Dictionary Based Compression for Images

Bruno Carpentieri

***Abstract*—**Lempel-Ziv methods were original introduced to compress one-dimensional data (text, object codes, etc.) but recently they have been successfully used in image compression. Constantinescu and Storer in [6] introduced a single-pass vector quantization algorithm that, with no training or previous knowledge of the digital data was able to achieve better compression results with respect to the JPEG standard and had also important computational advantages.

We review some of our recent work on LZ-based, single pass, adaptive algorithms for the compression of digital images, taking into account the theoretical optimality of these approach, and we experimentally analyze the behavior of this algorithm with respect to the local dictionary size and with respect to the compression of bi-level images.

***Keywords*—**Image compression, textual substitution methods. vector quantization.

I. Introduction

In textual substitution compression methods a dictionary D of strings is continuously updated to adaptively compress an input stream by replacing the substrings of the input sequence that have a correspondence in the local dictionary D by the corresponding index into D (these indices are referred as pointers).

The D can be a *static* or *adaptive* dictionary. Static dictionaries can be used when the behavior of the input source is well known in advance, otherwise an constantly changing, adaptive dictionary is used to give a good compromise between compression efficiency and computational complexity.

These algorithms are often called dictionary based methods, or dictionary methods, or Lempel-Ziv methods after the seminal work of Lempel and Ziv.

In practice the textual substitution compression methods are all inspired by one of the two compression approaches presented by Lempel and Ziv. These methods are often called LZ77 and LZ78 or LZ1 and LZ2 respectively in the order in which they have been published. There are many possible variants of LZ1 and LZ2 and they generally differ in the way the pointers in the dictionary are represented and in the limitations on the use of these pointers.

Lempel and Ziv proved that these proposed schemes were practical as well as asymptotically optimal for a general source model.

The LZ2 algorithm (also known as LZ78) is presented in Ziv and Lempel in [1].

By limiting what could enter the dictionary, LZ2 assures that there is at most one instance for each possible pattern in the dictionary.

Initially the dictionary is empty. The coding pass consists of searching the dictionary for the longest entry that is a prefix of a string starting at the current coding position.

The index of the match is transmitted to the decoder using $\lceil \log_2 N \rceil$ bits, where N is the current size of the dictionary. A new pattern is introduced into the dictionary by concatenating the current match with the next character that has to be encoded.

The dictionary of LZ2 continues to grow throughout the coding process.

In practical applications, to limit space complexity, some kind of deletion heuristic must be implemented.

This algorithm is the basis of many widely used compression systems.

Two-dimensional applications of textual substitution methods are described in Lempel and Ziv [2], Sheinwald, Lempel and Ziv [3], and Sheinwald [4].

All these approaches are based on a linearization strategy that is applied to the input data, after which the resulting mono-dimensional stream is encoded by using one-dimensional LZ type methods.

Storer [5] first suggested the possibility of using dynamic dictionary methods in combination with Vector Quantization to compress images.

Constantinescu and Storer in [6] pioneered this approach.

II. The Single-Pass Adaptive Vector Quantization Algorithm

In the Adaptive Vector Quantization image compression algorithm, as in the general one-dimensional lossless adaptive dictionary method, a local dictionary *D* (that in this case shall contain parts of the image) is used to store a constantly changing set of items.

The image is compressed by replacing parts of the input image that also occur in *D* by the corresponding index (we refer to it as *pointer*) into *D*.

This is a generalization to two-dimensional data of the textual substitution methods that Lempel and Ziv introduced for one-dimensional data.

The compression and decompression algorithms work in lockstep to maintain identical copies of *D* (which is constantly changing).

The compressor uses a *match heuristic* to find a match

Bruno Carpentieri is with the Dipartimento di Informatica of the University of Salerno (Italy) (e-mail: bc@dia.unisa.it).

between the input stream and the dictionary (the strategy used to identify the correct matching is generally a greedy strategy that finds the larger match between a dictionary entry and the portion of the image that has to be encoded) then removes this match from the not yet encoded part of the image and transmits the index of the corresponding dictionary entry.

Finally the encoder updates the dictionary via an *update heuristic* that depends on the current contents of the dictionary and on the match that was just found.

If there is not enough room left in the dictionary, a *deletion heuristic* is used to delete an existing entry.

We define as *Growing Border* the set of locations in the input image that have not been coded yet and that are closest to the already–coded locations.

A *Growing Point* is any not yet encoded point where a match may take place (see Fig. 1).

In the rest of this work we assume that growing points come only from the growing border and that they are located in corners, as shown in Figure 1.

The data structure that stores the growing points is called *Growing Points Pool*.

A *Growing Heuristic* is a set or rules used to identify the next growing point to be processed.

There are many different ways to implement the growing heuristic.

The growing heuristics (GH) are used to select a growing point in the growing points pool for the next matching.

This can be done by considering the order in which the growing points are added to the growing points pool or by limiting somehow the possible x and/or y coordinates of the next growing point.

As example of possible growing heuristics we can cite the *GH Wave*, that selects the GP *s* with coordinates x_s and y_s for which $x_s + y_s <= x + y$, for any other growing point with coordinates x and y in the growing point pool.

In this case the growing point pool will be initialized with the upper left corner pixel of the image and the covering of the image will result in a wave that is perpendicular to the image's diagonal (see Figure 1).

Another example of growing heuristic is the GH circular that selects from the growing points pool the growing point that has minimum distance from a specific image point (generally the center point of the image), as shown in Figure 2.

At each step the Adaptive Vector Quantization algorithm (*AVQ* from now on) selects a growing point *gp* of the input image.

The encoder uses a match heuristic to decide which block *b* of a local dictionary *D* is the best match for the sub-block anchored in *gp* of the same shape as *b*.

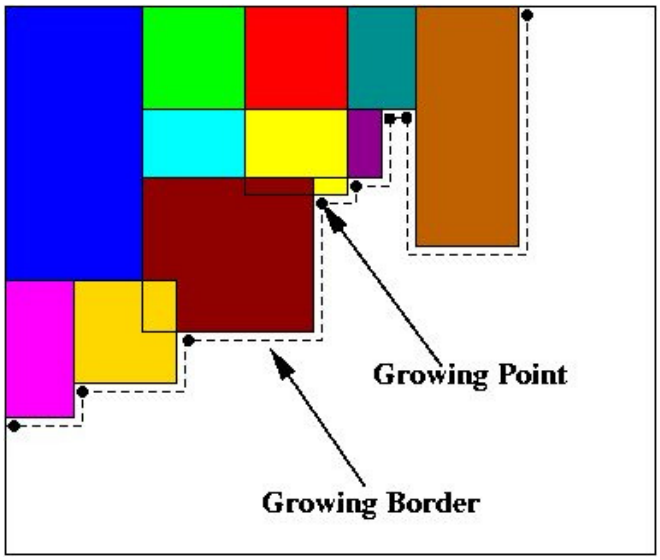

Fig. 1: Wave Growing Heuristic

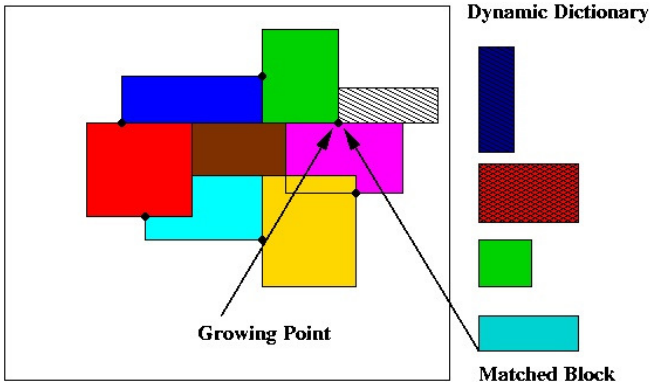

Fig 2: Circular Growing Heuristic

The match heuristic chooses the largest block for which the distortion from the original block (namely the mean squared error) is less or equal to a threshold *T*.

The threshold *T* could be fixed for the whole image, or dynamically adjusted to the image content: higher for smooth areas, lower for zones with large variance (as in the presence of sharp edges).

The Lossy Generic Encoding and Decoding Algorithms for on-line adaptive vector quantization can be summarized as follows.

Lossy Generic Encoding Algorithm:

*1. **Initialize** the local dictionary D to have one entry for each possible pixel value (e.g., each possible byte value in an 8-bit per pixel gray scale image) and the growing points pool with the initial set of growing points (e.g., a single point in the upper left corner for the wave method).*

*2. **Repeat***

A. Use a Growing Method to choose the next growing point gp.

B. Use a Match Method to find a block b in D that matches with acceptable fidelity the sub-block anchored in the growing point gp, and transmit the index of b.

C. Use an Update Method to add one or more new entries to D; if D is full, first use a Deletion Method to make space.

D. Update the growing points pool according to a Growing Points Update Method.

until the growing points pool is empty

Lossy Generic Decoding Algorithm:

*1. **Initialize** D by performing Step 1 of the Encoder's Algorithm.*

*2. **Repeat***

A. Perform Step 2A of the Encoder's algorithm to determine a growing point gp.

B. Receive the index of the block b to be placed at gp and output b.

C. Perform Step 2C of the Encoder's Algorithm modify D.

D. Perform Step 2D of the Encoder's Algorithm to modify the growing points pool.

until the growing points pool is empty

The operation of the generic algorithms is guided by the following heuristics:

The *growing heuristic* that selects one growing point from the available Growing Points Pool.

The *match heuristic* that decides what block b from the dictionary D best matches the portion of the image of the same shape as b defined by the currently selected growing point.

The *growing points update heuristic* that is responsible for generating new growing points after each match is made.

The *dictionary update heuristic* that adapts the contents of the dictionary D to the part of the image that is currently compressed/decompressed.

The *deletion heuristic* that maintains the dictionary D so it can have a predefined constant size.

Figure 3 illustrates the concepts of growing point, dynamic dictionary and current match.

The compression performance of AVQ typically equals or exceeds the compression obtained by the JPEG standard on different classes of images and often out-performs traditional trained–vector quantizers.

The class of images on which JPEG still prevails is the one on which it was tuned ("magazine photographs").

III. The Theoretical Performance of Lossless LZ-based Image Compression

The complexity and asymptotic optimality of the one dimensional Lempel-Ziv methods have been widely studied in the literature, and this theoretical analysis are probably one of the reasons of the huge popularity of these methods.

It is possible to prove with similar arguments also the asymptotic optimality of the AVQ algorithm.

As shown in [7], the two-dimensional image compression problem could be reduced to a lossless compression in one dimension problem if we consider the the *2-dimensional UC update method* (2D-UC), i.e. a dictionary update heuristic that adds to the dictionary the current match augmented by one pixel chosen among the immediate neighbors on its right or lower side.

This pixel is sent uncompressed to the decompressor together with the pointer.

Pixels are added to the right side first, from top to bottom, and to the bottom next, from left to right.

In this way the algorithm will tend to grow (almost) square matches.

If we use AVQ with the 2D-UC method to lossless encode an image I_{NxM} , where N and M are the width and height of the image, and there is no overlap between matches and if we take the sequence of matches and traverse it in a zig-zag order the result is a distinct parsing of a sequence that is a one-dimensional scan of the content of I_{NxM} .

If this sequence is drawn from an ergodic source, the optimality of the compression achieved on this string follows from :

Theorem (12.10.1 from Cover and Thomas [8]) :

Let $\{X_i\}_{-\infty}^{\infty}$ be a stationary ergodic process with entropy rate $\mathrm{H}(X)$, and let $c(n)$ be the number of phrases in a distinct parsing of a sample of length n . Then

$$\limsup_{n\to\infty} \frac{c(n)\lg c(n)}{n} \le \mathrm{H}(X)$$

with probability 1.

In other words if a string is parsed into distinct phrases (by any method) and the i^{th} phrase is charged a cost of $\lceil \log_2 i \rceil$, then the total cost is optimal in the information theoretic sense.

In [7] it is also shown how this complexity results also hold when we consider the possibility of a parsing that has a bounded overlap degree.

IV. Dictionary Size and Compression Performances

We have tested the lossy compression performance of our

AVQ implementation with respect to the local dictionary dimensions.

The compression/decompression process is depicted in Figure 4, which shows how the process works on the standard Lena image.

Figure 4-a is a partially decompressed version of Lena (the white part has not been compressed/decompressed yet) where each rectangle has been colored with a random solid color to illustrate the covering pattern.

Figure 4-b is the actual partially decompressed Lena image. It can be seen that larger rectangles are "grown" from smaller ones as the image is compressed and that to compress "background" of the image large matches are used whereas smaller matches of varying size are used to reproduce the details of Lena's face.

Of course the size of the local dictionary has to be finite otherwise we could not code the pointers effectively.

If the dictionary is too small then the algorithms looses parts of its adaptive behavior and this impacts the compression performance.

On the other hand if the dictionary is too large, the cost of encoding each pointer grows too much and again this can affect the compression performances.

We have tested our implementation with different dictionary sizes on the standard test data set including the classical Lena, Zelda, Balloon, Peppers, Gold, Barb, Barb2, Girl, Boats and Hotel images.

We have experimented with different dictionary sizes, ranging from 2^{11} to 2^{14} elements (and more).

The first experimental result is that for our data set a dictionary with more than 2^{14} elements does not work well: the cost of encoding the pointers becomes too large.

The best performance is obtained with a dictionary of 2^{14}, i.e. 16384, elements.

Figure 5 shows our test data set.

The ten images, from left to right, top to bottom, are respectively: Lena, Zelda, Balloon, Peppers, Gold, Barb, Barb2, Girl, Boats, Hotel .

Table 1 shows our experimental results.

In the table we have as columns the Compression Ratio, the values of the Signal to Noise Ratio (SNR) and Mean Squared Error (MSE) as quality measures, and the original and compressed sizes, both in bytes and KBs.

It is possible to have an average 4% improvement with respect to the above results if we carefully code the pointers while the dictionary fills up (i.e. the cost of coding a pointer while the dictionary fills up depends on the real actual number of entries that are currently in the dictionary).

Image	C.R.	SNR	MSE	Original size	Comp. size
Lena	6,614	23,42	16	263.224 (257 KB)	39.797 (38,8 KB)
Zelda	10,575	19,45	15	415.800 (406 KB)	39.321 (38,3 KB)
Balloon	8,651	23,91	4,3	415.800 (406 KB)	48.064 (46,9 KB)
Peppers	5,160	21,82	26	263.224 (257 KB)	51.012 (49,8 KB)
Gold	4,311	23,24	13,5	415.800 (406 KB)	96.458 (94,1 KB)
Barb	4,895	22,60	17,5	415.800 (406 KB)	84.945 (82,9 KB)
Barb2	4,410	21,39	17,5	415.800 (406 KB)	94.285 (92,0 KB)
Girl	8,573	23,30	16,5	415.800 (406 KB)	48.498 (47,3 KB)
Boats	4,906	24,71	8	415.800 (406 KB)	84.744 (82,7 KB)
Hotel	4,679	23,07	12	415.800 (406 KB)	88.858 (86,7 KB)

Table 1: experimental results with dictionary size 2^{14}

V. AVQ AND BILEVEL IMAGES

We have so far considered the application of the AVQ algorithm on images that have at least 8 bits of gray per pixel. Of course the approach we have presented can be successfully used also for color images.

The AVQ algorithm can also be efficiently used in the lossless and near-lossless compression of bi-level images (i.e. black and with images, as for instance the results of a normal fax transmission).

We just need to use the lossless version of the algorithm (i.e. the matching heuristic now will look only for exact matches, without admitting errors) and by initializing the dictionary with only to elements: the forst for the "black" pixel, and the second for the "white" pixel.

The results obtained are interesting, our algorithm outperformaces the standard *gzip* and *compress* lossless compressor.

Even if straight AVQ does not equal the performaces of the JPBIG standard, our first experiments seem to indicate that we opportune tuning the AVQ algorithm can be probably as fast and as efficient as JBIG.

In Table 2 we report our preliminary experiments on the standard CCITT test data set. The results are given in terms of compression ratio.

In the table there is also a column entry also for a slightly modified version of AVQ that we called AVQ-INIT.

The improvement consists in the inizialization of the dictionary also with the black and with patterns shown in Figure 6 and with a set of square bloks, all black and all white, of different sizes (6, 7, 8, …, 16, 32 , 64,…, 512).

VI. Conclusions and Future Research Directions

The AVQ algorithm introduced by Constantinescu and Storer is an adaptive vector quantization algorithm that applies to the digital image data the LZ2 textual substitution methods. This method maintains a constantly changing dictionary of variable sized rectangles by "learning" larger rectangles from smaller ones as the target image is compressed.

We have reviewed this LZ-based, single pass, adaptive algorithms for the compression of digital images and the asymptotic optimality of the AVQ algorithm.

We have experimentally analyzed its behavior with respect to the size of the local Dictionary.

We have also experimentally analyzed the AVQ performances on bi-level images.

We are testing different encoding strategies for dictionary information. Moreover the impact of overlapping during the encoding process has to be carefully studied.

References

[1] J. Ziv, A. Lempel, A Universal Algorithm for Data Compression, IEEE Transaction on Information Theory, Vol. IT-23, n. 3, May 1977.

[2] A. Lempel, J. Ziv, Compression of Two-Dimensional Images, in: A.Apostolico and Z.Galil (Ed.), Combinatorial Algorithms on Word, NATO ASI Series, Vol. F12, Springer-Verlag Berlin, Heidelberg 1985.

[3] D. Sheinwald, A. Lempel, J. Ziv, Two-dimensional Encoding by Finite State Encoders, IEEE Transaction on Communications, Vol. 38, pp. 341-347, 1990.

[4] D. Sheinwald, Finite State Two-Dimensional Compressibility, in: J. A. Storer (Ed.), Image and Text Compression, Kluwer Academic Press, Norwell, MA, 1992, pp.253-275.

[5] J. A. Storer, Data Compression: Methods and Theory, Computer Science Press, 1988.

[6] C. Constantinescu, J. A. Storer, On-Line Adaptive Vector Quantization with Variable Size Codebook Entries, Journal of Information Processing Management, 1994.

[7] F. Rizzo, J. A. Storer, B. Carpentieri, Overlap and channel errors in Adaptive Vector Quantization for Image Coding, Information Sciences, Vol. 171, pp. 125-143, 2005.

[8] T. M. Cover and J. A. Thomas [1991]. Elements of Information Theory, Wiley.M. Young, *The Techincal Writers Handbook.* Mill Valley, CA: University Science, 1989.

IMAGE	COMPRESS	GZIP	AVQ	AVQ-INT	JBIG
CCITT1	15.8	16.5	20.0	20.2	28.9
CCITT2	18.6	18.6	30.1	31.0	43.8
CCITT3	8.9	8.9	13.1	13.4	19.6
CCITT4	5.0	5.0	5.4	5.5	7.9
CCITT5	8.2	8.2	11.1	11.3	16.7
CCITT6	12.2	12.2	22.1	21.9	33.8
CCITT7	4.6	4.6	5.5	5.6	7.4
CCITT8	10.0	10.0	17.5	18.6	25.9

Table 2: AVQ compression ratio on bi-level images compared with other standard algorithms.

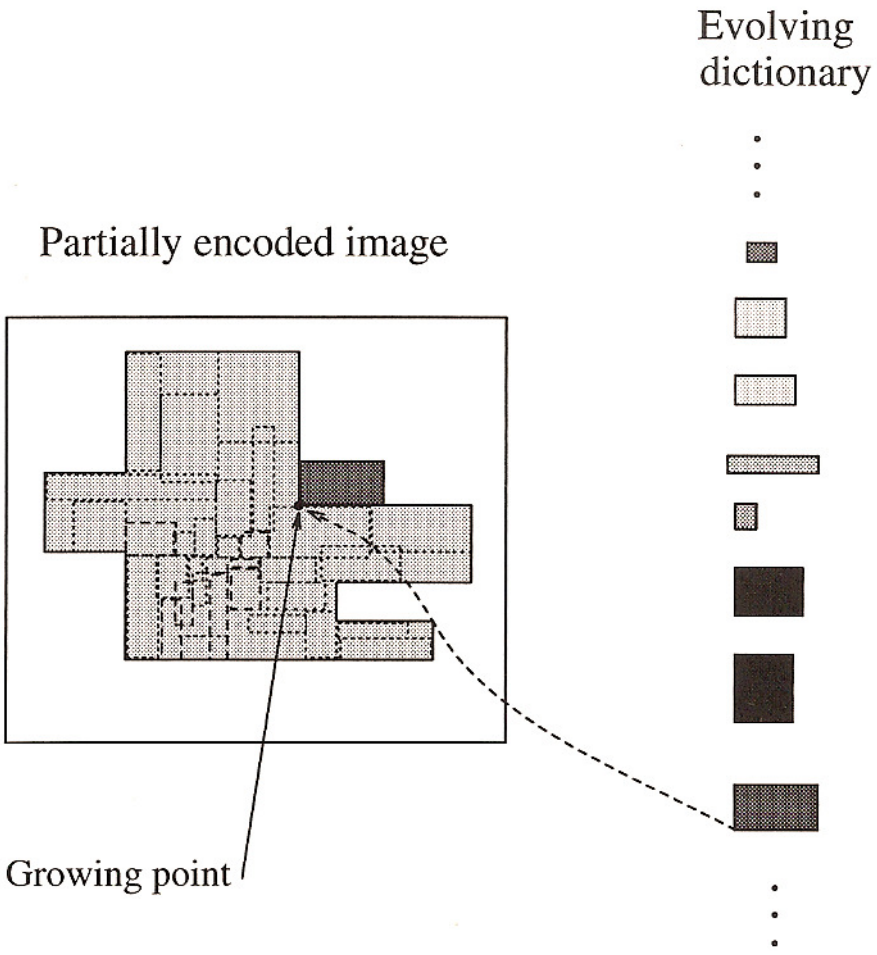

Figure 3: Adaptive Vector Quantization

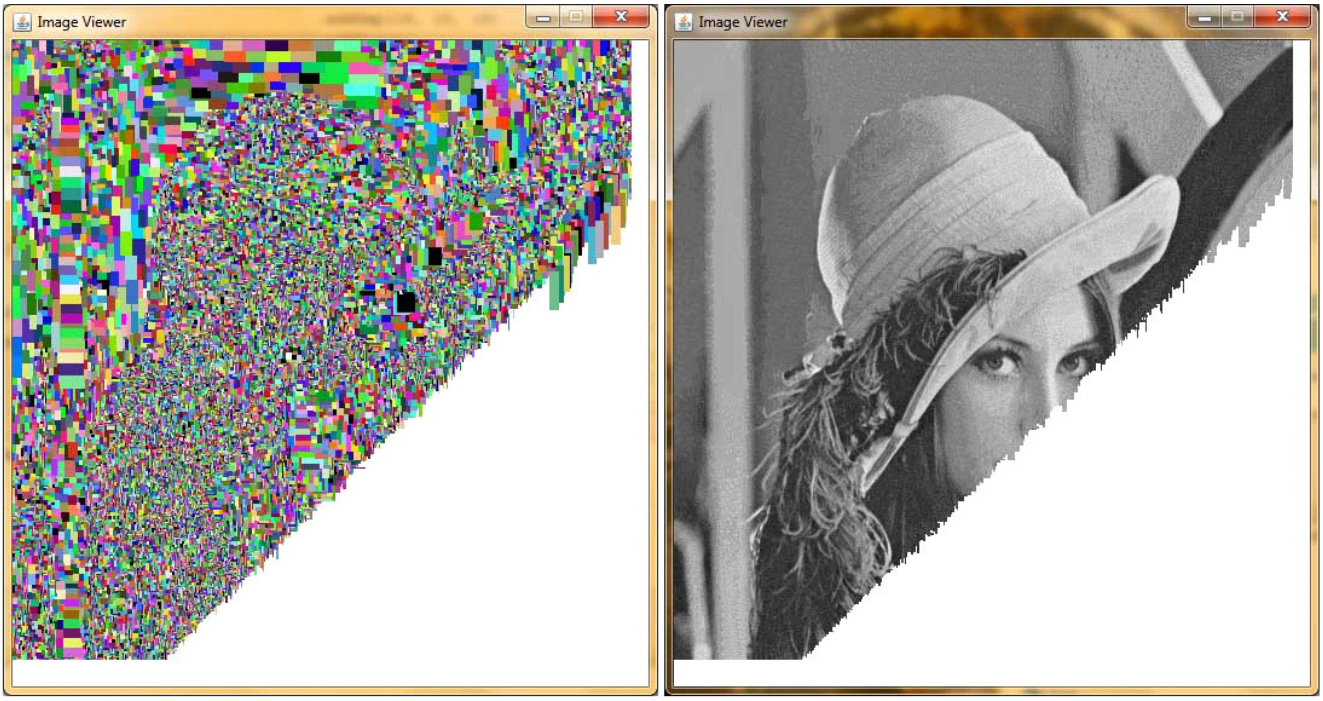

Figure 4: AVQ compression/decompression of "Lena"

Figure 5: Grayscale Test Data Set

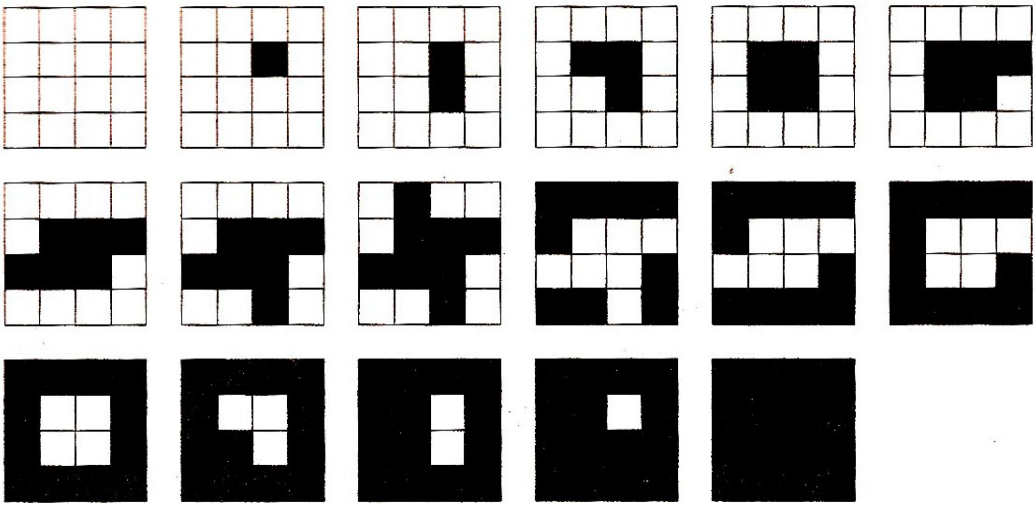

Figure 6: Dictionary inizialization patterns for bilevel images