implementing a *memoryless* fast codebook searching method on their own. We distinguish this method as 'strategy 2'. The search method can be formulated as follows:

Given an input vector $\mathbf{x}$ to be quantized
1. Find the best-matching weight vector for $\mathbf{x}$ from L2-SOM.
2. Select the best-matching codevector for $\mathbf{x}$ from the subset of basic SOM codebook corresponding to the best-matching L2-SOM node.

We present below the strategy 2 in an algorithmic form.

Algorithm: ***Strategy 2***

Input:

L2-SOM
Partition $\{\mathcal{P}_i\}$ of basic SOM codebook $\mathcal{C}$ (i.e., $\mathcal{P}_i \Rightarrow i$ - th node of L2 - SOM,) and $\bigcup_{\mathcal{P}_i \cap \mathcal{P}_{j \neq i} = \Phi} \mathcal{P}_i = \mathcal{C}$
Sequence $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_N\}$ of image vectors to be quantized.

Begin

For $k \leftarrow 1$ to N
do
- $winner_k \leftarrow WinnerSearch(\mathbf{x}_k : L2-SOM)$;
- $(x_k, y_k) \leftarrow BestMatch(\mathbf{x}_k : \mathcal{P}_{winner_k})$;

End
Output:

Sequence of code indexes $\{(x_k, y_k) \mid 1 \leq k \leq N\}$

3.1.3 Combined method: Restricted window search with L2-SOM look-up for re-initialization

As we have stated earlier, we intend to enhance the search performance of strategy 1 by augmenting it with a low-cost solution for re-initialization of the search after a derailment occurs. To find a good codevector after a derailment, a memoryless search technique (of which the exhaustive codebook search is the simplest but most expensive alternative) needs to be employed. The index of the current codevector then serves as the *previous index* for the next input, and thus the *restricted* codebook search can be resumed.

In strategy 1, we have used the *exhaustive* codebook search for addressing the derailment problem. However, the strategy 2 described above is a *memoryless* and *non-exhaustive* codebook search method with significantly lower search complexity. Thus, this search technique can be used very effectively to address the derailment problem in strategy 1. Therefore, we design our final combined method by performing the *SOM window search* as in 'strategy 1', for finding the codevectors in general, while we employ the *L2-SOM look-up* based codebook search for (1) *finding the first codevector* and (2) finding *the codevector when derailment occurs*. The method is depicted below in an algorithmic form.

Algorithm: ***Combined (Final) Method***

Input:

SOM codebook $\mathcal{C}$ of size $m \times n$
Search window size $s_h \times s_w$
Quality threshold T as PSNR value over image block
L2-SOM
Partition $\{\mathcal{P}_i\}$ of basic SOM codebook $\mathcal{C}$
Sequence $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_N\}$ of image vectors to be quantized.

Begin

For $k \leftarrow 1$ to N
do
- if k = 1 then
  - $winner_k \leftarrow WinnerSearch(\mathbf{x}_k : L2-SOM)$;
  - $(x_k, y_k) \leftarrow BestMatch(\mathbf{x}_k : \mathcal{P}_{winner_k})$;
- else
  - Compute the search window over the SOM lattice w. r. t. (x_{k-1}, y_{k-1});
    Let the set of codevectors within the window be $\mathcal{W}$
  - $(x_k, y_k) \leftarrow BestMatch(\mathbf{x}_k : \mathcal{W})$;
  - if $PSNR(\mathbf{y}_{(x_k, y_k)}, \mathbf{x}_k) > T$ then
    - $winner_k \leftarrow WinnerSearch(\mathbf{x}_k : L2-SOM)$;
    - $(x_k, y_k) \leftarrow BestMatch(\mathbf{x}_k : \mathcal{P}_{winner_k})$;
    - if $PSNR(\mathbf{y}_{(x_k, y_k)}, \mathbf{x}_k) > PSNR(\mathbf{y}_{(\hat{x}_k, \hat{y}_k)}, \mathbf{x}_k)$ then
      - $(x_k, y_k) \leftarrow (\hat{x}_k, \hat{y}_k)$;
- $k \leftarrow k + 1$;

End
Output:

Sequence of code indexes $\{(x_k, y_k) \mid 1 \leq k \leq N\}$

3.1.4 The search complexities

In strategy 1, the reduction in search complexity depends on the success rate of finding a suitable codevector in the search window without derailment which necessitates full codebook search. Let p_d be the probability of derailment for a vector $\mathbf{x}_i \in X$ to be coded. Then the number of search for encoding X is given by

$$N\,|\mathcal{W}| + |\mathcal{C}|\,p_d, \tag{1}$$

where $|\mathcal{W}|$ and $|\mathcal{C}|$ are the sizes of search window and codebook respectively. Clearly, p_d depends on the window dimension $s_w \times s_h$, the SOM dimension $m \times n$, the quality threshold T and most critically on the sequence of input vectors x_i. There is no straightforward way of computing it. Only a rough estimate can be formed empirically using the training data.

For strategy 2, the search complexity depends on the number of nodes M of the L2-SOM. Let us assume that all nodes of the L2-SOM are equally likely to be winner. Then each node of the L2-SOM has $\left(\frac{|\mathcal{C}|}{M}\right)$ number of codevector of the main codebook. Thus for an input vector $\mathbf{x}_i \in X$, the cost of

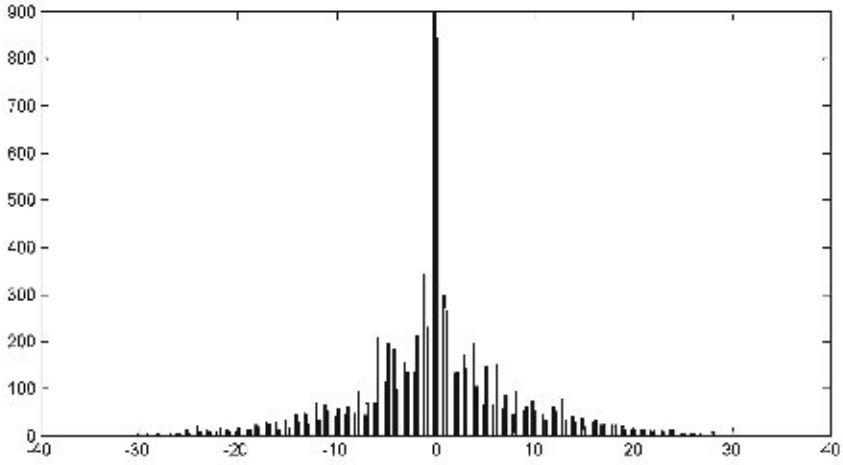

Fig. 1 Grouped histogram of the index offsets for exhaustive search vector quantization of the Lena image. For each of the offset values, the frequency of x and y offsets are shown side-by-side

finding a codevector is given by

$$M + \frac{|\mathcal{C}|}{M}, \tag{2}$$

and cost for encoding X is

$$N \times \left(M + \frac{|\mathcal{C}|}{M} \right). \tag{3}$$

Based on the above analysis, we may express the search complexity of the combined method as:

$$N\,|\mathcal{W}| + \left(M + \frac{|\mathcal{C}|}{M} \right) p_d. \tag{4}$$

It is evident that since $1 << M << |\mathcal{C}|$, for all practical purpose, $\left(M + \frac{|\mathcal{C}|}{M} \right) << |\mathcal{C}|$. Hence, computation cost given in Eq. (4) is much less than that given in Eq. (1).

Other than search complexity, another interesting characteristic of the above methods can be observed. The basis of the search method(s) developed here is twofold, (1) successive signal blocks to be quantized possess *high degree of similarity/correlation* and (2) as a result of *topology preservation property of SOM*, similar input vectors are represented by weight vectors of same node of SOM or nodes close to each other over the SOM lattice. Therefore, we usually find best-matching codevectors for successive inputs corresponding to nearby nodes over the lattice plane. Thus, if we express the sequence of code indexes as the sequence of 2-tuples of offset values over the SOM lattice, the smaller values will dominate the distribution of the offset values. This might lead to design of a good entropy coding scheme for the index values. This characteristic is likely to be further enhanced when the search is restricted to a smaller window over the lattice as in the strategy 1 and the combined method. In the experimental results (Fig. 4) we have provided a comparison of the distribution of the offset values for the methods discussed here.

3.2 Transformation and Huffman coding of the indexes

For a codebook generated with an $m \times n$ SOM the components of indexes vary from 0 to $m-1$ and 0 to $n-1$ respectively. However, due to the topology preservation property, even without restricting the search, neighboring image blocks are mapped into nearby nodes in the SOM lattice. So instead of using the absolute values of the coordinates, if we express the index of a vector in terms of offsets from the previous index, i.e., $(x_k^o, y_k^o) = (x_k - x_{k-1}, y_k - y_{k-1})$ then x_k^o and y_k^o are more likely to have small values. Figure 1 depicts the histograms of the index components expressed as offset values for 512×512 Lena image for a VQ using 32×32 SOM with 8×8 blocks (i.e., 64 dimensional vectors) employing exhaustive search. For restricted search the distribution is expected to have even sharper peak around 0.

Clearly coding of indexes in terms of offset values will allow us to perform efficient Huffman coding. However, using offset values stretches the range of index component values from $-(m-1)$ to $(m-1)$ and from $-(n-1)$ to $(n-1)$ respectively. Hence we need more code words. We can restrict the range of the offset values within 0 to $(m-1)$ or $(n-1)$, whichever is greater, if the index values are further transformed into $(x_k^{o'}, y_k^{o'})$ as follows:

- If $x_k^o \geq 0$ then $x_k^{o'} = x_k^o$
- Otherwise $x_k^{o'} = x_k^o + m$
- If $y_k^o \geq 0$ then $y_k^{o'} = y_k^o$
- Otherwise $y_k^{o'} = y_k^o + n$

Figure 2 depicts the histogram corresponding to the transformed offsets for the Lena image (corresponding to the index offsets shown in Fig. 1).

The decoder computes the index values (x_k, y_k) from $(x_k^{o'}, y_k^{o'})$ as follows:

- If $x_k^{o'} \leq (m - x_{k-1})$ then $x_k = x_{k-1} + x_k^{o'}$
- Otherwise, $x_k = x_{k-1} + (x_k^{o'} - m)$
- If $y_k^{o'} \leq (n - y_{k-1})$ then $y_k = y_{k-1} + y_k^{o'}$

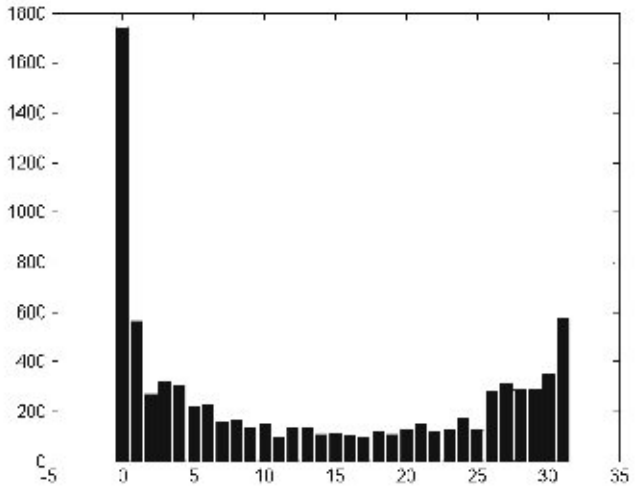

Fig. 2 Histogram of the transformed offset values for exhaustive search vector quantization of the Lena image. The frequencies depicted for each value is the sum of x and y frequencies

– Otherwise, $y_k = y_{k-1} + (y_k^{o'} - n)$

In our scheme we have used mean-removed vectors. Thus we need to store/transmit the block average corresponding to each index. For achieving efficient Huffman coding of the average values also, here we use a difference based value transformation scheme developed by the authors and reported in [19]. The scheme exploits closeness of the average values of neighboring blocks due to their high correlation.

4 Experimental results

As we have stressed earlier in the paper, we expect the proposed method to work efficiently with a *large codebook* consisting of *high dimensional vectors*. Our experimental protocols have also been selected to reflect the above characteristics of the VQ. In our experiments, we have trained a 32×32 SOM with training vectors generated from a composite of sixteen 256 level images each of size 256×256. This results in a large codebook with 1,024 codevectors. Further, we use image blocks of size 8×8, which makes the dimension of the vectors 64. We report the test results with three 512×512 images **Lena**, **Barbara** and **Boat**. In the training set 16 images of size 256×256 are used. The experimental setup for the results reported here are as follows: The search window is set to 8×8 and the quality threshold T is set at 30 dB PSNR. A 6×6. Level-2 SOM is trained with the 1024 codevectors in the basic codebook. The experimental results for strategies 1, 2 and the combined method are summarized in Table 1. Note that, the strategies 1 and 2 are fully implementable restricted search methods on their own.

In Table 1 the search complexity is expressed in terms of the number of codevectors examined during the search procedure. For the exhaustive search, for 4096 blocks in a test image the number of codevectors examined is $4096 \times 1024 = 4194304$. For the restricted search methods the complexity is expressed as percentage of the number of codevectors examined with respect to the exhaustive search (shown in parenthesis). It is evident from the results, that compared to the exhaustive search all the three proposed methods substantially decrease the search time without any significant sacrifice in the reproduction quality. The reproduction quality of the SOM-based VQs with exhaustive search are studied in detail and compared with existing methods in [19]. The exhaustive search results are better than those reported in [19] for the same block sizes. This is due to the use of larger codebooks in the current work. The strategy 1 nearly halves the search complexity with negligible decrease in PSNR values for all images. Strategy 2 reduces the search complexity for all the images to 15–16%, i.e., 1/6-th with drop of PSNR 0.37, 0.26 and 0.34 dB for Lena, Barbara and Boat respectively, compared to the exhaustive search. The proposed combined

Table 1 Comparison of VQ with exhaustive search and restricted searches (Strategies 1,2 and the combined method)

Image	Search method	PSNR (dB)	No. of distance calculations (% w.r.t. exhaustive search)	Compression ratio (bpp)
Lena	Exhaustive	28.95	4194304	0.227
	Strategy 1	28.82	1993984 (47.5%)	0.218
	Strategy 2	28.58	687393 (16.4%)	0.226
	Combined	**28.47**	**472071 (11.3%)**	**0.218**
Barbara	Exhaustive	24.37	4194304	0.231
	Strategy 1	24.34	2710144 (64.6%)	0.227
	Strategy 2	24.11	654899 (15.6%)	0.232
	Combined	**24.09**	**604710 (14.4%)**	**0.228**
Boat	Exhaustive	26.97	4194304	0.207
	Strategy 1	26.93	2348224 (56.0%)	0.203
	Strategy 2	26.63	656693 (15.6%)	0.207
	Combined	**26.61**	**512804 (12.2%)**	**0.203**

Fig. 3 The reproduced images for **a** exhaustive search, **b** SOM window search (strategy 1), **c** Level 2 SOM search (strategy 2) and **d** proposed combined search method for Lena image

method reduces the search complexity to about 11% with PSNR decrease of 0.48 dB for Lena, to about 14% with drop in PSNR of 0.28 dB for Barbara and to 12% with loss of PSNR 0.36 dB for Boat. Thus the comparison of strategy 2 and combined method reveals a significant decrease in search complexity for the combined method with a very little sacrifice in quality.

The compression ratios reported here are the final values with Huffman coding of the transformed index offsets and the difference coded block averages. As can be seen from the results, strategy 1 and combined method produce almost the same compression ratio for all images while the exhaustive search and strategy 2 results show marked similarity in compression ratio. This is the consequence of the restriction imposed on strategy 1 and combined method through the use of the search window. Figure 3 shows the reproduced Lena image quantized using exhaustive search VQ, strategies 1, 2 and combined method in the panels (a) to (d) respectively. Visual inspection reveals almost no difference of quality among them. Figure 4 shows the histograms of offset values (for each offset value x and y frequencies are shown side-by-side) for the methods studied.

The Fig. 4a and 4c show marked similarity while 4b and 4d are almost identical with more concentration of the offset values at and around zero. This explains the higher compression rates achieved by the strategy 1 and combined method due to more efficient Huffman coding of the indexes.

Now we compare the overall performance of the three schemes based on the three factors, namely *reproduction quality*, *search complexity* and *compression ratio*. Strategy 1 has the best reproduction quality (see Fig. 3) as well as its compression ratio is either lowest (for Barbara) or the same as that of the combined method. However, it has a much higher search complexity compared to the strategy 2 and combined method. Strategy 2 reduces the search complexity to a great extent with a slight decrease in reproduction quality, but it has compression ratio higher than strategy 1. The combined method reduces the search complexity further with negligible decrease in reproduction quality, but achieves low compression ratio similar to strategy 1. Thus, considering three factors together the combined method outscores both strategy 1 and strategy 2 when applied separately.

As mentioned earlier, for the design of the encoder, the choice of the *search window size* $s_h \times s_w$ and the *quality threshold* T play important roles. We have conducted an empirical study by designing the VQ (the combined method) with various choices of the search window sizes and quality thresholds, and collected the statistics for quantizing the

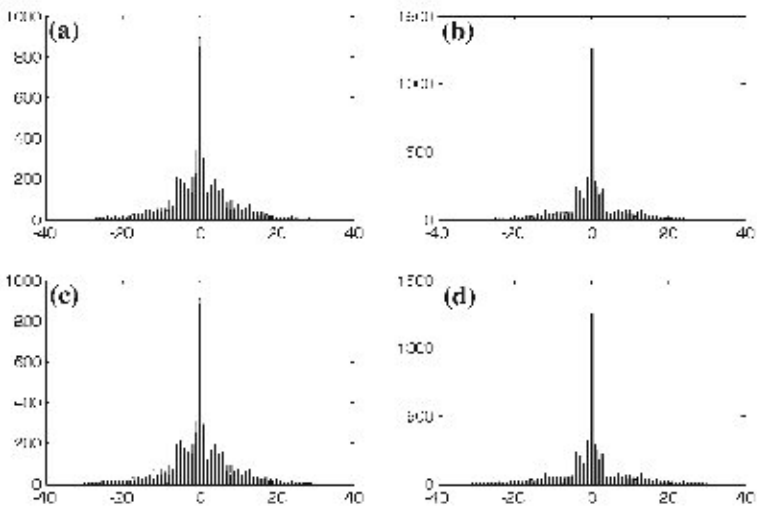

Fig. 4 The histogram of offset values. **a** exhaustive search, **b** SOM window search (strategy 1), **c** Level 2 SOM search (strategy 2) and **d** proposed combined search method for Lena image. x and y values are grouped for each value

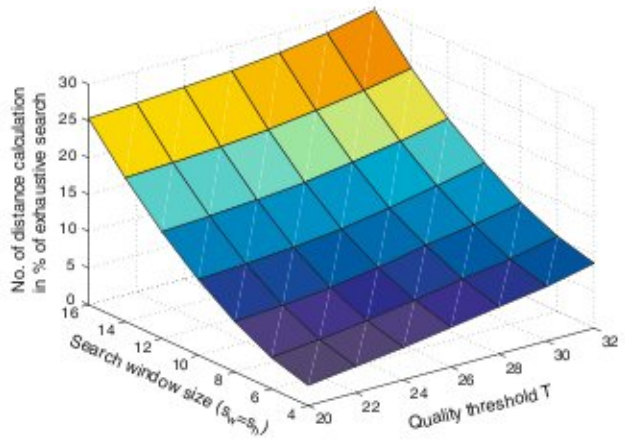

Fig. 5 Variation of number of distance calculation with search window size and quality threshold for quantizing Lena image

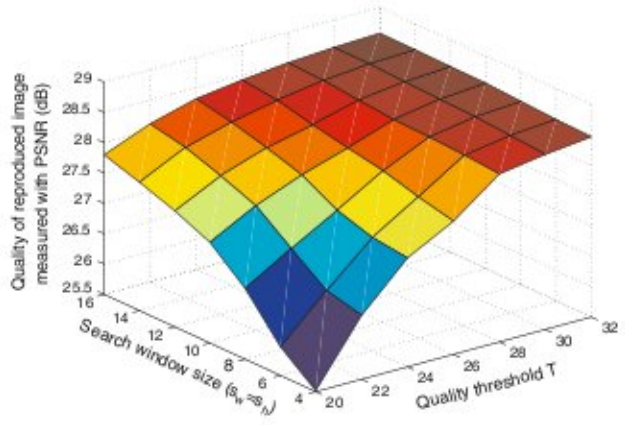

Fig. 6 Variation of reproduction quality (measured in PSNR) with search window size and quality threshold for quantizing Lena image

Lena image. In Fig. 5, the variation in number of distance calculations for different window sizes and quality thresholds is depicted. It can be observed that for both increase of window size as well as threshold value, the distance computation increases. However, the variation is much less with respect to threshold value compared to the variation with window size. This clearly indicates the strong possibility of finding a good match within a small window. In Fig. 6, the variation of the reproduction quality is presented. Here it is evident that the quality threshold has more influence on the reproduction quality than the window size. Figure 7 depicts the frequency of Level 2 SOM searches caused by the derailments. This also indicates a greater influence of the quality threshold than the window size.

5 Conclusion

Kohonen's SOM is used by several researchers for designing the codebook of a vector quantizer. However, these methods

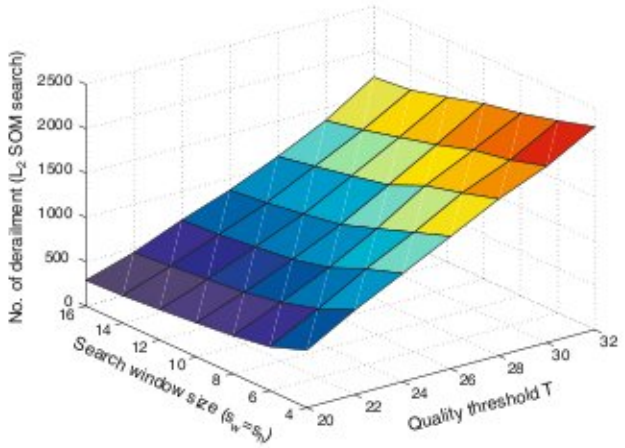

Fig. 7 Variation of number of level 2 SOM searches (i.e, derailments) with search window size and quality threshold for quantizing Lena image

are restricted mainly to the use of SOM as a clustering algorithm. In this work, apart from utilizing the clustering property, we exploited the other notable property of topology preservation to formulate an encoding method with reduced search complexity. First we designed two separate strategies using SOM for fast codebook search. The first strategy used the main SOM generated codebook and exploited the topology preservation property by restricting the codebook search to a small window. This strategy is designed in line with the finite state VQs without explicit calculation of state codebooks. However, exhaustive search of the codebook is performed when a good match is not found within the window. Evidently, each exhaustive search requires $|\mathcal{C}|$ distance calculations. The second strategy uses along with the basic SOM another smaller SOM (L2-SOM) trained with the weight vectors of the basic SOM and produces a partition of the codevectors. This strategy does not exploit the topology preservation property but partitions the codebook into smaller subcodebooks. Thus, for every codebook search on the average $M + \frac{|\mathcal{C}|}{M}$ distance calculations need to be performed. Finally, we proposed a method that combined the best features of both strategies, i.e, the L2-SOM as well as restricted search within a window, to deliver best overall performance. Thus in the combined method, the decrease in the total cost of derailments with respect to the strategy 1 is $\left(|\mathcal{C}| - \left(M + \frac{|\mathcal{C}|}{M}\right)\right) p_d$ distance calculations. The use of SOM and restricted search combined with suitable transformations of index values and block averages enabled us to apply Huffman encoding to enhance the compression ratio without compromising reproduction quality.

The choice of two design parameters, "the search window size" and the "quality threshold" influence the computational load of the encoder significantly. However, our empirical study shows that the rate of increase in computational load

with increase of quality threshold for a fixed window size is much greater than that when the threshold is kept constant and window size is increased. This indicates that if a match satisfying certain threshold is to be found within the search window, more often than not it is found within a small neighborhood of the previous index. Thus this finding also indicate the suitability of SOM-based codebook search methods proposed in this work.

References

1. Abut, H.: IEEE Reprint Collection, chapter Vector Quantization. IEEE Press, Piscataway (1990)
2. Amerijckx, C., Verleysen, M., Thissen, P., Legat, J.: Image compression by self-organized kohonen map. IEEE Trans. Neural Netw. **9**(3), 503–507 (1998)
3. Baker, R.L., Gray, R.M.: Differential vector quantization of achromatic imagery. In: Proc. Int. Picture Coding Symposium pp. 105–106 (1983)
4. Bezdek, J.C.: Pattern Recognition with Fuzzy Objective Function Algorithms. Plenum, New York (1981)
5. Campos, M.M., Carpenter, G.A.: S-tree: self-organizing trees for data clustering and online vector quantization. Neural Netw. **15**, 505–525 (2001)
6. Chang, P.C., Gray, R.M.: Gradient algorithms for designing adaptive vector quantizer. IEEE Trans. ASSP **ASSP-34**, 679–690 (1986)
7. Czihó, A., Solaiman, B., Lováni, I., Cazuguel, G., Roux, C.: An optimization of finite-state vector quantization for image compression.Signal Proc. Image Comm. **15**, 545–558 (2000)
8. Gersho, A., Gray, R.M.: Vector Quantization and Signal Compression. Kluwer, Boston (1992)
9. Gray, D.L., Neuhoff, R.M.: Quantization. IEEE Trans. Inf. Theory **44**(6), 1–63 (1998)
10. Hamzaoui, R., Saupe, D.: Combining fractal image compression and vector quantization. IEEE Trans. Image Proces. **9**(2), 197–208 (2000)
11. Klautau, A.B.R. Jr.: Predictive vector quantization with intrablock predictive support region. IEEE Trans. Image Proces. **8**(2), 293–295 (1999)
12. Karayiannis, N.B.: Fuzzy vector quantization algorithms and their application in image compression. IEEE Trans. Image Proces. **4**(3), 1193–1201 (1995)
13. Kaski, S., Kangas, J., Kohonen, T.: Bibliography of self-organizing map (som) papers: 1981–1997. Neural Computing Surveys (online Journal at http://www.cse.ucsc.edu/NCS/) **1**, 102–350 (1998)
14. Khalil, H., Rose, K.: Predictive vector quantizer design using deterministic annealing. IEEE Trans. Signal Proces. **51**(1), 244–254 (2003)
15. Kohonen, T.: The self-organizing map. Proc. IEEE **78**(9), 1464–1480 (1990)
16. Koikkalainen, P., Oja, E.: Self-organizing hierarchical feature maps. In: Proc. IJCNN-90, International Joint Conference on Neural Networks, Washington, DC, vol. II. pp. 279–285, Piscataway. IEEE Service Center (1990)
17. Kossentini, F., Chung, W., Smith, M.: Conditional entropy constrained residual vq with application to image coding. IEEE Trans. Image Proces. **5**, 311–321 (1996)
18. Laha, A., Chanda, B., Pal, N.R.: Accelerated codebook searching in a som-based vector quantizer. In: Proceedings of World Congress in Computational Intelligence (WCCI 06 - IJCNN) pp. 5945–5950 (2006)
19. Laha, A., Pal, N.R., Chanda, B.: Design of vector quantizer for image compression using self-organizing feature map and surface fitting. IEEE Trans. Image Proces. **13**(10), 1291–1303 (2004)
20. Lai, Y.-C., Liaw, J.Z.C.: Fast-searching algorithm for vector quantization using projection and triangular inequality. IEEE Trans. Image Proces. **13**(12), 1554–1558 (2004)
21. Linde, Y., Buzo, A., Gray, R.M.: An algorithm for vector quantizer design. IEEE Trans. Commun. **COM-28**, 84–95 (1980)
22. Lloyd, S.P.: Least-squares quantization in pcm. IEEE Trans. Inf. Theory **IT-28**, 129–137 (1982)
23. Martinetz, T., Schulten, K.: Topology preserving networks. Neural Netw. **7**(3), 507–522 (1994)
24. Nasrabadi, N.M., Feng, Y.: Vector quantization of images based upon the kohonen self-organization feature maps. Proc. 2nd ICNN Conf **1**, 101–108 (1988)
25. Nasrabadi, N.M., King, R.A.: Image coding using vector quantization: a review. IEEE Trans. Commun. **36**(8), 957–971 (1988)
26. Oja, M., Kaski, S., Kohonen, T.: Bibliography of self-organizing map (som) papers: 1998–2001. Neural Computing Surveys (online Journal at http://www.cse.ucsc.edu/NCS/), **3**, 1–156 (2002)
27. Yair, E., Zager, K., Gersho, A.: Competitive learning and soft competition for vector quantizer design. IEEE Trans. Signal Proces. **40**(2), 394–309 (1992)
28. Zeger, K., Vaisey, J., Gersho, A.: Globally optimal vector quantizer design by stochastic relaxation. IEEE Trans. Signal Proces. **40**(2), 310–322 (1992)