Image Compression via Textual Substitution

BRUNO CARPENTIERI
Dipartimento di Informatica ed Applicazioni "R. M. Capocelli"
Università di Salerno
Via S. Allende – 84081 Fisciano (SA)
ITALY
bc@dia.unisa.it http://www.dia.unisa.it/professori/bc

Abstract: - Textual substitution methods, often called dictionary methods or Lempel-Ziv methods, after the important work of Lempel and Ziv, are one-dimensional compression methods that maintain a constantly changing dictionary of strings to adaptively compress a stream of characters by replacing common substrings with indices (pointers) into a dictionary. Lempel and Ziv proved that the proposed schemes were practical as well as asymptotically optimal for a general source model. Two-dimensional (i.e. images) applications of textual substitution methods have been widely studied in the past. Those applications involve first the application of a linearization strategy to the input data, and then the encoding of the resulting mono-dimensional vector using LZ type one-dimensional methods. More recent strategies blend textual substitution methods with Vector Quantization.
In this paper we discuss the textual substitution methods for image compression, with particular attention to the AVQ class of algorithms, and review recent advances in the field.

Key-Words: - Data Compression, Dictionary Compression, Image Compression, Vector Quantization.

1 Textual Substitution Methods

An interesting approach to compressing a string of characters is textual substitution.

A textual substitution method is any compression method that compresses input by identifying repeated substrings and replacing some substrings by references to other copies. Generally, textual substitution methods are implemented as on line, dynamic, lossless, one-dimensional, compression algorithms.

Lempel and Ziv [1] proved that the proposed schemes were asymptotically optimal for a general source model. The compression algorithms that apply these schemes are generally known as Lempel Ziv (LZ) coders and are divided in two different classes: LZ1 and LZ2 type methods. These two classes differ in the way the pointers are represented and in the type of objects to which the pointers point.

LZ1 methods (see [1]) represent pointers by triples <d,l,c> (d is the displacement of the matching substring relative to the current position, l is the length of the match, and c is the character that follows the newly encoded string). The "dictionary" consists of all the substrings that occur previously in the input within a finite distance from the current position. This kind of dictionary is therefore a first–in–first–out (FIFO) queue of the immediate past, and it is generally referred to as "sliding window".

The LZ2 algorithm is presented in Ziv and Lempel [2]. LZ2-based schemes, also called LZ78 or dictionary methods, attempt to code the repeated occurrence of a substring with an index in a specific dictionary. 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 $\log_2(N)$ bits, where N is the dictionary size. This method updates the dictionary, that is generally initialized with the alphabet symbols, with new strings that can be, for example, a concatenation of the previous match with some new set of strings based on the current match. A trie data structure is often used to represent the dictionary; when the memory is full several strategies can be used to gain new space: the trie could be removed and reinitialized, it could be "frozen", or some substrings could be deleted to leave space to new data.

Although asymptotically identical, in practice LZ2 encoding is faster than LZ1, while the converse is true for decoding. Nevertheless, the general method is attractive, and it is the basis of many widely used compression systems.

Two-dimensional (i.e. images) applications of textual substitution methods have been widely studied in the past, for instance by Lempel and Ziv [3] themselves and by Sheinwald, Lempel and Ziv [4]. Those researches involve first the application of a linearization strategy to the input data, and then the encoding of the resulting mono-dimensional vector using textual substitution, one dimensional, methods.

Storer, in [5], first suggested the possibility of using dynamic dictionary methods in combination with Vector Quantization for lossless and lossy compression of bi-dimensional data.

The AVQ algorithm of Constantinescu and Storer [6] and the LZ1 based algorithm of Storer and Helfgott [7] pioneered this approach by showing that it has a number of advantages with respect to current state-of-the art algorithms such as JPEG.

In this paper we discuss the textual substitution methods for image compression, with particular attention to the AVQ class of algorithms, and review recent advances in the field.

2 Sliding Window Methods for Image Compression

Storer and Helfgott, in [7], present a generalization of the LZ1 method to lossless compression of bi-level images. They use a wave heuristic to scan the image and a multi–shape, two–dimensional, suffix trie to represent the dictionary (i.e. the window of the already coded pixels).

For each growing point, the largest square match of size k, for a given k, is computed and then the window is searched for the entry of size k*u or v*k that will code the largest number of unmarked pixels. If such match exists, then the quadruple <x,y,w,h> is sent to the decoder, where x and y are the location of the matching pattern and w and h are its dimensions. A raw block is sent otherwise.

The implementation proposed in [7] is very fast in practice (especially for decoding), and it is not too far to the current state of the art for lossless compression of bi-level images (i.e. JBIG).

Rizzo, Storer and Carpentieri in [8] address the same problem and show how it is possible to improve upon the basic algorithm by using hashing methods and by coding pointers carefully.

In order to reduce computation time they bound the window height. As shown in Figure 1 (from [8]) the window will contain only entries between the Window BackFront and Window ForeFront.

If the window size exceeds some predefined maximum size, the back front is advanced.

CCITT	gzip	SH	SWBM	AVQ	JBIG	JBIG2
1	17.22	17.40	19.14	20.00	39.69	78.66
2	20.46	28.00	28.12	31.00	62.07	65.15
3	11.61	12.00	12.67	13.00	25.29	43.07
4	5.10	5.50	5.45	5.60	10.49	32.54
5	9.78	10.50	11.20	11.60	21.55	47.79
6	17.76	19.30	24.68	22.50	42.93	44.89
7	4.68	5.30	5.36	5.70	9.70	14.34
8	12.87	13.50	17.76	18.40	36.11	39.71

Table 1: A comparison of Storer-Helfgott (SH), SWBM and other compression algorithms on the CCITT test set.

This strategy impacts compression time significantly because only locations within the boundaries are involved in the match search. The width of the window is dynamically bounded too. Bounding the search window is also a way to significantly reduce the entropy of the displacement pointers. In [7] and [8], the proposed two-dimensional, sliding window, block matching algorithms are tested on the CCITT image test set. The compression ratio obtained for lossless compression are shown in Table 1 (from [8]) where [7] is SH and [8] is SWBM.

3 LZ2 Methods for Image Compression

The adaptive vector quantization algorithm (AVQ from now on) presented by Constantinescu and Storer in [6] has been the first effective application of the LZ2 strategy to images.

This approach does not reduce the two-dimensional data to one dimension (as in [4]), but blends the textual substitution strategy with the Vector Quantization compression approach to images.

In [6] it is defined as *Growing Border* the set of locations in the input image that have not yet been encoded and that are closest to the already–coded locations (see Figure 2). A *Growing Point* is any coded point not yet encoded where a match may take place. 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.

There are many different ways to implement the growing heuristic. The one depicted in Figure 3 is usually addressed as *Circular Heuristic*: coding starts from a given location of the input image and the growing heuristic will always pick the closest growing point to the starting one. When the initial growing point is the upper left corner of the input image, the resulting heuristic is addressed as the *Wave Heuristic*, since the image will be coded in a path parallel to the secondary diagonal (i.e. like a "wave", see Figure 2).

At each step the AVQ algorithm selects a growing point of the input image. The encoder uses a match heuristic to decide which block of a local dictionary is the best match for the sub-block anchored on that growing point. The match heuristic chooses the largest block for which the distortion from the original block 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.

It will be higher for smooth areas and lower for zones with large variance (as, for example, sharp edges).

At the starting of the encoding process, the local dictionary contains one entry for each possible pixel value; during the encoding process the dictionary is updated by using a *Dictionary Update Heuristic*, possibly preceded by a *Deletion Heuristic*. The basic structure of the encoding algorithm is described in Algorithm 1:

1. Initialize the local dictionary *D* to have one entry for each pixel of the input alphabet and the growing points pool with the initial set of growing points.
2. Repeat until the *growing points pool* is empty:
   a) Use a *Growing Heuristic* to choose the next growing point *gp* from the growing points pool.
   b) Use a *Match Heuristic* to find a block *b* in *D* that matches with acceptable fidelity the sub-block anchored in the growing point *gp*.
   c) Transmit logD bits for the index of *b*.
   d) Use a *Dictionary Update Heuristic* to update the dictionary *D* (if *D* is full, first use a *Deletion Heuristic* to make space).
   e) Update the growing points pool according to a *Growing Points Update Heuristic*

Algorithm 1: AVQ Coding Algorithm

As reported in [6] and [8], for a given overall fidelity, the compression achieved by the AVQ approach often equals or exceeds the traditionally trained VQ approach and the JPEG algorithm.

As reported in [6] the compression performance of AVQ is weakest on standard magazine photographs, on which it generally equals JPEG or achieves slightly less compression for the same quality, and strongest on technical images as scientific images, fingerprints, medical images, handwriting, graphs, etc, where the adaptation abilities of AVQ yield significant improvements over JPEG and other similar methods. Besides good compression performance the AVQ algorithm has other advantages over traditional image compression techniques. It is a single-pass adaptive algorithm that requires no dictionary or codebook to be provided in advance. Moreover one can provide in advance a precise estimate on the distortion of any sub-block of the image and, with a fixed codebook size, it is possible to vary continuously the fidelity-compression tradeoff. The algorithm presents fast (table-lookup) decoding and practical real-time encoding time.

Image	SNR	JPEG	AVQ	MGSAVQ
Baloon	24.2	27.03	11.85	20.55
	15.3	55.64	39.33	70.47
Barb	22.3	6.10	4.86	5.31
	13.5	19.43	14.16	14.67
Barb2	21.7	5.85	5.13	5.18
	13.6	21.64	14.81	13.99
Board	27.1	13.15	9.78	11.85
	18.9	39.60	28.23	39.12
Boats	24.2	8.19	5.97	6.89
	14.3	22.66	24.62	26.25
Girl	23.0	13.21	8.53	9.19
	14.9	43.90	28.06	29.47
Gold	22.8	7.24	5.96	4.86
	14.4	41.33	23.10	25.43
Hotel	23.5	6.93	5.81	6.21
	15.5	25.98	15.68	18.33
Lena	23.2	8.09	5.34	6.35
	14.1	39.01	20.98	23.95
Peppers	22.3	9.04	6.24	6.75
	14.9	38.68	21.00	25.29
Zelda	20.0	27.69	12.94	19.37
	13.6	51.46	36.92	48.71

Table 2: AVQ vs MSGAVQ

4 More on the AVQ Algorithm

The basic AVQ algorithm is robust against minor changes, such as the choice of growing method.

From the theoretical point of view, it is not easy to analyze its performance.

Alzima, Spankowski, and Grama, in [9] and [10], proposed a formal analysis of the algorithm; although they do not fully resolve the question of whether the AVQ algorithm is an (asymptotically) optimal compression algorithm, they do show experiments that confirm its competitive performance.

Rizzo, Storer and Carpentieri in [11] perform a theoretical analysis of the algorithm, by taking into account one of the key AVQ aspects: matches are allowed to overlap, therefore it is not necessary to perform bin packing in order to cover the image with variable size and shape matches. [11] proves that AVQ is asymptotically optimal (under certain conditions), depending on the overlapping factor

defined as the average number of times that a pixel is covered by a match.

Overlapping is an aspect of AVQ that is really new: this mechanism was not present (or needed) in the one-dimensional lossless textual substitution algorithms or in the lossy two-dimensional VQ, from which AVQ was derived.

Rizzo, Storer and Carpentieri in [8] improve the AVQ performance by adding a mean gain shape transformation to the match heuristic (they call this new algorithm MGSAVQ). Namely, let m_b, g_b and $\vec{s}_b$ be respectively the mean, the gain and the shape of the block b in the dictionary and m_{gp}, g_{gp} and $\vec{s}_{gp}$ be respectively the mean, the gain and the shape of the block x anchored at the growing point gp of the same size and shape of b. They change the match heuristic in Step 2.b as follows:

2.b Compute the vector $\vec{y}_b = g_{gp} \cdot \vec{s}_b + m_{gp} \cdot \vec{1}$ and choose b as the current match if its size is maximal and the distance $d(\vec{x}, \vec{y}_b)$ is less or equal to the fixed threshold T.

Once the best match is found, it is necessary to transmit m_{gp} and g_{gp} to the decoder along with the index of b in the dictionary. In order to minimize the entropy of the output stream, the mean is differentially encoded and it is fed to an arithmetic encoder.

Table 2 (from [8]) shows the experimental results obtained on the typical AVQ image test set at two different levels of quality. The first column indicates the target signal to noise ratio, while the second, third and fourth columns, contain the relative compression ratios of JPEG, AVQ and MGSAVQ respectively.

As in the one-dimensional case, the two-dimensional dynamic dictionary methods, and specifically the AVQ algorithm, turn out to be sensitive to channel errors.

In fact even the flipping of a single bit in the bit-stream is likely to cause catastrophic errors at the decoder side.

Figure 4, from [11], shows what happens when a random byte is altered in the coded stream of the Lena image: the decoded image is almost fully corrupted.

In the one-dimensional case with the presence of channel errors, because of the sequential nature of the input, we can expect a total loss of information.

In the two-dimensional case instead we might expect a different behavior: for example we might expect that a single error has only local effects or, in the worst case, that it spoils only a wedge-shaped area starting from the growing point at which the error occurs.

Instead, as we can see in Figure 4, after the error occurs, the decoded image starts to degrade heavily and no information could be extracted from the corrupted areas. Moreover, the black region in the middle of the decoded image indicates that part of the original image has not been decoded at all.

In fact, as in the one-dimensional case, the error propagates and scrambles the content of the dictionary. Moreover the error causes new (wrong) growing points to be added to the growing point pool, and this will disrupt the entire process.

Rizzo, Storer and Carpentieri, in [11], modify AVQ so that a channel error does not propagate, or at least there is a graceful degradation in the decoded output quality.

A simple approach to solve this problem is to divide the image into a fixed number of macro-areas of a given size and to compress each area separately (i.e., starting with a new, empty dictionary for each macro-area). This way, if the channel is such that the errors occur only in isolated bursts, in general the errors may effect only a few macro-areas, depending on the size of the macro-areas themselves, the duration of the burst and the time at which the error occurs.

This strategy may be profitable when the size of the input is large enough, even though in the worst-case scenario there could be at least one burst for each macro-area.

Another possible strategy to tackle the channel errors is to impose a more rigid structure on the dictionary and to change the amount and type of information that is transmitted through the channel.

The dictionary is then divided into as many sub-dictionaries as the different number of legal block sizes. Each sub-dictionary has the same size and the encoding algorithm is basically the same as the baseline AVQ.

This approach, in [11], is called AVQ-MD. The reorganization of the dictionary allows a reduction of the risk of error propagation in the presence of a noisy channel. In fact, if we assume that the error affects only the information on the match index, the degradation in the decoded image could still be severe but most of the information would be preserved.

In Figure 5, from [11], it is shown the experimental result of decoding the test image Lena when different bit error rates (BER) occur.

5 Conclusion

In this paper we have reviewed the textual substitution methods for image compression.

The LZ1 class of algorithms has been extended to the two-dimensional case by Storer and Helfgott, but

the overall performance of the two-dimensional algorithms is not, at the moment, competitive with the state of the art algorithms for image compression.

The LZ2 class of algorithms, instead, is more promising: 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.

Its compression performance is weakest on standard magazine photographs, on which it generally equals JPEG or achieves slightly less compression than JPEG for the same quality, and it is stronger on technical images as scientific images, fingerprints, medical images, handwriting, graphs, etc, where the adaptation abilities of AVQ yield significant improvements over JPEG and also over other traditional compression methods.

This algorithm has been successively studied also from a theoretical point of view. Its asymptotical optimality has been proven, depending on the overlapping factor that is defined as the average number of times a pixel is covered by a match.

The impact of channel transmission error on the decoding process has been studied too and solutions have been proposed by modifying AVQ so that a channel error does not propagate, or at least there is a graceful degradation in the decoded output quality.

The original AVQ algorithm has been further improved on the class on images on which JPEG does best (magazine photographs) by transforming the input vector in a way similar to the one used in mean-shape-gain vector quantization.

This improved MGSAVQ algorithm, along with better compression results with respect to AVQ, brings also an improvement in the overall visual quality of the decoded image, especially at high compression rate.

The full potential of the presented algorithms has not been totally explored yet and it should be possible to fill the gap in terms of compression ratio with the current standards. We are examining different organizations and encoding strategies of dictionary information.

Future researches, following the footsteps in [12], [13], [14], involve a more careful data representation and an accurate application of the coding strategies to the output of the textual substitution methods, as for example by using arithmetic coding, so to improve the overall compression performance.

References:

[1] A. Lempel and J. Ziv, "On the Complexity of Finite Sequences", *IEEE Transactions on Information Theory*, Vol. 22, No. 1, 1976, pp. 75-81.

[2] A. Lempel and J. Ziv, "Compression of Two-Dimensional Images", In A. Apostolico and Z. Galil editors. *Combinatorial Algorithms on Words*, Springer-Verlag, 1985, pp. 141-154.

[3] J. Ziv, A. Lempel, "Compression of Individual Sequences via Variable Rate Coding", *IEEE Transaction on Information Theory*, Vol. IT-24, No. 5, 1978, pp. 530-536.

[4] D. Sheinwald, A. Lempel and J. Ziv, "Two-dimensional encoding by finite state encoders", *IEEE-COM 38*, 1990, pp. 341-347.

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

[6] C. Constantinescu and J.A. Storer, "Improved Techniques for Single-Pass Adaptive Vector Quantization", *Proceedings of the IEEE*, Vol. 82, No. 6, 1994, pp. 933-939.

[7] J. A. Storer and H. Helfgott, "Lossless Image Compression by Block Matching", *The Computer Journal*, Vol. 40, No. 1, 1997, pp. 137-145.

[8] F. Rizzo, J. A. Storer and B. Carpentieri, "LZ-based Image Compression", *Information Sciences*, Vol. 135, No. 1-2, 2001, pp. 107-121.

[9] M. Alzima, W. Szpankowski, and A. Grama, "2D-pattern Matching Image and Video Compression", Proceedings of the Data Compression Conference, 1999, IEEE Comp. Society Press, pp. 424-433.

[10] M. Alzima, W. Szpankowski, and A. Grama, "2D-pattern Matching Image and Video Compression: Theory, Algorithms and Experiments", *IEEE Transactions on Image Processing*, Vol. 11, No. 3, 2002, pp. 318-331.

[11] F. Rizzo, J. A. Storer, B. Carpentieri, "Overlap and channel errors in Adaptive Vector Quantization for image coding", *Information Sciences*, Vol. 171, No. 1-3, 2005, pp. 125-143.

[12] Luca Matola , Bruno Carpentieri, "Color Re-indexing of Palette-Based images", WSEAS Trans. on Information Sciences and Applications, Vol. 3, n. 2, pp. 455-461, Feb. 2006, WSEAS Press.

[13] Anna Ansalone, Bruno Carpentieri, "How to Set "Don't care" Pixels when Lossless Compressing Layered Documents", WSEAS Trans. on Information Sciences and Applications, Vol. 4, n. 1, pp. 220-225, Jan. 2007, WSEAS Press.

[14] Bruno Carpentieri, "Learning how to compress from correlated examples: the lossless case",

WSEAS Trans. on Systems, Vol. 2, n. 4, Oct. 2003, pp. 856-860, WSEAS Pub..

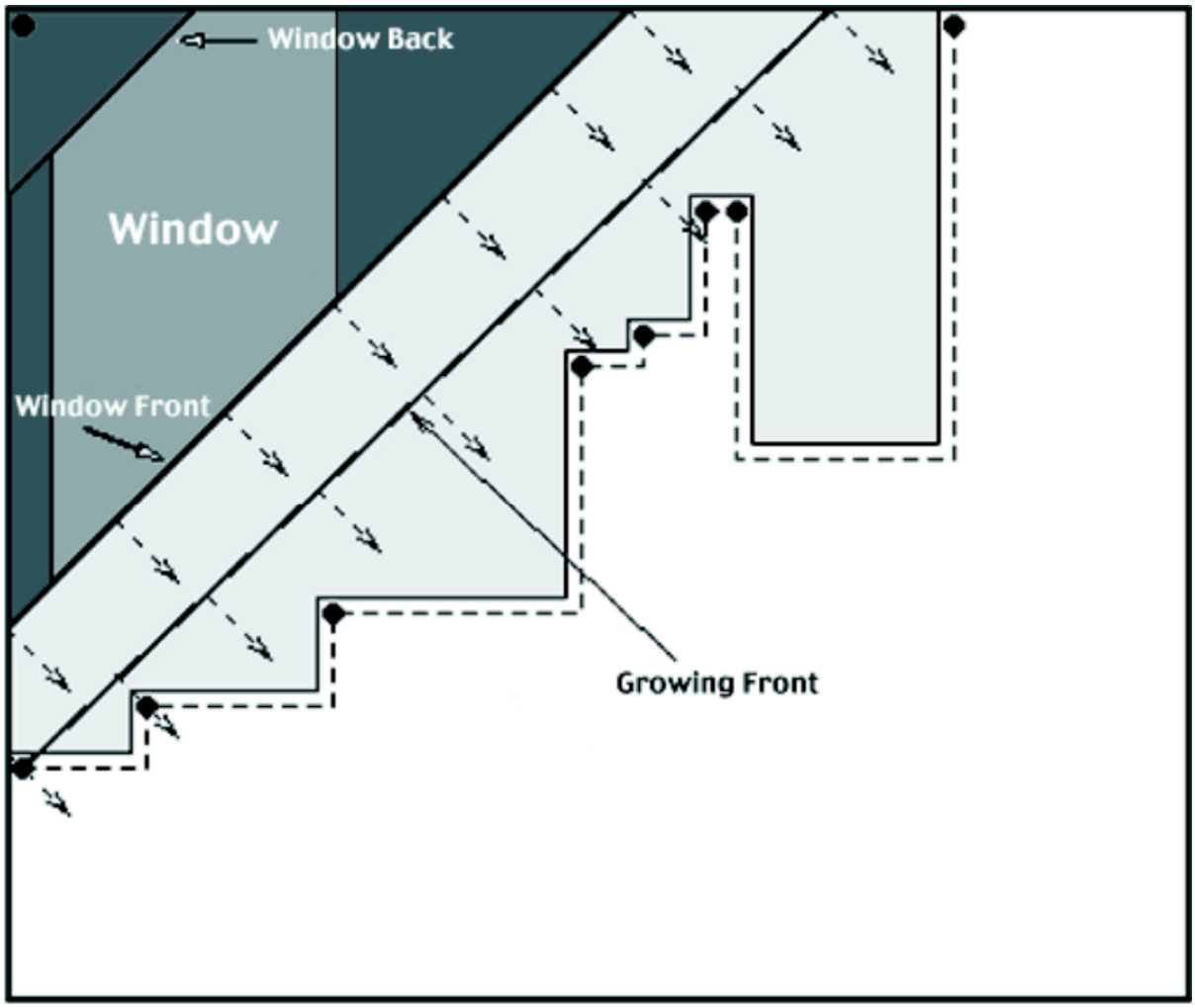

Fig. 1: Two dimensional Sliding Window

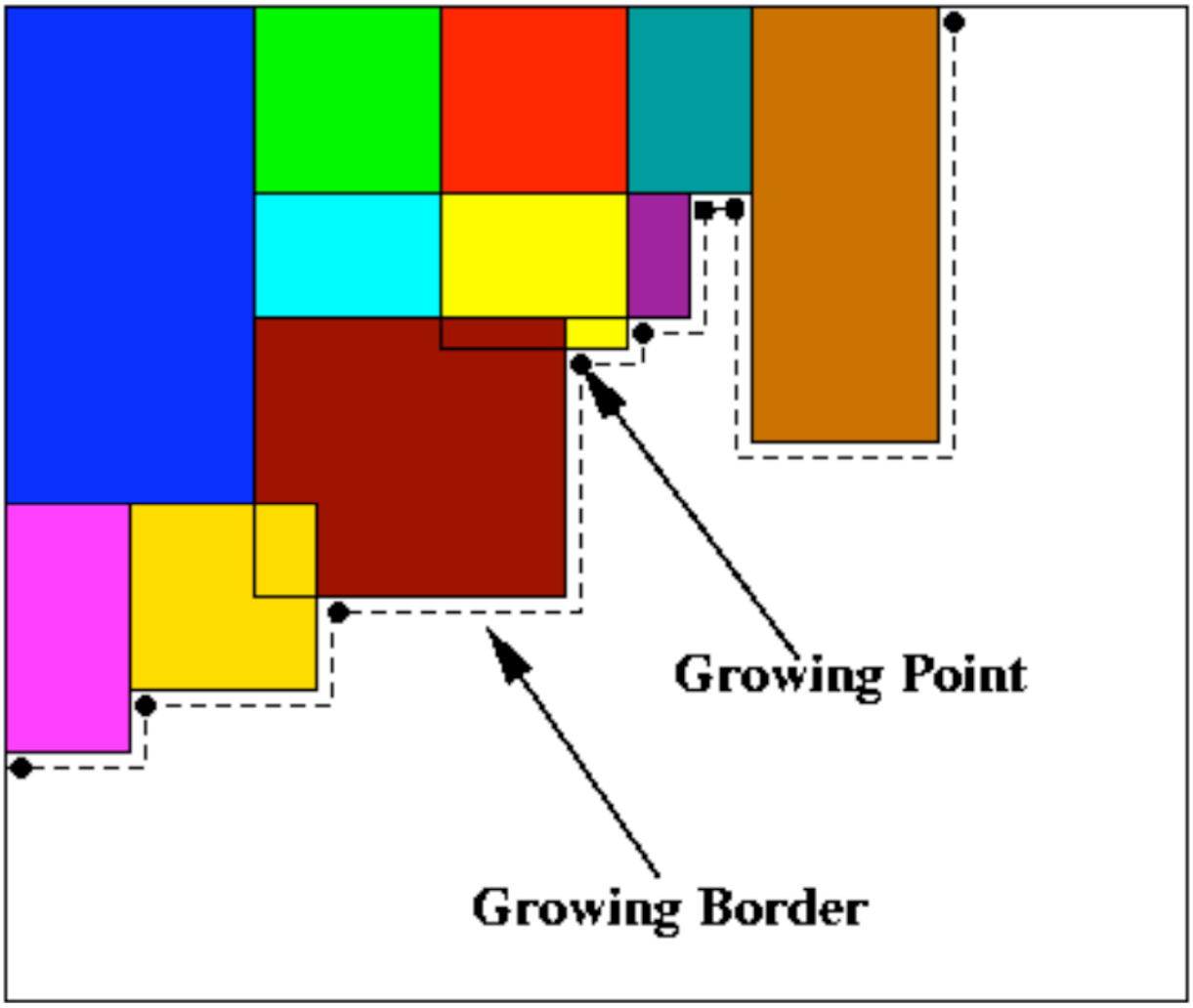

Fig. 2: Growing Border and Growing Points

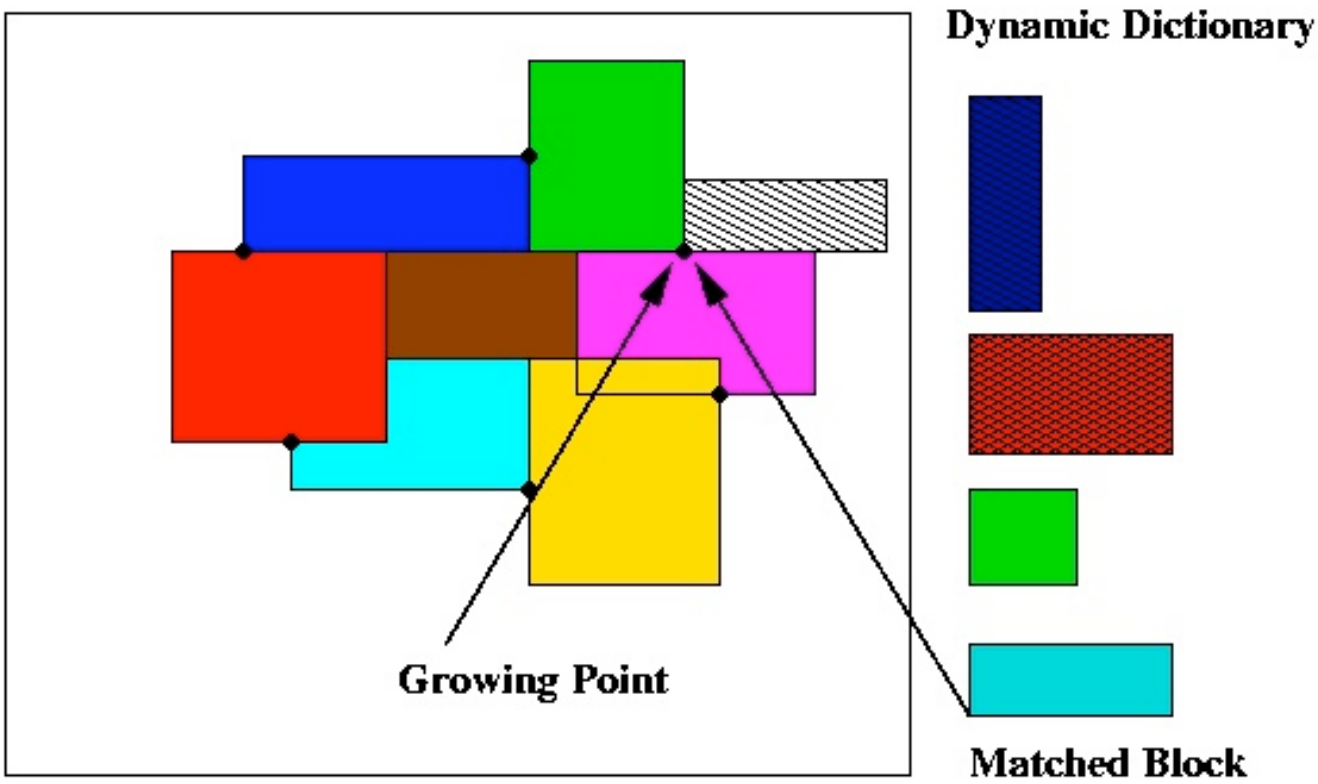

Fig.3: *Circular* growing heuristic

Fig.4: *Lena* original image and *Lena* decompressed after 1 byte in the input stream has been corrupted

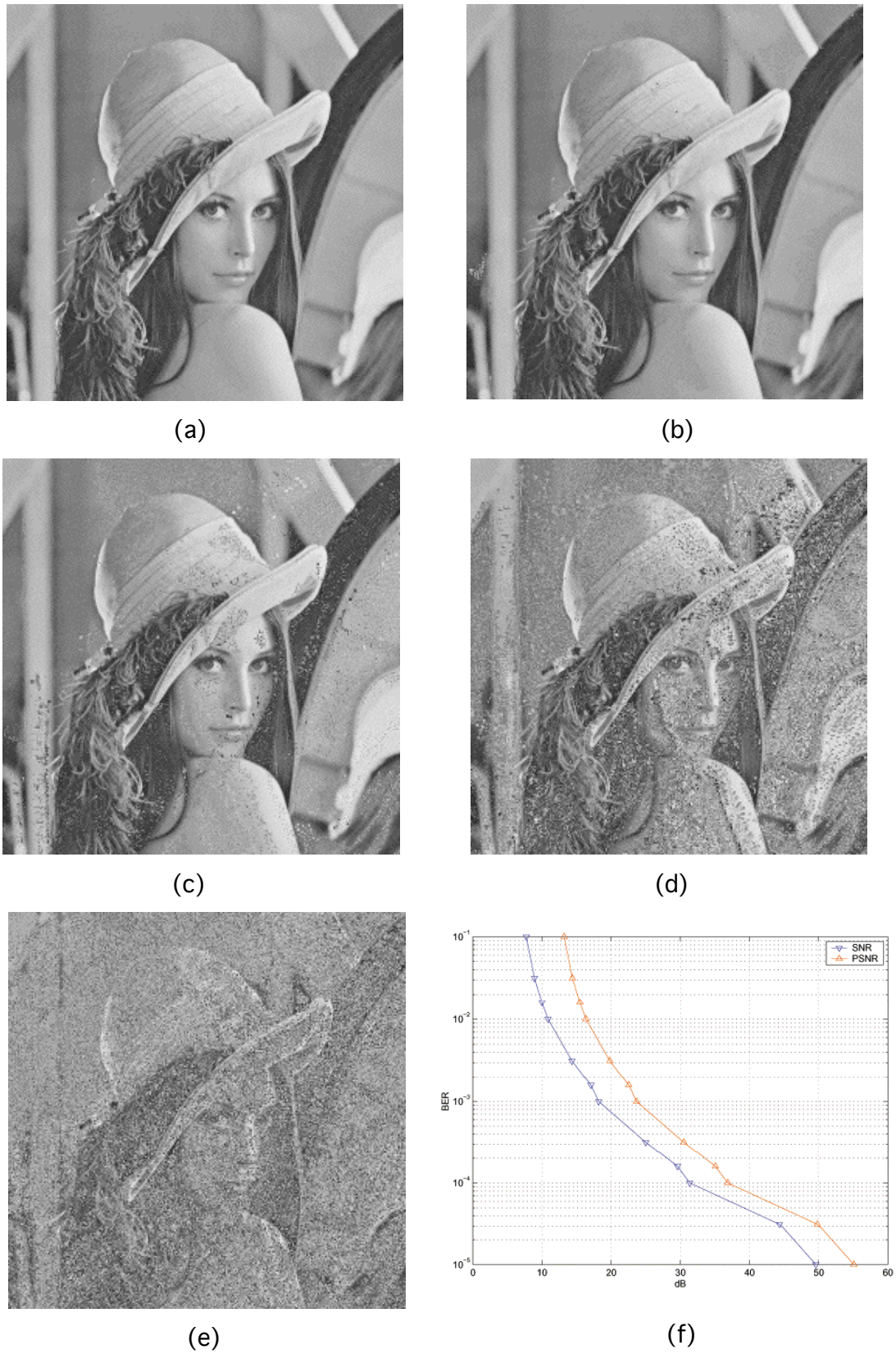

Fig. 5: AVQ-MD: degradation when channel errors affect the match pointer (average over 100 runs). (a) BER 10^{-5}, (b) BER 10^{-4}, (c) BER 10^{-3}, (d) BER 10^{-2}, (e) BER 10^{-1}, (f) BER vs SNR/PSNR