Acta Cryst. (1998). D**54**, 184–199

Data Compression for Diffraction Patterns

J.-L. FERRER,[a]* M. ROTH[a] AND A. ANTONIADIS[b]

[a]*IBS J.-P. Ebel CEA-CNRS/LCCP, 41 avenue de Martyrs, 38027 Grenoble CEDEX 1, France, and* [b]*IMAG-LMC, University Joseph Fourier, BP 53, 38041 Grenoble CEDEX 9, France. E-mail: ferrer@lccp.ibs.fr*

(*Received 26 December 1996; accepted 13 May 1997*)

Abstract

Efficient coding (lossless) and compression (lossy) of diffraction patterns is important in protein crystallography experiments because of storage and transmission limitations. The goal is to reduce the bit-rate significantly while keeping diffraction peak intensity distortion at an acceptable level. This paper presents an overview of coding and compression techniques more or less adapted to such problems. A large part of this study is dedicated to time-frequency-transform based compression algorithms and some of their extensions. Wavelet based software has been developed and tested. Results are compared with the discrete cosine transform (DCT) and other classical algorithms. These tools seem attractive and very promising for analyzing and compressing signals with singularities and transient phenomena such as diffraction peaks. Tests were performed on a standard protein crystallography data set coming from the CCD detector of D2AM beamline at the European Synchrotron Radiation Facility at Grenoble. These were compressed with DCT and wavelet-based algorithms. It appears that alterations of the result of the processing of restored images remain very weak for compression rates up to 10. These preliminary results indicate that the proposed wavelet method is a good standard technique for efficient compression of diffraction patterns.

1. Introduction

Crystallography of biological macromolecules, and more generally experiments using two-dimensional detectors, provide a huge amount of data, with an increasing data-collection rate because of the development of third-generation synchrotron sources and fast detectors. The volume and rate of these data become a major problem that the fast progress of hardware (networks, CPU, hard disks, *etc.*) cannot solve.

In this context, the development of compression algorithms, adapted to each kind of data, becomes a major challenge. Typical diffraction-pattern images consist of a flat background sprinkled with diffraction peaks. Compression of such images, usually much more efficient than simple traditional coding, unavoidably leads to a loss of information. Since these images are subjected to stringent quantitative analysis, this loss must be estimated precisely and the compression method must be proven not to discard useful information.

This paper gives an overview of available coding (§3) and compression methods (§4). A large part of this study is dedicated to wavelet transform based compression algorithms and some of their extensions. Indeed, these tools seem attractive and very promising for analyzing and compressing signals with singularities and transient phenomena such as diffraction peaks. Results on protein crystallography data coming from the CCD detector of the D2AM beamline (Roth, Ferrer, Simon & Geissler, 1992; Simon *et al.*, 1992; Fanchon, Ferrer, Kahn, Bertet & Roth, 1995) at the European Synchrotron Radiation Facility at Grenoble (ESRF) indicate that the proposed wavelet methods excel over standard techniques for efficient compression.

2. Data analysis

2.1. *Volume of data*

The development of fast two-dimensional detectors for crystallography experiments has drastically reduced the experiment time costs. As a consequence, the data flow of these experiments, when coupled to intense radiation sources, has strongly increased. A multiwavelength anomalous diffraction (MAD) experiment on biological macromolecules is certainly one of the most demanding experiments with respect to the data volume and rate problem. Indeed, it consists of a repetition, at several wavelengths, of a typical crystallography experiment. Moreover, as the studied object (the biological macromolecule) is very complex (several thousand degrees of freedom), each of these data sets, which includes this information as intensities of diffraction peaks, is very large. Typically, we need about 1800 frames for a 'three-wavelength' MAD experiment with 0.3° rotation per frame and 180° of total rotation range (required to get the maximum of completeness for $P2$ crystal symmetry for example). With a 1024 × 1024 pixel 16-bit dynamic detector each frame requires 2M bytes. The total storage capacity required for this experiment is 3.6G bytes. The storage and the transfer and backup of these data become a real problem. Indeed, while this experiment can be performed in about 10 h on the D2AM beamline at the ESRF, the time requested for data transfer is presented in Table 1.

Table 1. *Time requested for transfer and backup of a typical data set.*

The compression rate (R) obtained with this data set for the DAT with build-in compression (LZ based algorithm) is about 2.

Hardware	Theoretical rate	Tested rate (bytes s^{-1})	Time
Local Ethernet link	10M bits s^{-1}	500K	2 h
Shared Ethernet network	10M bits s^{-1}	10K	4.2 d
Local ATM link (Raid HD)	115M bits s^{-1}	5M	12 min
Local ATM link (std. HD)	115M bits s^{-1}	2M	30 min
90 m DAT backup	183K bytes s^{-1}	167K	6 h
90 m DAT backup with compression	$R \times$ 10K bytes s^{-1}	334K	3 h
120 m DAT backup	510K bytes s^{-1}	450K	2 h 14 min
120 m DAT backup with compression	$R \times$ 510K bytes s^{-1}	900K	1 h 7 min

Therefore, it is obvious that data compression becomes a necessity for short-time handling and long-time backup of the data.

2.2. *Statistical analysis, relevant information*

Compression methods with no loss of information, *i.e.* fully reversible lossless compression, can be used without any knowledge of the useful information in the image. However, the compression rate (the ratio between the sizes of initial and compressed files) depends strongly on the statistics of the image and cannot be continuously monitored. We shall refer to such invertible methods as 'lossless coding' or 'lossless redundancy removers' because all the information in the original image can be recovered and the compression is achieved by removing redundant information (§3). In contrast, lossy compression suppresses redundancy but also a 'small' part of the information. Such a process allows a tunable compression rate, but without guarantee for relevant information which must be analyzed accurately and a comparison made between initial and rebuilt data (§4). Therefore, a statistical analysis of the image is the first step in choosing a compression method. Indeed, images of interest are not purely stochastic and exhibit redundancy. There are three types of redundancy (Hilton, Jawerth & Sengupta, 1994): spatial redundancy (correlation between adjacent pixels in direct space or band limited in Fourier space), spectral redundancy (correlation of spectral values of the same pixel location) and temporal redundancy (low change for the same pixel in adjacent frames of a video sequence).

A sequence of diffraction patterns exhibits all these kinds of redundancy. Indeed, a diffraction pattern consists of a set of narrow diffraction peaks sprinkled on a low-intensity background (see Fig. 1). A correlation diagram of adjacent pixels exhibits these two populations as two sets of spots close to the main diagonal (see Fig. 2). However, the short-range correlation which appears in this diagram decreases rapidly at a larger distance (see Fig. 3). Moreover, each value in the dynamic range does not appear with the same probability, as shown by a histogram (see Fig. 4). This last point can be expressed by means of the Shannon entropy which is maximal when every value of the intensity appears with the same probability and vanishes when all pixels have the same value (see §3 and §5). The last form of redundancy, the temporal redundancy, is obvious for a data collection which consists of a lot of diffraction patterns performed at closed orientations of the sample: across the stack of frames each peak appears and disappears slowly.

As described above, diffraction patterns consist mainly of a background and sometimes there is a diffraction

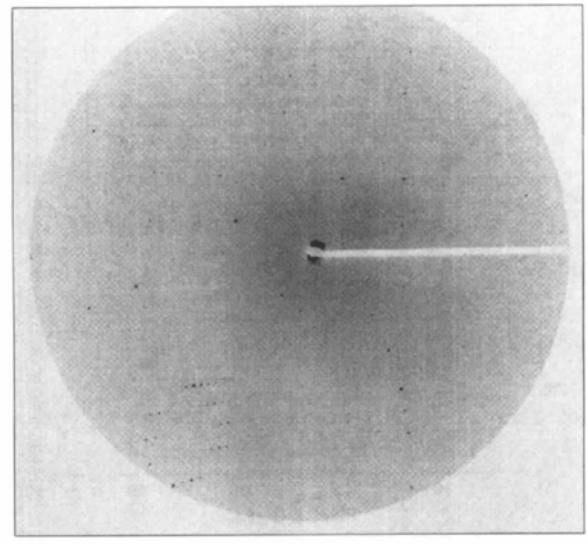

Fig. 1. Diffraction pattern obtained with a crystal of lithostathine protein on the D2AM beamline at the ESRF (test image).

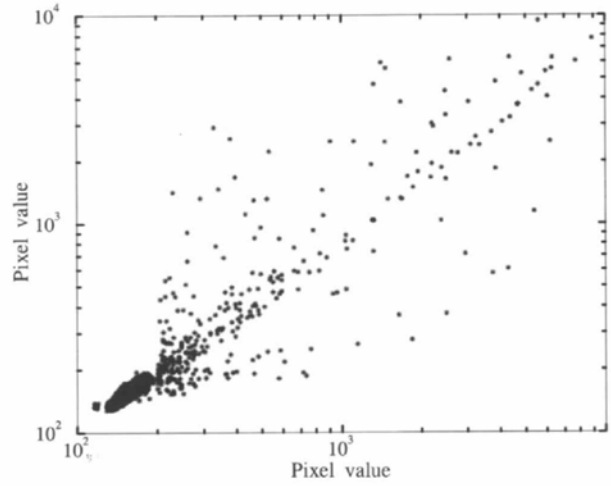

Fig. 2. Pixel-to-pixel correlation of the test image (see Fig. 1): each pixel is plotted *versus* its closest right neighbor.

peak. The pertinent information concerns the position and intensity of these peaks corrected for background and an estimation of the variance of this intensity. The intensity is computed by integration (sum of the value of pixels) after removing of the background estimated by local averaging techniques. This processing is very close to astrometric and photometric measurements on astronomy images (White, Postman & Lattanzi, 1991). For such images which are subjected to careful quantitative analysis, visual criteria (Watson, Yang, Solomon & Villasenor, 1996; Algazi, Kato, Miyahara & Kotani, 1992) are not relevant. In our case, the comparison of peak intensities, calculated on the original and the reconstructed image, will be the absolute criteria for estimating the efficiency when irreversible compression algorithms are applied.

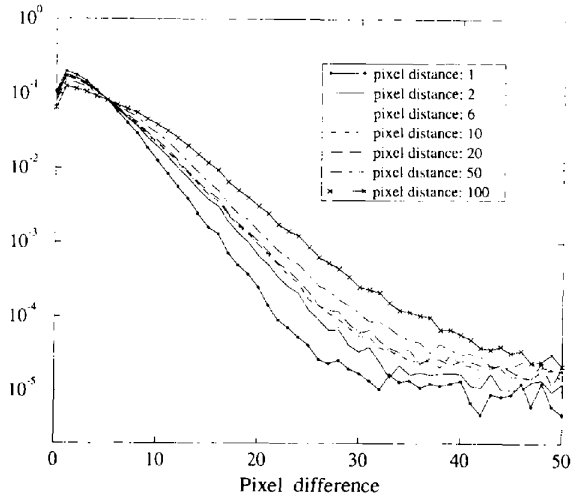

Fig. 3. Histograms of the pixels differences for close pixels of the test image (probability of the difference for pixel distances from 1 to 100 pixels).

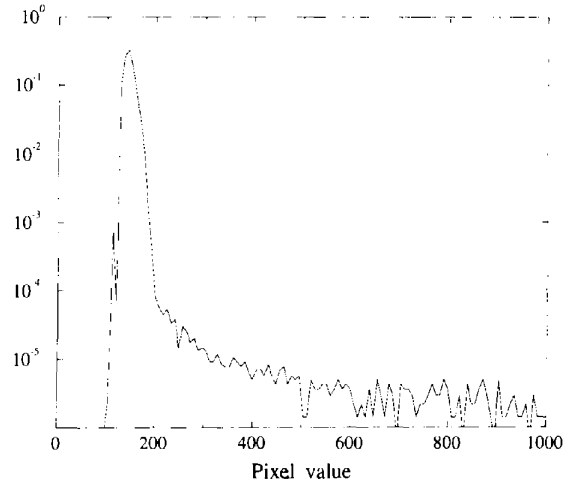

Fig. 4. Histogram of the test image (see Fig. 1): 99.9% of the pixels are in the 100–200 range.

3. Coding (lossless compression)

Reversible compression methods generally consist of a modeling process followed by coding. The set of both processes is usually called 'coding'. Modeling is performed by a collection of rules which associate code to an entry datum or set of data in a more efficient way. This efficiency is expressed as the maximum decrease in the size of output data compared with the size of input data.

The easier coding process, the linear predictor method, consists of coding the difference between the value of a pixel, in the case of an image, and an estimated value, calculated from adjacent pixels, or *versus* the value of the same pixel in previous images in the case of a set of similar images. This reduces the dynamic range and, thus, the required number of bits for coding. In the case of noisy images, like diffraction patterns, this leads to a very low compression rate, compared with the one obtained with the methods that follow.

Historically, the first efficient general purpose coding processes were the Shannon–Fano and the Huffman coding process (Huffman, 1952). The idea is to associate the shortest code to frequently encountered input symbols in the input stream as close as possible to the optimal length estimated by Shannon,

$$\text{number of bits} \simeq -\log_2(\text{occurring probability}). \quad (1)$$

Rapidly, the first step, which consists of a statistical study of the overall data set – the result must be transmitted to the uncoding process – was replaced by an adaptive process which evaluates probabilities of symbols and performs coding during the same scanning process. The compression rate, low at the beginning, increases rapidly. This is the adaptive Huffman coding. It leads to the best compression rate with integer size coding, estimated by the Shannon entropy,

$$\text{minimum length} \geq -N \sum_i (n_i/N) \times \log_2(n_i/N), \quad (2)$$

where n_i denotes the number of input symbols equal to i, and N is the total number of symbols. The arithmetic coding, close to the Huffman coding, eliminates the limitation of the integer size of the previous coding and leads to the output size given by expression (2).

The above coding processes exploit the non-uniformity of the input symbols statistic. Dictionary modeling-based coding processes improve the compression rate by exploiting the non-uniform statistics of input sequences of symbols, thus reducing the spatial redundancy (see §2). These algorithms were developed by Ziv, Lempel & Welch (Ziv & Lempel, 1977, 1978; Welch, 1984) and are called LZ77, LZ78 (dictionary with fixed-length window which is translated along the symbol stream during the scanning) and LZW algorithms (the dictionary is a logical tree which is built character per character). They all are adaptive coding processes where each known

sequence occurring in the input stream is replaced by its index in the dictionary. The dictionary is set up to date whenever an unknown sequence appears.

Recently, two new types of algorithms are becoming very popular. The first one is based on the so-called finite context modeling. The second one consists of block sorting of data. Both are usually coupled to an arithmetic coding or some other classical redundancy remover.

Table 2 recalls the common implementations of the previously discussed algorithms (see references in §6).

Table 2. *Common implementations of some algorithms for data compression*

Algorithm	Particularity	Usual name	Environment
Huffmann	Adaptive Huffman code	*compact*	UNIX
	Huffman code, byte-by-byte	*pack*	UNIX
Lempel–Ziv (LZ77)		*PKZIP*	PC
		gzip	UNIX
Lempel–Ziv & Welch		*compress*	UNIX
		ARC	PC
		PKWare	PC
	Japan	*Lharc*	PC
		ARJ	PC
	Graphical format	*GIF*	All
		MNP-5	Modems
	CCITT standard	*V42bis*	Modems
Finite context modeling		*HA (HSC)*	UNIX
		Stat	UNIX
		CALIC	UNIX
Block sorting		*BWT*	PC, UNIX
		bzip	UNIX

4. Lossy compression

4.1. *General principle*

The key of transform-domain data compression methods is based on finding a signal or image representation, preferably one computable *via* a fast transform algorithm, that provides the ability to represent complicated signals or images accurately with a relatively small number of bits. A first step is to find a transformation which gathers relevant information on a small and intense part of the output signal. The output of such a transform may be a set of coefficients with only a few of them being large and uncorrelated. The efficiency of such compression methods depends mainly on this first step: the choice of the transform. No standard solution exists, and this choice must usually be adapted to the signal to be compressed. Efficiency of the representation can then be measured by several numerical criteria (cost functions). Some of them exhibit additive properties, which reduce the algorithmic complexity. Four of these additive criteria, used in our tests, are listed below. We use the notation $\mathbf{x} = \sum_i c_i \mathbf{e}_i$ to denote the decomposition of a vector $\mathbf{x}$ on an orthonormal basis $\{\mathbf{e}_i\}$, and t to denote a given threshold.

Shannon entropy:

$$S = -\sum \left(|c_i|^2 / ||x||^2 \right) \log \left(|c_i|^2 / ||x||^2 \right),$$

where $||x||^2 = \sum |c_i|^2$ and $|c_i|^2 \log |c_i|^2 = 0$ for $c_i = 0$.

Bit-length norm:

$$S = -\sum \log[1 + (|c_i|/t)].$$

Density of representation:

$$S = \sum I(|c_i| > t),$$

where I denotes the indicator function.

Log energy:

$$S = -\sum \log \left(|c_i|^2 / ||x||^2 \right) \text{ for } (c_i \neq 0),$$

where $||x||^2 = \sum |c_i|^2$.

The second step of a lossy compression scheme consists in replacing the real number coefficients c_i of the image's expansion obtained by the transform with lower precision approximations which can be coded in a (small) finite number of digits. If the transform step is effective, then the new coordinates are mostly very small and can be set to zero, while only few coordinates are large enough to survive. This is usually done by thresholding or 'quantization'. For example, if (c_i) represent the expansion coefficients of an image s, we may find that keeping only n coefficients leads to a reconstruction of s with a negligible error. Moreover, instead of storing the large c_i as floating-point numbers, one may round them off to a short fixed-point representation typically through a quantization process. This step corresponds to the practical notion of quantization. The output of thresholding and quantization is a stream of small integers, most of which are the same (namely 0). The loss of information in the recovered image $\hat{s}$ after thresholding and quantization is mainly because of these procedures. Several strategies $\hat{c}_i = \eta(c_i)$ for thresholding coefficients may be used. We list below the most usual ones.

Linear hard thresholding,

$$\eta(c_i) = c_i I(|c_i| > t),$$

where $t = a \cdot \max(|c_i|)$ and I denotes the indicator function.

Logarithmic hard thresholding,

$$\eta(c_i) = c_i I(|c_i| > t),$$

where, this time, $t = a \cdot \left| \log[\max(|c_i|)] \right|$ (a is an user-defined positive coefficient).

Soft thresholding,

$$\eta(c_i) = \operatorname{sgn}(c_i)\,(|c_i| - t)_+,$$

where $(u)_+ = \max(0, u)$ and t is a chosen fixed threshold.

One can summarize the thresholding process to a setting which evaluates inexactly the coefficients of s with the aim of reconstructing s itself with a minimal loss. The thresholded coefficients are then quantized before being stored. The range of transformed values is divided into numbered subintervals or bins. Any value falling into a bin is approximated by the bin's index. Quantization is undone by replacing the bin index with the value at the center of the bin. An example is uniform quantization. One chooses a quantum q and discretizes the floating point thresholded coefficients by recording, instead of $|\hat{c}_i|$, the integer ℓ such that $2\ell q$ is the closest to $|\hat{c}_i|$ among all such multiples. The reconstruction error is then, at most, of size q in the retained coordinates and of size c_i in the discarded coordinates.

In most cases, lossy compression includes a third step. This final step consists of removing redundancy with a coding process (see §3). Its aim is to replace the stream of small integers that must be stored with a more efficient alphabet of variable-length characters. It takes advantage on the large number of zero or small coefficients produced by the previous step.

Most of the algorithms which are tested below (§6) are designed according to these three steps (see Fig. 5).

4.2. *Frequency analysis*

The role of transform-domain data compression has traditionally been filled by the fast Fourier transform and its related fast trigonometric transforms, particularly the discrete cosine transform (DCT) (see *e.g.* Rao & Yip, 1990). An important class of alternatives is provided by wavelet transforms. The purpose of this subsection is to briefly review the main ideas behind such transforms.

4.2.1. *Fourier transform*. The best known transform is the Fourier transform for $s(t) \in L^2(R)$,

$$s(t) \to \int_{-\infty}^{+\infty} s(t) \exp(-2i\pi\omega t)\, \mathrm{d}t.$$

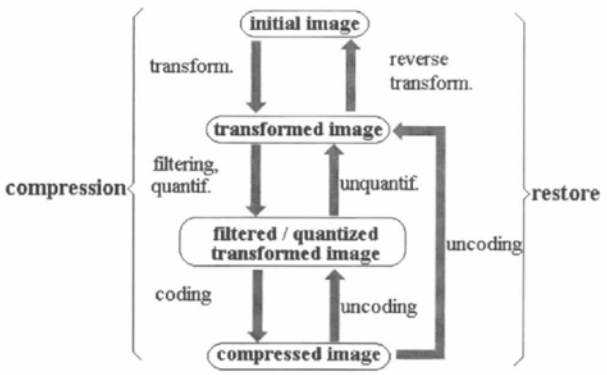

Fig. 5. General algorithm of compression, including three main steps: (i) transform; (ii) filtering, like thresholding or quantization (unreversible operations); and (iii) redundancy removal.

Limitation of this transform comes from the lack of spatial localization: localized information of $s(t)$ is spread all along the Fourier space and reciprocally. As a consequence, all coefficients are necessary with the same accuracy to build back locally the initial signal. Fourier transform is global and provides a description of the overall regularity of signal. Transform-coefficient quantization strategies tend to sacrifice high-frequency signal content to preserve more important lower frequency information, and if no space-frequency localization is imposed on a Fourier decomposition, the universal loss of high-frequency content results in Gibbs artifacts. The use of this transform for compression of functions with local features is, therefore, inefficient. Some form of tiling or windowing is, therefore, an unavoidable component of any Fourier-based image coding algorithm.

4.2.2. *Windowed Fourier transform*. To improve and localize the properties of the Fourier transform, the oscillating vector $W_\omega(t) = \exp(-2i\pi\omega t)$ can be multiplied by a compactly supported, or at least rapidly decreasing, function (window function) $\Psi(t)$. In one dimension this transform can be depicted as in Fig. 6 below.

The scalar product of a signal $s(t)$ with $\Psi(t-i)W_\omega(t)$ provides the ω-frequency component of $s(t)$ within Ω_i, the compact support of $\Psi(t-i)$, which is called the instantaneous frequency component. A particular choice of Ψ leads to a basis of the space $L_2(R)$ (Gabor, 1946). A decomposition into this basis generates a time-frequency transformation where localized properties of the initial signal are kept efficiently. However, this windowed Fourier transform has no orthonormal properties (orthogonality is convenient for treating adequately an eventually noisy signal) and is limited by the fixed scale of $\Psi(t)$.

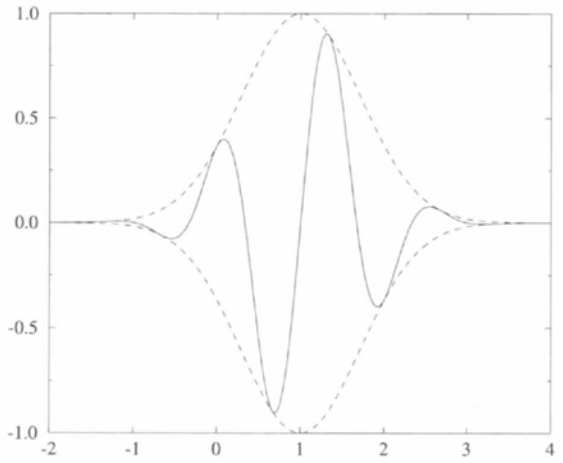

Fig. 6. In the windowed Fourier transform, the classical oscillating vector $W_w(t)$ of the Fourier transform is multiplied by the window function (dashed line) $Y(t-t_0) = \exp[-(t-t_0)^2]$, here at the location $t_0 = 1$.

A discrete form of such a windowed transform is the discrete cosine transform (DCT) which is used in the image compression standard drafted by the International Standards Organization's Joint Photographic Experts Group (known as 'JPEG') (Pennebaker & Mitchel, 1992). The JPEG compression standard is based on partitioning a digital image into 8 × 8 pixel blocks, applying a two-dimensional discrete cosine transforms to each block and compressing the output of each eight discrete cosine transform by applying an integer division quantization matrix. Even at moderate compression rates, the JPEG algorithm sometimes produces highly objectionable 'tiling artifacts' resulting from boundary mismatches between the quantized low-frequency cosine modes in adjacent tiles.

In the case of our diffraction patterns, the tiling artifacts and edge effects inherent in JPEG image coding may be vexatious since they affect the intensity of diffraction peaks occurring between two adjacent boxes.

In contrast, decompositions based on compactly supported wavelets achieve simultaneous space–frequency localization with no need for windowing and are particularly well suited for compressing high-resolution images (Wickerhauser, 1992), so let us briefly review the mathematical ideas behind wavelet transforms.

4.2.3. *Wavelet transform.* A wavelet series (or 'multiresolution decomposition') is a scale-based decomposition of a signal into a linear superposition of dilates and translates of two special functions: a scaling function, φ, which carries the mean value and other low frequency behavior of the signal, and a mother wavelet, ψ, which has mean 0 and encodes details of the signal from different length scales. Recall that the span $\{e_j; j \in Z\}$ denotes the linear space of all finite linear combination of e_j's, *i.e.* of the form $\sum_{j\in J} c_j e_j$ where J is a finite subset of Z. One defines spaces in terms of the dilates and translates of the two functions φ and ψ,

$$V_j = \overline{\text{span}}\{\varphi_{j,k}(x) = 2^{j/2}\varphi(2^j x - k);\ k \in Z\}$$

$$W_j = \overline{\text{span}}\{\psi_{j,k}(x) = 2^{j/2}\psi(2^j x - k);\ k \in Z\},$$

the overbar denoting the closure. The scaling spaces, V_j generate a multiresolution approximation of $L_2(R)$,

$$V_j \subset V_{j+1},$$

and the wavelet spaces W_j, 'fill the gaps' between successive scales,

$$V_{j+1} = V_j \oplus W_j,$$

where $\oplus$ denotes the direct sum of subspaces.

In particular, we can start with approximation on some nominal scale, say V_0, and then use wavelets to fill in the missing details on finer and finer scales,

$$L_2(R) = V_0 + \oplus_{j=0}^{\infty} W_j.$$

Multiresolution analysis also holds the key to construct scaling functions and mother wavelets: since $\varphi \in V_0 \subset V_1$, it follows that the scaling function for a multiresolution approximation may be obtained as the solution of the two-scale dilational equation,

$$\varphi(x) = \sum_k h_0(k)\varphi(2x - k), \tag{3}$$

for some suitable sequence of coefficients, $h_0(k)$. Once φ has been found, an associated mother wavelet is given in a similar-looking recipe,

$$\psi(x) = \sum_k h_1(k)\varphi(2x - k). \tag{4}$$

Of course some effort is required to produce appropriate coefficient sequences, $h_0(k)$ and $h_1(k)$. Daubechies (Daubechies, 1992) developed such sequences with finite length that produce compactly supported functions φ and ψ with whatever regularity one needs.

Another interpretation of a multiresolution analysis is to see it as a description of the analyzed signal s in terms of its 'local averages' (the terms in V_0) and its 'local details' (the terms in the W_j spaces). The decomposition may or not be orthogonal; in the orthogonal case (which includes the compactly supported Daubechies wavelets) the coefficient of the wavelet decomposition of a signal s are given by,

$$s_{j,k} = \langle s, \varphi_{j,k}\rangle = \int_R s(t)\varphi_{j,k}(t)\,\mathrm{d}t$$

$$d_{j,k} = \langle s, \psi_{j,k}\rangle = \int_R s(t)\psi_{j,k}(t),\,\mathrm{d}t.$$

Given these coefficients, the signal s can be written as,

$$s = \sum_{k\in Z} s_{0,k}\varphi_{0,k} + \sum_{j\geq 0}\sum_{k\in Z} \mathrm{d}_{j,k}\psi_{j,k},$$

or, in a more symbolic way,

$$S = S_m + \sum_{j=0}^{m-1} D_{m-j}. \tag{5}$$

There is also a discrete version of multiresolution analysis for sampled data. Prior to the discovery of continuous wavelets, multiresolution transform methods have been studied in the field of digital signal processing under the name of multirate filter banks. Filter banks provide fast effective means of separating a digital signal into different frequency components. When the filter is finite (as in the case of compactly supported wavelets), this frequency separation is accomplished using only local computations (as compared to non-local methods like the discrete-time Fourier transform).

One of the big discoveries for both the wavelet and filter banks theories was that distortion-free filter banks

can be formed using the coefficient sequences, h_0 and h_1, from the two-scale dilation equations [(3), (4)] for a multiresolution approximation. By cascading (*i.e.* composing) the analysis bank with itself a number of times, one can form a digital signal decomposition with dyadic frequency scaling, known as a discrete wavelet transform (DWT), leading to fast transformation algorithms that are even faster than the discrete fast Fourier transform,

$$\begin{aligned} s_{j,k} &= \sum_{i=0}^{n-1} h_0(i) s_{j+1,2k+i}, \quad k = 0 \ldots 2^j - 1 \\ d_{j,k} &= \sum_{i=0}^{n-1} h_1(i) s_{j+1,2k+i}, \quad k = 0 \ldots 2^j - 1 \end{aligned} \tag{6}$$

where $s_{j,k}$ is periodized ($s_{j,k+2^j} = s_{j,k}$) and n is the finite length of the coefficient sequences h_0 and h_1. Reconstruction is performed with,

$$\begin{aligned} s_{j,k} &= \sum_{i=0}^{n-1} h_0(2k - i) s_{j-1,i} + \sum_{i=0}^{n-1} h_1(2k - i) s_{j-1,i+2^{j-1}}, \\ k &= 0 \ldots 2^{j-1} - 1 \end{aligned} \tag{7}$$

For the Daubechies-2 wavelet transform (the so-called Haar transform, with $n = 2$), coefficients are,

$$\begin{aligned} h_0(0) &= 1/2^{1/2}, h_0(1) = 1/2^{1/2} \\ h_1(0) &= 1/2^{1/2}, h_1(1) = -1/2^{1/2}. \end{aligned}$$

The interested reader can find a textbook exposition on the subject by consulting, for example, Meyer (1992).

Fig. 7 illustrates the DWT principle. For a discrete signal $\{s_i\}_{i=1,\ldots,2^m}$, the recursive computation of scale detail coefficients from the smallest scale $2^0 = 1$ (the initial function) up to the largest scale 2^m can be described with a two-dimensional pyramid scheme (for $m = 3$). Each scale function is split into a scale and a detail function, and so on. The discrete wavelet transform is defined as the sum of details and the lower scale coefficients.

4.2.4. *Wavelet packet analysis*. The wavelet packets method is a generalization of wavelet decomposition methods that offers a wider range of possibilities for signal analysis and compression.

In wavelet analysis a signal is split into an approximation and a detail. The approximation is then itself split into a second-level approximation and detail and the process is repeated [see equation (5)]. For an m-level decomposition there are $m + 1$ possible ways to encode a signal,

$$\begin{aligned} S &= S_1 + D_1 \\ &= S_2 + D_2 + D_1 \\ &= S_3 + D_3 + D_2 + D_1. \end{aligned}$$

In wavelet packet analysis, the details as well as the approximations are split. One then obtains a wavelet decomposition binary tree (see Fig. 8). The leaves of every connected binary subtree of the wavelet packet tree correspond to an orthogonal basis. For a finite energy signal any wavelet-packet basis will provide exact reconstruction and will offer a specific way to encode the signal. For instance, a wavelet-packet analysis allows the signal $S = S_0$ to be represented as $S_1 + SSD_3 + DSD_3 + DD_2$ which is an example of representation that is not possible with ordinary wavelet analysis. Based on the organization of the wavelet-packet library, a signal of length $N = 2^m$ can be expanded in more than 2^N different ways. Indeed, if we denote by U_n the number of different ways to encode the signal S up to the n level ($n < m$), we then get,

$$U_{n+1} = U_n^2 + 1.$$

As this number may be very large, and since explicit enumeration is generally unmanageable, choosing one

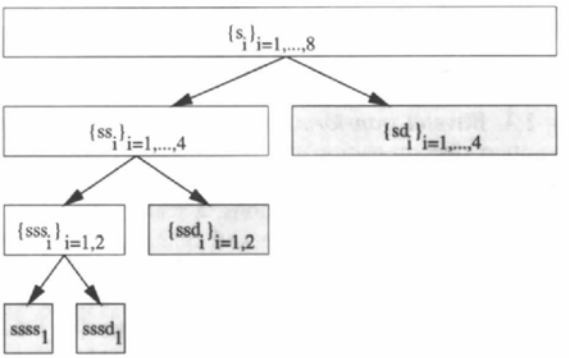

Fig. 7. Discrete wavelet transform: illustration of the recursive computation for an eight coefficient long initial signal. The gray boxes contain coefficients of the DWT, made from the detail coefficients at each scale and the lowest scale coefficient (bottom-left box).

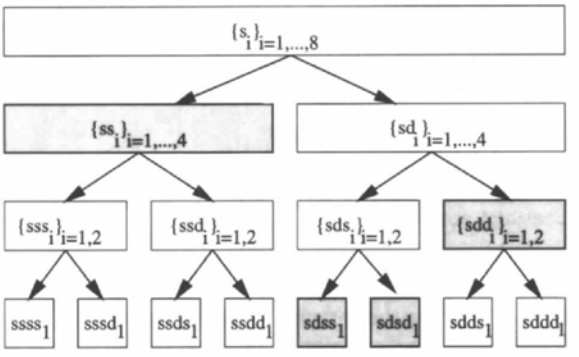

Fig. 8. Wavelet packet analysis. At the opposite of the DWT, each set of detail coefficient is split again into scale and detail coefficients. This lead to several wavelet basis, each one made from the choice of sets of coefficient to cover the initial signal. In gray, one of such a basis.

out of all these possible ways to encode a given signal leads to an interesting NP-hard problem (Davis, 1994). It is necessary to find an optimal decomposition with respect to a convenient criterion, computable by an efficient algorithm. One may use a cost criterion, and choose an appropriate decomposition by looking at each node of the wavelet-packet tree and quantifying the information to be gained by performing each split. Cost criteria relying upon functionals with an additive-type property are well suited for such an optimal decomposition. Classical entropy-based cost criteria match quite well these conditions (Coifman, Meyer & Quake, 1989).

The choice of the 'best' decomposition in a wavelet-packet tree is an example of a more general problem: find an optimal representation of a signal as a linear combination of elements forming a library of bases. Another algorithm available for solving such problems is the matching pursuit algorithm (Mallat & Zhang, 1993). The inconvenience of both these algorithms, compared with a classical wavelet decomposition, is that the indexes of the basis vectors that are chosen must be stored for an efficient reconstruction. The decrease of the required number of coefficients that are necessary for a given quadratic error (QDRI see §5.2) of the representation is compensated by the increase of stored information, and globally the compression rate decreases, at least for the images we are dealing with in this paper.

5. Test setup

5.1. *Image description*

Lossless and lossy compression methods described above have been tested with our protein crystal diffraction patterns. They were produced by a XRII-CCD camera on the D2AM beamline at the ESRF (European Synchrotron Radiation Facility, 1994). This camera, developed at the ESRF (Moy, 1994; Bourgeois, Moy, Svensson & Kvick, 1994), is made from a modified Thomson image intensifier and a Princeton CCD camera, coupled by lenses. This device produces 16 bit, 1242 $\times$1152 pixel images plus a 4100 byte long header (total size of an image: 2 865 668 bytes). The gain of this detector, which is the product of the fixed gain of the intensifier and the tunable losses of the lenses, has been tuned in order to be close to one. In this context, one X-ray photon at a typical energy of 12.5 keV produces about five electrons in a well of the CCD chip. These electrons are converted by an ADC to one count on a full scale of 65 535 (2 byte coding). However, the total signal of a pixel, which can be roughly considered as having Poisson statistics, is not only the result of incoming X-rays on the entrance window but can be split into three origins: (i) detector contribution (offset, dark current, read-out noise, *etc.*), (ii) parasitic events (cosmic rays, ions hitting the cathode of the intensifier); and (iii) X-rays (scattering, diffracted photons).

As a consequence, the image consists of a few sharp spots (diffraction peaks, cosmic rays) on a noisy background (scattering, dark current, read-out noise). This visual aspect hides another difference due to the origin of the counting. On one hand, the contribution of the CCD itself (dark current, read-out noise) which has no significant spatial correlation. On the other, the contribution of the intensifier (X-rays, ions and cosmic rays): due to the focusing limit, the image of a single photon hitting the entrance window is spread onto several pixels of the CCD (2 or 3 pixels FWHM). This leads to a spatial convolution of patterns by the so-called point spread function (PSF). Each part of the structure of an image, including diffraction peaks with a natural size sometimes smaller than a pixel, is broadened by few pixels. As a consequence, close pixels become highly correlated, as illustrated with the test image (see Fig. 2). This short-range correlation decreases rapidly with distance: for pixels at a distance equal to 2, the probability of obtaining a difference lower than 5 is 30% less than that for very close pixels (see Fig. 3).

5.2. *Organization of tests*

The tests performed include all or part of the following steps (see Fig. 5): (*a*) transform (§4); (*b*) filtering, quantization (irreversible steps, see §4); and (*c*) removal of redundancy (*i.e.* coding, see §3).

The result is evaluated according to two criteria: compression ratio and accuracy of reconstruction. The compression ratio is expressed as the ratio, in percent, between the sizes of the compressed and initial files. The counterpart of this compression ratio for lossy compression processes is the difference between the reconstructed image and the initial one. This is measured by four quantities: (i) the $L_1(R^N)$ distance: $\sum_{i,j}|a_{ij}-a'_{ij}|/N$, (ii) the quadratic distance: $[\sum_{i,j}(a_{ij}-a'_{ij})^2/N]^{1/2}$; (iii) the absolute difference in integration of a strong peak; and (iv) the absolute difference in integration of a weak peak, (with i and j the horizontal and vertical coordinates of the pixel and N the number of pixels per image: N = 1 430 784). The second criterion, the quadratic distance between the restored and the initial images (QDRI), will be the criterion used for graphical comparisons (see §6).

These quantities can be easily evaluated on a single image. So they are used to compare several algorithms with a great number of parameter sets for each one (§6.1 and §6.2). At the end, when few algorithms have been selected, a more relevant criterion is used (§6.3): a full set of rebuilt images is processed and the result is compared with the initial data processing. The processing is performed with the *XDS* program of W. Kabsch (Kabsch, 1988*a*,*b*, 1993). During the data processing, the R_{sym} factor is computed. This is a good factor of merit of the quality of data, and thus a good evaluation of the data alteration during the compression–decompression operation. This factor is an averaged measurement of the

difference between integrated reflections which should be identical for crystallographic reasons,

$$R_{\rm sym} = \left(\sum_{hkl} \sum_{i=1}^{N_{hkl}} |I_{i,hkl} - \langle I_{hkl} \rangle| \right) \Big/ \left(\sum_{hkl} \sum_{i=1}^{N_{hkl}} \langle I_{hkl} \rangle \right)$$

with

$$\langle I_{hkl} \rangle = \sum_{i=1}^{N_{hkl}} I_{i,hkl} / N_{hkl},$$

where N_{hkl} is the number of equivalent reflections and $\sum_{hkl}$ the sum on non-equivalent ones.

Another criterion is obtained by merging data which should be equivalent. This is commonly performed for a data set collected with two crystals of the same protein. Once the scaling is done, a mean ratio can then be computed for reflections which are shared by both data sets. This operation is performed with *Xscale*, a program of the *XDS* package.

The last criterion is the visual aspect of images. Indeed, the first evaluation of the quality of data is given by a visual check of images: diffraction peaks well contrasted, no parasitic features, *etc.* As far as possible, it is of some importance to avoid visual artifacts like the blocking effect (mosaic-like aspect due to steps from one block to another: this may occur when the algorithm splits the image in such blocks), edge effects, *etc.*

Another important parameter is the compression time, since this operation has to be performed on-line. This time, estimated by the Unix *time* command, includes reading, compression and writing of the compressed image. It is given for HP 9000/730 workstation with 1.3G bytes SCSI2 hard drives. This information is relevant for optimized software only. So, it is given for several coding methods, performed with well known software, but not for methods which are under development, as well as for software running only in an interactive mode (see Table 3 below).

Table 3. *Comparison between several lossless compression algorithms*

The Cpu time is given for a HP 9000/730 workstation.

Coding algorithm	Software name	% of initial size	Time (s)
Huffman	*pack*	45.4	2.32
Adaptive Huffman	*compact*	45.4	24.32
JBIG	*s1r4* (AT&T)	29.5	84.56
LZW	*compress*	33.1	4.00
LZ77	*gzip* 1.2.4	34.4	71.47
Difference coding	*Pack* from *CCP*4	30.9	4.02
Difference coding	*FIT2D*	26.0	—
Arithmetic coding	*ha* 0.999 (ASC)	33.6	136.2
Finite context	*ha* 0.999 (HSC)	24.6	53.17
Finite context	*Stat* package	24.6	32.33
Burrows–Wheeler Transform	*BWT* package	27.3	26.72
pl. cod. + LZ77	—	30.0	—
pl. cod. + rep. + LZ77	—	30.2	—

6. Tests

6.1. *Lossless compression*

In order to compare the algorithms and software, we have used a typical image (see Fig. 1) produced by a crystal of lithostathine protein (Bertran *et al.*, 1996). This image is part of a data set collected in 1995.

Various coding methods have been tested. Most of these tests were performed with Unix tools (*compact*, *compress*, *gzip etc.*). As expected, LZ-based coding [*compress* (Welch, 1984) software, used in our tests at the maximum compression rate and, as a consequence, the slowest mode] is much more efficient than Huffman coding (*pack* and *compact* software). A slightly better result is obtained by a bit-plane coding combined with LZ algorithm. Bit-plane coding consists of a reorganization of the data to gather bits of the same weight on close bytes. This coding can be combined with a repetition coding (a sequence of the same bit value is replaced by the number of times this value occurs, coded on 2 bytes). Such a coding is very efficient with bit-plane of high weight which is mainly at zero in our example (most of the pixels belong to the background and are lower than 200, see Fig. 4). The standardized JBIG algorithm (ISO, 1991) perform such a bit-plane coding. After dividing an image into bitplanes, compression is carried out using a Q-coder (Pennebaker, Mitchell, Langdon & Arps, 1988).

Another promising method (Burrows-Wheeler transform, see Nelson, 1966) consists of performing first a sorting and then a run-length encoding and an order-0 adaptive encoding. In contrast, diffraction patterns dedicated tools such as *Pack* (from the *CCP*4 package, see Abrahams, 1993) and *FIT2D* (Hammersley, 1995) are fast and perform a quite reasonable compression rate. Both algorithms exploit the spatial correlation between close pixels, assuming that pixels differences can be coded with a low bit rate. For example, compression is achieved by *Pack* by first calculating the differences between every pixel and the truncated value of four of its neighbors. After calculating the differences, they are encoded in a packed array, with a reduced number of bits.

The best result was obtained with a finite context model based coding. The software we used also performs an arithmetic coding on a statistical model based on a sliding window dictionary (ASC method). However, the compression rate is higher when it runs the finite context model (HSC method) (Hirvola, 1997). In any case, the compression time is quite long. Other implementations of a finite context algorithm are *CALIC* (Wu & Memon, 1995) and *Stat* (Bellard, 1995). The latter is faster than HA but requires an optimization of some parameters according to the type of image to be compressed. All of these results are gathered in the Table 3.

Table 4. *Compression tests with the DCT*

dct + ZZ(16 × 16) means that a DCT is performed with 16 × 16 block size, followed by a 'zigzag' sorting of coefficients. quant.($Q = 4$), 4 ↘ 2 bytes means that each block is divided by a quantification matrix $a_{i,j} = 1 + Q(1 + i + j)$ and coefficients are coded as short integers instead of 4 byte integers, which is equivalent to a scalar quantization. The compression ratio (the size of the compressed file as a percentage of the initial one) is obtained after a last coding step performed with the Unix *compress* command.

Transform	Filtering	Compression ratio	Distance L_1	Distance quadr	Abs (error) on peak Weak (%)	Abs (error) on peak Strong (%)
dct (16 × 16)	—	65.83	0.0	0.0	0.0	0.0
dct + ZZ(16 × 16)	—	64.83	0.0	0.0	0.0	0.0
dct (16 × 16)	4↘2 bytes	29.78	0.1633	0.4717	2.4920	0.00564
dct + ZZ(16 × 16)	4↘2 bytes	30.22	0.1633	0.4717	2.4920	0.00564
dct + ZZ(16 × 16)	quant.(Q) = 1)	25.13	0.8562	1.1593	1.2624	0.00037
dct + ZZ(16 × 16)	quant.(Q = 2)	14.83	1.4510	1.8920	2.7111	0.01014
dct + ZZ(16 × 16)	quant.(Q = 3)	10.09	1.7757	2.3168	1.2624	0.00566
dct + ZZ(16 × 16)	quant.(Q = 4)	7.64	1.9699	2.5821	3.4629	0.01798
dct + ZZ(16 × 16)	quant.(Q = 5)	6.15	2.1011	2.7731	3.8222	0.03905
dct + ZZ(16 × 16)	quant.(Q = 1), 4↘2 bytes	11.22	2.6036	3.9600	1.5694	0.01401
dct + ZZ(16 × 16)	quant.(Q = 2), 4↘2 bytes	7.75	2.8002	4.6994	2.9892	0.02582
dct + ZZ(16 × 16)	quant.(Q = 3), 4↘2 bytes	6.24	3.5451	6.8299	4.6401	0.04120
dct + ZZ(16 × 16)	quant.(Q = 4), 4↘2 bytes	5.58	2.3530	3.1612	8.8706	0.00075
dct + ZZ(16 × 16)	quant.(Q = 5), 4↘2 bytes	4.65	2.4537	3.3555	3.7922	0.01042
dct (8 × 8)	—	55.60	0.0	0.0	0.0	0.0
dct + ZZ(8 × 8)	—	55.58	0.0	0.0	0.0	0.0
dct (8 × 8)	4↘2 bytes	23.51	0.4211	0.6725	1.1548	0.00080
dct + ZZ(8 × 8)	4↘2 bytes	23.14	0.4211	0.6725	1.1548	0.00079
dct + ZZ(8 × 8)	quant.(Q = 1)	24.57	0.8760	1.1794	0.7726	0.01139
dct + ZZ(8 × 8)	quant.(Q = 2)	14.82	1.4764	1.9135	0.9637	0.00937
dct + ZZ(8 × 8)	quant.(Q = 3)	10.30	1.8022	2.3343	2.6331	0.02603
dct + ZZ(8 × 8)	quant.(Q = 4)	7.91	1.9976	2.5977	2.1682	0.02213
dct + ZZ(8 × 8)	quant.(Q = 5)	6.45	2.1311	2.7829	4.9375	0.03350
dct + ZZ(8 × 8)	quant.(Q = 1), 4↘2 bytes	10.36	2.9520	4.1561	2.8247	0.01436
dct + ZZ(8 × 8)	quant.(Q = 2), 4↘2 bytes	7.49	3.1030	4.7318	1.1168	0.03503
dct + ZZ(8 × 8)	quant.(Q = 3), 4↘2 bytes	5.39	4.1721	6.9473	3.5630	0.03335
dct + ZZ(8 × 8)	quant.(Q = 4), 4↘2 bytes	4.73	2.3846	3.1582	2.2631	0.04420
dct + ZZ(8 × 8)	quant.(Q = 5), 4↘2 bytes	3.97	2.4660	3.2835	4.0733	0.06168

Considering that the size of the compressed file gives an idea of the true amount of non-redundant information contained in the initial file, this size has been measured after removing some features of the image. This filtering has been done with mathematical morphology tools. Combination of erosions and dilatations authorizes to remove components of controlled size. With this method, either diffraction peaks, low spatial frequency components of the background or high-frequency noise have been compressed separately with *compress*. This shows that the main part of the 'information' is contained in the high-frequency background, the compressed size of which is close to that of the complete image, whereas the compressed size of peaks and low spatial frequencies of the background is negligible.

6.2. *Lossy compression*

In this section, several lossly compression algorithms are tested on the same test image described in §6.1.

Discrete cosine transform (DCT) was tested for various matrix sizes, quantizations and coding methods. Quantization is characterized by a 'Quality' number (Nelson, 1993). In this case, the quantization process consists of an integer division of the transformed matrix components by [1 + (1 + i + j) × Quality], where i and j are row and column indices. The results are listed in Table 4. In this table the ZZ means that a zigzag sorting of coefficients was performed. This is a sort of coefficients of each block in order that low-frequency coefficients occur first in a sequential order. These results show that the QDRI and compression ratio are not very sensitive to the zigzag sorting of coefficients or to the block size (8 × 8 or 16 × 16 in our examples).

Wavelet transform has been tested for various Daubechies and Coifman wavelet (Daubechies, 1992) functions, filters and codings. The results are listed in Tables 5 and 6. The deep parameter is the number of iterations (default value: 10). The wavelet transform is a periodized one-dimensional transform, applied to columns of the image (except for case '*N*row**n*', where several columns where gathered). When iterations of the transform lead to an odd number of scale coefficients, the following iteration is performed after removal of the last scale coefficient, which is considered as a detail one in further iterations. The so-called wave[int] consists of an

Table 5. *Compression tests with the wavelet transform*

wavelet(deep = 7)(order = 4) [resp. coiflet(deep = 7)(order = 4)] means that the fourth order (default: second order) Daubechies wavelet (resp. Coifman wavelet) transform is performed up to the seventh level (default: tenth level for wavelet and seventh level for coiflet). wave[int] means that an integer – interpolating – wavelet transform is performed. 3.75Log(max) is the value of the threshold when applied. sort + del:85% means that coefficients are sorted and the 85% smallest ones are suppressed. 3↘2 bytes means that coefficients are coded as short integers instead of the required number of bytes (here 3). 1 byte = 8 bits. JBIG + c means that a mask of non-zero coefficients is created and coded with JBIG, when non-zero coefficients are sorted along columns. The compression ratio (the size of the compressed file as a percentage of the initial one) is obtained after a last coding step performed with the Unix *compress* command.

Transform	Filtering	Coding	Compression ratio	Distance L_1	Distance quadr	Abs (error) on peak Weak (%)	Abs (error) on peak Strong (%)
wavelet(deep = 7)	/		48.18	0.0	0.0	0.0	0.0
wavelet(deep = 10)	/		107.2	0.0	0.0	0.0	0.0
wavelet(deep = 7)	/	sort	48.88	0.0	0.0	0.0	0.0
wavelet(deep = 10)	/	sort	106.6	0.0	0.0	0.0	0.0
wavelet(deep = 7)	3↘2 bytes	JBIG + r	35.75	0.0054	0.0738	0.0244	0.00331
wavelet(deep = 7)	0.25Log(max), 3↘2 bytes	JBIG + r	30.95	0.1833	0.4285	1.1899	0.00451
wavelet(deep = 7)	1.25Log(max), 3↘2 bytes	JBIG + r	9.669	1.7786	2.2655	2.3832	0.05566
wavelet(deep = 7)	3↘2 bytes	JBIG + c	34.62	0.0054	0.0738	0.0244	0.00331
wavelet(deep = 7)	0.25Log(max), 3↘2 bytes	JBIG + c	29.99	0.1833	0.4285	1.1899	0.00451
wavelet(deep = 7)	1.25Log(max), 3↘2 bytes	JBIG + c	9.297	1.7786	2.2655	2.3832	0.05566
wavelet(deep = 10)	27 bits↘2 bytes		34.85	3.7e−5	0.0743	0.1647	0.00330
wavelet(deep = 7)	3↘2 bytes		34.86	0.0054	0.0738	0.0244	0.00331
wavelet(deep = 7)	0.25Log(max), 3↘2 bytes		30.66	0.1833	0.4285	1.1899	0.00451
wavelet(deep = 7)	0.50Log(max), 3↘2 bytes		23.23	0.7046	0.9994	0.7422	0.00011
wavelet(deep = 7)	0.75Log(max), 3↘2 bytes		17.91	1.0727	1.4148	0.4780	0.00606
wavelet(deep = 7)	1.00Log(max), 3↘2 bytes		12.75	1.5197	1.9427	2.5144	0.01282
wavelet(deep = 7)	1.25Log(max), 3↘2 bytes		9.910	1.7786	2.2655	2.3832	0.05566
wavelet(deep = 7)	1.50Log(max), 3↘2 bytes		6.951	2.0896	2.6714	0.1417	0.03636
wavelet(deep = 7)	1.75Log(max), 3↘2 bytes		5.553	2.2520	2.8925	5.3765	0.05587
wavelet(deep = 7)	2.00Log(max), 3↘2 bytes		4.145	2.4108	3.1171	8.5171	0.07178
wavelet(deep = 7)	2.25Log(max), 3↘2 bytes		3.410	2.5125	3.2644	9.8234	0.08215
wavelet(deep = 7)	2.50Log(max), 3↘2 bytes		2.720	2.6079	3.4066	12.721	0.10048
wavelet(deep = 7)	2.75Log(max), 3↘2 bytes		2.251	2.6812	3.5128	8.5661	0.11706
wavelet(deep = 7)	3.00Log(max), 3↘2 bytes		1.983	2.7265	3.5791	4.4911	0.20795
wavelet(deep = 7)	3.25Log(max), 3↘2 bytes		1.779	2.7678	3.6419	2.3075	0.20636
wavelet(deep = 7)	3.50Log(max), 3↘2 bytes		1.636	2.7985	3.6880	0.0538	0.23228
wavelet(deep = 7)	3.75Log(max), 3↘2 bytes		1.524	2.8267	3.7302	2.0413	0.25104
wavelet(deep = 7)	4.00Log(max), 3↘2 bytes		1.435	2.8502	3.7661	5.8390	0.29458
wavelet(deep = 1)	sort + del:85%, 3↘2 bytes		17.31	1.6917	2.7095	3.7630	0.00042
wavelet(deep = 2)	sort + del:85%, 19 bits↘2 bytes		10.00	2.2696	3.9326	0.7611	0.09098
wavelet(deep = 3)	sort + del:85%, 20 bits↘2 bytes		7.690	2.2240	2.8803	8.9372	0.04392
wavelet(deep = 4)	sort + del:85%, 21 bits↘2 bytes		9.962	1.9280	2.4608	2.3773	0.06356
wavelet(deep = 5)	sort + del:85%, 22 bits↘2 bytes		9.710	1.8392	2.3450	5.2044	0.06186
wavelet(deep = 6)	sort + del:85%, 23 bits↘2 bytes		9.805	1.7987	2.2919	4.2849	0.06105
wavelet(deep = 7)	sort + del:85%, 24 bits↘2 bytes		9.805	1.7888	2.2787	2.3832	0.05790
wavelet(deep = 8)	sort + del:85%, 25 bits↘2 bytes		9.800	1.7867	2.2755	2.8706	0.06256
wavelet(deep = 9)	sort + del:85%, 26 bits↘2 bytes		9.805	1.7868	2.2757	3.0070	0.06256
wavelet(deep = 10)	sort + del:85%, 27 bits↘2 bytes		9.777	1.7868	2.2757	3.1432	0.06006
wavelet(order = 4)	sort + del:85%,↘2 bytes		9.712	1.7714	2.2550	13.846	0.00581
wavelet(order = 6)	sort + del:85%,↘2 bytes		9.972	1.7739	2.2574	3.5377	0.01550
wavelet(order = 8)	sort + del:85%,↘2 bytes		9.951	1.7698	2.2542	8.1749	0.04035
wavelet(order = 12)	sort + del:85%,↘2 bytes		10.10	1.7720	2.2579	4.7521	0.00113
wavelet(order = 16)	sort + del:85%,↘2 bytes		10.52	1.7755	2.2626	6.1388	0.03932
wavelet(order = 20)	sort + del:85%,↘2 bytes		10.76	1.7753	2.2637	1.9947	0.00956
wavelet(NROWx2)	sort + del:85%, 27 bits↘2 bytes		21.12	1.7869	2.2759	1.6287	0.04979
wavelet(NROWx6)	sort + del:85%, 27 bits↘2 bytes		29.00	1.7869	2.2759	1.7668	0.04979
wavelet(deep = 7)	morph.(1/5), 3↘2 bytes		11.80	1.7190	2.2211	1.5568	0.01116
wavelet(deep = 7)	morph.(2/6), 3↘2 bytes		10.28	1.7725	2.2611	2.3832	0.02435
wavelet(deep = 7)	morph.(1/10), 3↘2 bytes		13.97	1.6105	2.1460	5.3662	0.00515
wavelet(deep = 7)	1.25Log/$2^{m/2}$, 3↘2 bytes	JBIG + c	14.42	1.4468	1.8569	0.0798	0.01335
wavelet(deep = 7)	1.50Log/$2^{m/2}$, 3↘2 bytes	JBIG + c	11.44	1.7191	2.1962	1.8809	0.01383
wavelet(deep = 7)	1.75Log/$2^{m/2}$, 3↘2 bytes	JBIG + c	10.38	1.8213	2.3296	1.3101	0.01800
wavelet(deep = 7)	2.00Log/$2^{m/2}$, 3↘2 bytes	JBIG + c	8.589	1.9822	2.5469	1.8561	0.02870
wavelet(deep = 7)	2.00Log/2^m, 3↘2 bytes	JBIG + c	12.48	1.7677	2.2791	1.4309	0.01035
wavelet(deep = 7)	2.50Log/2^m, 3↘2 bytes	JBIG + c	11.01	1.8928	2.4527	0.1777	0.04176

Table 5 (*cont.*)

Transform	Filtering	Coding	Compression ratio	Distance L_1	Distance quadr	Abs (error) on peak Weak (%)	Abs (error) on peak Strong (%)
wavelet(deep = 7)	$Q = 0.5\text{Log}/2^{m/2}$, $3\searrow 2$ bytes	JBIG + c	20.48	0.7675	1.0127	1.7558	0.00290
wavelet(deep = 7)	$Q = 1.0\text{Log}/2^{m/2}$, $3\searrow 2$ bytes	JBIG + c	13.35	1.3481	1.7077	0.8030	0.00143
wavelet(deep = 7)	$Q = 1.5\text{Log}/2^{m/2}$, $3\searrow 2$ bytes	JBIG + c	9.003	1.7632	2.2401	3.7591	0.02239
wave[int](deep = 7)	/		30.60	0.0	0.0	0.0	0.0
wave[int](deep = 7)	/	JBIG + c	30.12	0.0	0.0	0.0	0.0
coiflet(order = 1)	$3\searrow 2$ bytes	JBIG + c	36.58	0.0052	0.0721	0.0	0.00091
coiflet(order = 1)	1.25Log(max), $3\searrow 2$ bytes	JBIG + c	9.053	1.8124	2.3063	6.9373	0.00661
coiflet(order = 2)	1.25Log(max), $3\searrow 2$ bytes	JBIG + c	8.349	1.8428	2.3453	0.1298	0.02855
coiflet(order = 3)	1.25Log(max), $3\searrow 2$ bytes	JBIG + c	8.900	1.8135	2.3092	2.2459	0.03507

Table 6. *Compression tests with the S-transform and the H-transform.*

3.75Log(max) is the value of the threshold when applied. The compression ratio (the size of the compressed file as a percentage of the initial one) is obtained after a last coding step performed with the Unix *compress* command.

Transform	Filtering	Compression ratio	Distance L_1	Distance quadr	Abs (error) on peak Weak (%)	Abs (error) on peak Strong (%)
S-tran1D(deep = 10)	/	30.56	0.0	0.0	0.0	0.0
S-tran1D(deep = 7)	/	30.58	0.0	0.0	0.0	0.0
S-tran1D(deep = 10)	1.25Log(max)	9.052	3.1307	3.9913	15.619	0.15402
S-tran1D(deep = 7)	1.25Log(max)	9.181	2.5549	3.2493	8.6029	0.00272
H-tran2D(deep = 7)	/	36.20	0.0	0.0	0.0	0.0
H-tran2D(deep = 10)	/	36.20	0.0	0.0	0.0	0.0
H-tran2D	0.3Log(max)	33.00	1.3107	1.6614	0.5526	0.00339
H-tran2D+1	0.3Log(max)	33.00	0.7915	1.0925	0.5526	0.00339
H-tran2D	0.5Log(max)	28.89	1.8141	2.2064	22.906	0.01340
H-tran2D+1	0.5Log(max)	28.89	1.2333	1.6024	22.906	0.01340
H-tran2D+2	0.5Log(max)	28.89	1.1524	1.5057	22.906	0.01340

invertible wavelet transform – the (2,2) interpolating transform, (Calderbank, Daubechies, Sweldens & Yeo, 1996) – where scale and detail coefficients are integers. Three kinds of filters were applied to columns of coefficients: logarithmic thresholding [detail coefficients smaller than $a \cdot \log(\max)$ are set to zero], decreasing logarithmic thresholding (the threshold is divided by 2^m or $2^{m/2}$, where m is the number of iterations of the wavelet transform required for the considered coefficient) and fixed ratio (short + del:n%: the n smaller detail coefficients are set to zero). Scalar quantization (conversion of floating point coefficient to '2 bytes' integers) and vectorial quantization (integer division of coefficients by thresholds described above) has been tested too. The order parameter is the number of coefficients of the Daubechies transform. For order = 2 (the Haar transform), scale and detail coefficient are, respectively, the sum and the difference of coefficients of the previous iteration. As they have the same parity, the scale parameter can be divided by 2 without loss of information. So, the initial signal, coded with 15 bits (the initial image minus the first bit-plane), leads to a transformed image coded with 16 bits. This is called S-tran1D in Table 6. The two-dimensional generalization, close to the H transform described in White *et al.* (1991), is called H-tran2D+n (with n higher bit planes removed before processing). In this transform, the scale block is divided up into blocks of 2×2 pixels $\{a_{00}, a_{10}, a_{01}, a_{11}\}$. Then, we compute,

$$b_{00} = \frac{a_{00} + a_{10} + a_{01} + a_{11}}{4}$$

$$b_{10} = a_{00} - a_{10} + a_{01} - a_{11}$$

$$b_{01} = a_{00} + a_{10} - a_{01} - a_{11}$$

$$b_{11} = a_{00} - a_{10} - a_{01} + a_{11}$$

All b_{00} are gathered in the next scale block (the scale block). Some information is lost by the 2 bits shifting ($a_{00} + a_{10} + a_{01} + a_{11}$ divided by 4). When performing the reverse transformation, this information is restored with the following relationship,

$$4a_{00} - b_{10} - b_{01} - b_{11} = a_{00} + a_{10} + a_{01} + a_{11}. \quad (8)$$

The S and H transforms fully exploit the advantages of the Haar transform with an optimized compactness. So,

Table 7. *Compression tests with the wavelet packet transform.*

packets?(deep = 7) means that a wavelet packet transform is performed up to the seventh level (default: tenth level). 1.25Log(max) is the value of the threshold when applied. sort+del:85% means that coefficients are sorted and the 85% smallest ones are suppressed. 3 ↘2 bytes means that coefficients are coded as short integers instead of the required number of bytes (here 3). The compression ratio (the size of the compressed file as a percentage of the initial one) is obtained after a last coding step performed with the Unix *compress* command.

Transform	Filtering	Compression ratio	Distance L_1	Distance quadr	Abs (error) on peak Weak (%)	Abs (error) on peak Strong (%)
packetsS(deep = 7)	3↘2 bytes	32.90	0.1095	0.3309	1.4456	0.00415
packetsS(deep = 7)	sort + del:85%, 3↘2 bytes	11.46	2.3921	9.5324	3.9811	0.00657
packetsS(deep = 7)	1.25Log(max), 3↘2 bytes	10.12	1.8472	2.3663	3.0902	0.00351
packetsI(deep = 7)	1.25Log(max), 3↘2 bytes	12.32	1.7619	2.2560	0.6193	0.00611
packetsJ(deep = 7)	1.25Log(max), 3↘2 bytes	12.52	1.7814	2.2825	6.4964	0.00579
packets1(deep = 7)	1.25Log(max), 3↘2 bytes	15.32	1.6342	2.1063	3.3478	0.01260

they are efficient lossless compression algorithms. However, they are not well adapted to filtering of detail coefficients: the parity information is lost when such a filtering is performed. Indeed, any alteration of the two right bits of b_{ij} coefficients of the H transform will produce the same error on the mean value $a_{00} + a_{10} + a_{01} + a_{11}$ when the two right bits of this term are evaluated with the relation (8) during the image reconstruction. Because of this a small alteration of any unsignifiant detail coefficient may propagate up to the larger scale coefficients. This creates a systematic bias which increases the discrepancy.

The main result of the wavelet transform based compression is that diffraction peaks are well preserved down to a compressed size of 20% (error on the integration of the weak peak is smaller than 1%). Indeed, strong diffraction peaks correspond to fast variation from pixel to pixel, and so large values of detail coefficients which are higher than threshold of the filtering. As the image difference is zero only at peak positions, it can be used as a mask for conditional filtering. This appears in Table 5 with the name 'wavelet/morph.(m/n)'. In these cases, the binary mask was obtained by a morphologic treatment (m erosions and n dilations). The result is a reduction by a factor of five for the difference in the integration of the strong peak from initial and restored image.

Wavelet packet transform was tested with one of the conditions used for the wavelet transform tests (Table 7). These tests were performed with the Haar transform. Various criteria were used for the basis optimization: (i) packetsS: minimization of the Shannon entropy; (ii) packetsI: maximization of the number of coefficients higher than a threshold equal to norm/10000; (iii) packetsJ: maximization of the number of coefficients higher than a threshold estimated from the median: at the mth iteration, the threshold is $2^m(\log N\text{row}/N\text{row})^{1/2} \times$ *median*; and (iv) packets1: split of each scale and detail coefficient.

No significant improvement was observed with respect to the wavelet transform, whatever is the criterion used, for the basis optimization. This observation can be compared with the behavior with respect to the deep parameter previously observed in the case of the wavelet transform. This proves that, for our images, the first iterations (deep ≤ 5) are the most important steps of the wavelet transform, when the mean value of close pixels is separated from the difference.

6.3. *Comparisons*

Discrepancies as a function of the compression ratio of the previous tests are shown in Fig. 9. Most of them are located along a diagonal. This means that, for efficient algorithms, the QDRI can be expressed *versus* the compression rate (R = initial size/final size) as,

$$\text{QDRI} \simeq 3.4\left(1 - \frac{3}{R}\right),$$

for a compression rate from 3 up to 20 (final size $\geq 5\%$).

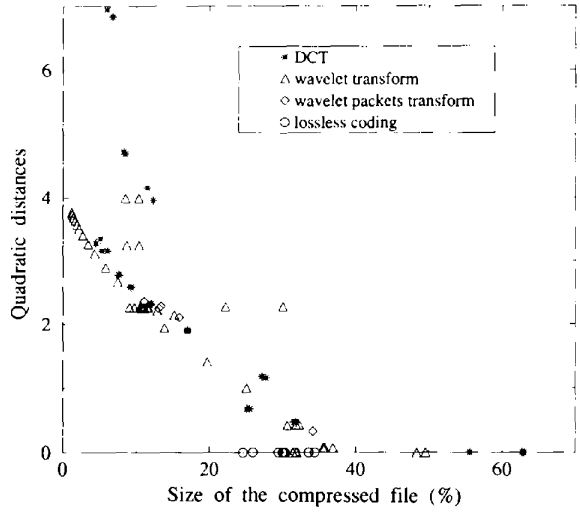

Fig. 9. Graphical comparison of compression algorithms: quadratic error (QDRI) is plotted *versus* the compressed size.

Table 8. *Comparison of DCT-compressed R (≃ 10) and wavelet-compressed (wlt, R ≃ 10 and R ≃ 20) data sets with the standard data set*

Processing is performed with *XDS* software. R_{sym} (from *XDS*), ratio of intensities (from *Xscale*) and normalized difference criteria are used.

Criteria	Standard	DCT ($R \simeq 10$)	wlt ($R \simeq 10$)	wlt ($R \simeq 20$)
R_{sym} for resolution $\geq$ 6.32 Å, (%) (number of reflections)	5.1 (1590)	5.1 (1574)	5.1 (1585)	5.1 (1487)
R_{sym} for resolution $\in$ [4.47 Å, 6.32 Å] (%) (No. of reflections)	5.1 (2979)	5.2 (2953)	5.2 (2982)	5.1 (2973)
R_{sym} for resolution $\in$ [3.65 Å, 4.47 Å] (%) (No. of reflections)	5.3 (2007)	5.3 (1968)	5.3 (2018)	5.2 (1943)
ratio of common intensities	1.000	0.993	0.997	1.001
$\sum \lvert I - I^{ref} \rvert / \sum I^{ref}$ for $3 \leq I/\sigma \leq 10$ (210 reflections) (%)	0	5.0	5.0	9.2
$\sum \lvert I - I^{ref} \rvert / \sum I^{ref}$ for $10 \leq I/\sigma \leq 20$ (470 reflections) (%)	0	1.8	1.9	3.6
$\sum \lvert I - I^{ref} \rvert / \sum I^{ref}$ for $20 \leq I/\sigma \leq 30$ (720 reflections) (%)	0	0.9	0.8	1.6
$\sum \lvert I - I^{ref} \rvert / \sum I^{ref}$ for $30 \leq I/\sigma \leq 40$ (5100 reflections) (%)	0	0.5	0.4	0.4
$\sum \left(\lvert I - I^{ref} \rvert / \sigma^{ref} \right)$ for $3 \leq I/\sigma \leq 10$	0	0.334	0.330	0.596
$\sum \left(\lvert I - I^{ref} \rvert / \sigma^{ref} \right)$ for $10 \leq I/\sigma \leq 20$	0	0.290	0.296	0.568
$\sum \left(\lvert I - I^{ref} \rvert / \sigma^{ref} \right)$ for $20 \leq I/\sigma \leq 30$	0	0.232	0.219	0.418
$\sum \left(\lvert I - I^{ref} \rvert / \sigma^{ref} \right)$ for $30 \leq I/\sigma \leq 40$	0	0.184	0.141	0.188

At a compression rate higher than 20, an asymptotic behavior is reached: the difference increases rapidly *versus* the compression rate. In this range, wavelet transform based algorithms seem to be much more efficient than Fourier transform based (DCT).

In the intermediate compression range ($3 \leq R \leq 20$) both 'wavelet(deep = 7)/*a*log(max), 3↘ 2 bytes, JBIG + *c*' and 'dct + *ZZ*(16 × 16)/*Q* = *n*, 4 bytes' have a good behavior (close to the diagonal described above). So, they have been chosen so as to compare the effect of both algorithms on classical data processing (see §5). Parameters are chosen to get a compression rate close to 10 (and also 20 for wavelet based compression), instead of ~2 with *compress* for the test data set used for this comparison. It is important to notice that the restored images, compressed with *compress*, exhibit a compression rate of ~2.4 in both cases ('DCT' and 'wavelet'), when we reach a rate of 10 if we consider the transformed image. This proves that the high compression rate of these algorithms is due to the reorganization of information rather than the filtering. The test data set used here comes from an experiment on a protein (cytochrome c_3) crystal. This crystal diffracts up to 2 Å. The lattice is cubic, with a parameter equal to ~110 Å, measured at $\lambda = 1$ Å at a distance of 420 mm between the sample and the detector. For this data set, reflections have been collected up to 3.65 Å. The following comparison based on this data set is given as an illustration of the efficiency of wavelet-based compression algorithms. However, it would be risky to extrapolate the results from only one example to other kinds of diffraction data. It is reasonable to believe that such a wavelet transform based algorithm should perform well with closer reflections, as it automatically adapts to the scale of information, saving, if necessary, more low-scale detail coefficients. The behavior of this algorithm with respect to signal-to-noise ratio (I/σ) seems to be good too, as reflections down to $I/\sigma = 3$ are well preserved. However, we cannot extrapolate to a lower signal-to-noise ratio, no more than to other situations, without further study. Such a study is beyond the scope of this paper.

With our test data set it appears that 'wavelet compressed' data are quite close but slightly better than DCT-compressed data considering the number of measured reflections. This remains true if we consider the R_{sym} criterion for low and high intensities. At the intermediate intensities, DCT-compressed data are better. In any case, the R_{sym} is well preserved and the difference is small compared with the standard deviation. These results are listed in Table 8. The statistical comparison of the result of this data processing, in terms of relative intensity difference, shows that reflections with low signal-to-noise ratio may be slightly modified. This can be explained: intensity are rescaled during this data processing with an evaluation of the gain of the detector. This gain is computed from the noise of the background. Considering that the statistic should be Poissonian, the gain is expressed as the ratio of the variance on the expectancy of this noise,

$$\text{gain} \simeq \frac{\langle a_{ij}^2 \rangle - \langle a_{ij} \rangle^2}{\langle a_{ij} \rangle}.$$

Although the reflection itself is very well preserved, the background is smoothed by the compression. The estimation of the gain by the first step of the processing with *XDS*, 1.68 for the initial data, is then reduced to 1.01 for 'wavelet compressed ($R = 10$)' data and 0.76 for 'DCT-compressed' data. However, if the first step of the data processing – the indexation – is performed with the initial data set, the correct value of the gain is preserved and the result of the integration step of the restored data is improved. These results are illustrated in Fig. 10.

The last comparison we can perform is a visual comparison. This is illustrated by Figs. 11 and 12 (to be compared with Fig. 13). Although it is not very relevant in our case, it exhibits kinds of alterations which are generated by both transforms.

7. Conclusions

Lossless and lossy compression of diffraction patterns has been considered. Compression of such images is a challenging but difficult problem, as these images exhibit a large uncorrelated noise and are used for numerical evaluation. However, the results already obtained are welcome in the competition between the development of X-ray sources and the increase in the capacity of bidimensional detectors on one hand, and the improvements on hard drives and network bandwidth on the other.

In the case of the lossless compression (coding), a limit appears for each class of algorithm, no matter which software is used. For our test image, we obtained a compression rate of 2.2 for Huffman coding, 3.0 for LZ77, LZW and arithmetic coding, 3.7 for BWT and 4.1 for the so-called finite context modeling. Although the latter is the most efficient, it is limited by the CPU time required (32.3 s instead of 4.0 s for the Unix *compress*

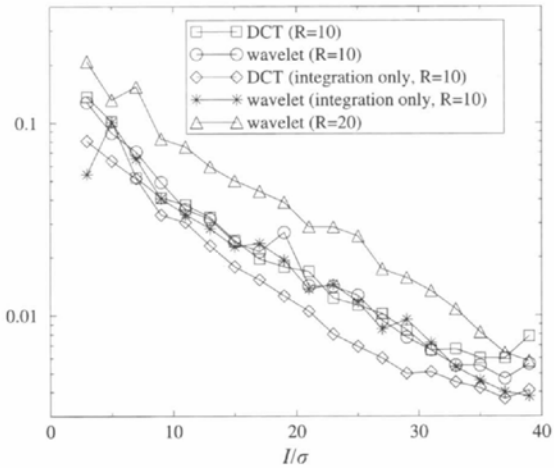

Fig. 10. Comparison of DCT compressed (compression rate $R \simeq 10$) and wavelet compressed ($R \simeq 10$ and $R \simeq 20$) with a standard data set of restored images. Data processing is performed with *XDS*, and a mixed processing where only the integration step is made with restored images. The relative difference $(1/N_i \sum_{N_i} |I_j - I_j^{\text{ref}}|^2)^{1/2} / (1/N_i \sum_{N_i} I_j)$ is plotted *versus* the intensity normalized to its standard deviation. N_i is the number of reflections with a signal-to-noise ratio between $i-1$ and $i+1$.

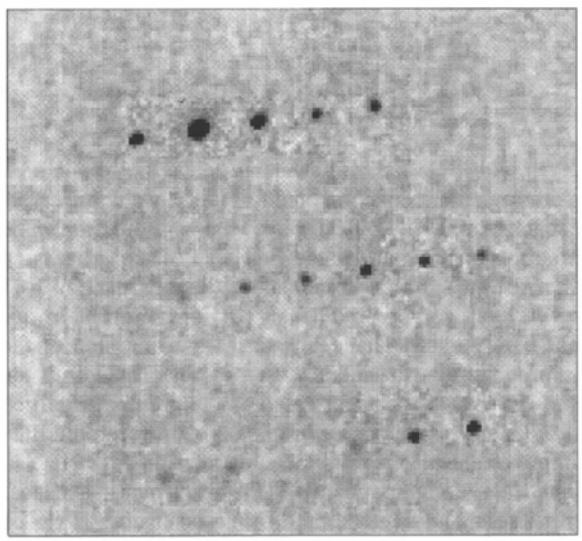

Fig. 11. Enlargement of an area of the test image compressed with the 'dct + $ZZ(16 \times 16)/Q$ = 3/4 bytes'. We can observe an alteration at the foot of diffraction peaks.

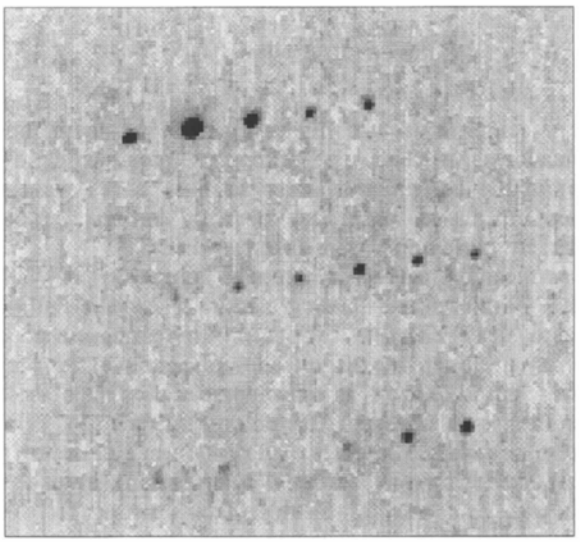

Fig. 12. Enlargement of an area of the test image compressed with the 'wavelet(deep = 7)/1.25log(max), 3↘ 2 bytes, JBIG + c'. Alteration of the background, due to the filtering of detail coefficients at large scale, is homogeneously spread.

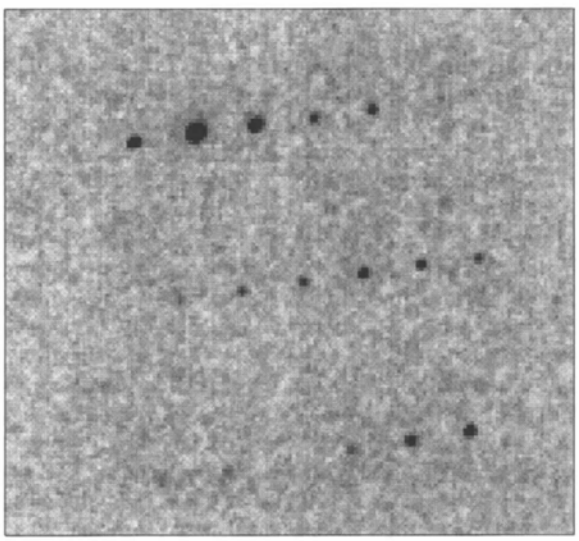

Fig. 13. Enlargement of an area of the initial test image.

command on our HP730 station). Use of a faster computer should reduce this limitation.

In the case of lossy compression, the problem is quite different. DCT as well as wavelet-based algorithms lead to a compression rate of up to 10 without any significant visual degradation of the image. Quantitative alteration, evaluated by numerical criteria from data-processing software, appears to be very weak. For these criteria, the wavelet transform seems slightly better than the DCT. Brute comparison of the results of this data processing shows that intensity differences remain low, compared with the standard deviation estimated by the integration software. However, it appears that reflections with a low signal-to-noise ratio are slightly modified. This can be explained: some statistical parameters are computed from the noise of the background and are subsequently used in the integration step. Although the reflection itself is very well preserved, this background is smoothed by the compression. This can be reduced in the future by a modified version of the data-processing software or by reintroducing the noise on the restored images. In any case, this does not alter significantly the result of the data processing. So, a strategy may consist of processing the data with compressed images up to the end. Once the parameters of this processing are established, a final run can be performed on initial images so as to get the most accurate result, which should be very close to the previous one. Such a strategy will save a lot of shared disk space during most of the processing time and long-time archiving on disk becomes possible. Moreover, network transfer and reconstruction of compressed diffraction images is, most of the time, much faster than transfer of uncompressed images. This is of great interest during data collection, when one wants to send images on a remote computer in order to process these data with a specific software, or to take advantage on a fast CPU.

In future, initial images should become useless once it has been proved that the electron-density map obtained with compressed images has the same quality.

We are grateful to the referees for their suggestions. We also thank Pierre Legrand, Ulrike Willms and Andy Hammersley for theirs comments and Jay Bertran for his critical reading of the manuscript.

References

Abrahams, J. P. (1993). *Jnt CCP4 ESF-EACBM Newslett. Protein Crystallogr.* **28**, 3–4.

Algazi, V. R., Kato, Y., Miyahara, M. & Kotani, K. (1992). *Proceedings of SPIE, Applications of Digital Image Processing XVI*, **1771**, 396–405.

Bellard, F. (1997). Unpublished work.

Bertran, J. A., Pignol, D., Bernard, J.-P., Verdier, J.-M., Dargorn, J.-C. & Fontecilla-Camps, J. C. (1996). *EMBO J.* **15**, 2678–2684.

Bourgeois, D., Moy, J.-P., Svensson, S. O. & Kvick, A. (1994). *J. Appl. Cryst.* **27**, 868–877.

Calderbank, A. R., Daubechies, I., Sweldens, W. & Yeo, B.-L. (1996). Wavelet Transforms that Map Integers to Integers. Department of Mathematics, Princeton University, USA. http://cm.bell-labs.com/who/wim/papers/papers.html #integer.

Coifman, R., Meyer, Y. & Quake, S. (1989). In *Proceedings of the Conference on Wavelets*, Marseille, France.

Daubechies, I. (1992). CBMS-NSF Lecture Notes No. 61, Soc. Indust. Appl. Math., Philadelphia, USA.

Davis, G. (1994). PhD thesis, New York University, USA.

European Synchrotron Radiation Facility (1994). *ESRF Beamline Handbook.* Grenoble: ESRF.

Fanchon, E., Ferrer, J.-L., Kahn, R., Berthet, C. & Roth, M. (1995). *ESRF Newslett.* **24**, 6–7.

Gabor, D. (1946). *J. Instr. Electr. Eng. (London).* III, **93**, 429–457.

Hammersley, A. P. (1995). *Fit2d* v5.18, *Reference manual* v1.6. Internal Report EXP/AH/95-01, ESRF, France.

Hilton, M. L., Jawerth, B. D. & Sengupta, A. (1994). *Multimedia Systems* **2**, 218–227.

Hirvola, H. (1997). Ha 0.999 archiver (beta version). http://www.rarf.riken.go.jp/archives/Linux/utils/compress/ha0999p-linux.lsm.

Huffman, D. A. (1952). *Proc IRE*, **40**, 1098–1101.

ISO (1991). *Progressive bi-level image compression, revision* 4.1, ISO/IEC JTC1/SC2/WG9, CD 11544.

Kabsch, W. (1988*a*). *J. Appl. Cryst.* **21**, 67–71.

Kabsch, W. (1988*b*). *J. Appl. Cryst.* **21**, 916–924.

Kabsch, W. (1993). *J. Appl. Cryst.* **26**, 795–800.

Mallat, S. & Zhang, Z. (1993). *Tech. Rep. 611 IMAG*, pp. 1–42.

Meyer, Y. (1992). *Les Ondelettes.* France: Armand Colin.

Moy, J.-P. (1994). *Nucl. Instrum. Methods Phys. Res. A*, **348**, 641–644.

Nelson, M. (1993). *La compression de donnés.* France: Dunod.

Nelson, M. (1996). *Dr Dobb's J.* http://web2.airmail.net/markn/articles/bwt/bwt.htm.

Pennebaker, W. B. & Mitchell, J. L. (1992). *JPEG Still Image Data Compression Standards.* New York: Van Nostrand Reinhold.

Pennebaker, W. B., Mitchell, J. L., Langdon, G. G. & Arps, R. B. (1988). *IBM J. Res. Develop*, **32**, 771–726.

Rao, K. R. & Yip, P. (1990). San Diego: Academic Press.

Roth, M., Ferrer, J.-L., Simon, J.-P. & Geissler, E. (1992). *Rev. Sci. Instrum.* **63**, 1043–1046.

Simon, J.-P., Geissler, E., Hecht, A. M., Bley, F., Livet, F., Roth, M., Ferrer, J.-L., Fanchon, E., Cohen-Addad, C. & Thierry, J.-C. (1992). *Rev. Sci. Instrum.* **63**, 1051–1054.

Watson, A. B., Yang, G. Y., Solomon, J. A. & Villasenor, J. (1996). *Proc. SPIE Hum. Vis. Electron. Imaging B*, **2657**, 382–392.

Welch, T. (1984). *IEEE Comput.* **17**, 8–19.

White, R. L., Postman, M. & Lattanzi, M. (1992). *Compression of the Guide Star Digitized Schmidt Plates, Digitized Optical Sky Surveys*, edited by H. T. MacGillivray & E. B. Thomson, pp. 167–175. Dordrecht: Kluwer Academic Publishers.

Wickerhauser, M. V. (1992). *Digital Signal Process.* **2**, 204–226. http://wuarchive.wustl.edu/doc/techreports/wustl.edu/math/.

Wu, X. & Memon, N. (1995). *NIU Computer Science Homepage.* http://www.cs.niu.edu/.

Ziv, J. & Lempel, A. (1977). *IEEE Trans. Inf. Theory*, **23**, 337–343.

Ziv, J. & Lempel, A. (1978). *IEEE Trans. Inf. Theory*, **24**, 530–536.