LA-UR- 99-4298

Approved for public release; distribution is unlimited.

Title: A NEW APPROACH FOR FINGERPRINT IMAGE COMPRESSION

Author(s): Mazieres, Bertrand

Submitted to: Informal Distribution

Los Alamos
NATIONAL LABORATORY

LA-UR- 98-4298

A New Approach for Fingerprint Image Compression

Los Alamos National Laboratory, NM
Bertrand MAZIERES
Jul-Dec 1997

Contents

Résumé

Dans le cadre de l'archivage de sa base d'empreintes digitales, le FBI a spécifié un standard en 1993. Cette publication concerne un décodeur qui s'attend à traiter un fichier dont le format est inspiré largement de JPEG. L'algorithme, quant à lui, repose sur une transformée en ondelettes suivie d'une quantification uniforme scalaire des sous-bandes et de leur codage entropique précédé d'un run-length coding.

Après une brève évocation de ce que faisait le premier codeur, nous étudions ici une nouvelle méthode pour réaliser l'allocation de bits, connue sous le nom de quantificateur scalaire à contrainte entropique.
Nous abordons ensuite certains aspects du codage entropique et évoquons quels paramètres sont importants dans cette phase de codage sans pertes.
Enfin nous verrons quels resultats sont obtenus avec ce nouveau codeur tel qu'implémenté fin décembre 97, c'est-à-dire sans reglages particuliers.

Introduction

The FBI has been collecting fingerprint cards since 1924 and now has over 200 million of them. Digitized with 8 bits of grayscale resolution at 500 dots per inch, it means 2000 terabytes of information. Also, without any compression, transmitting a 10 Mb card over a 9600 baud connection will need 3 hours.

Hence we need a compression and a compression as close to lossless as possible : all fingerprint details must be kept. A lossless compression usually do not give a better compression ratio than 2:1, which is not sufficient.
Compressing these images with the JPEG standard leads to artefacts which appear even at low compression rates.

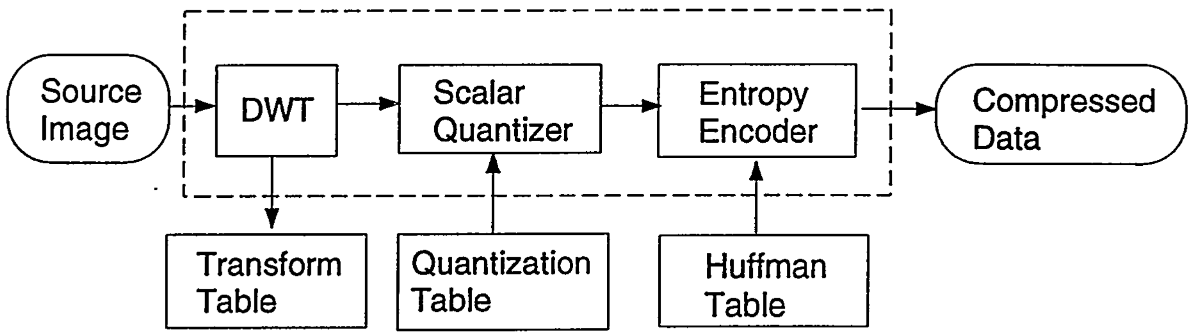
Therefore the FBI has chosen in 1993 a scheme of compression based on a wavelet transform, followed by a scalar quantization and an entropy coding : the so-called WSQ. This scheme allows to achieve compression ratios of 20:1 without any perceptible loss of quality.

The publication of the FBI ([1]) specifies a decoder, which means that many parameters can be changed in the encoding process : the type of analysis/reconstruction filters, the way the bit allocation is made, the number of Huffman tables used for the entropy coding, ...

The first encoder ([4]) used 9/7 filters for the wavelet transform and did the bit allocation using a high-rate bit assumption. Since the transform is made into 64 subbands, quite a lot of bands receive only a few bits even at an archival quality compression rate of 0.75 bit/pixel.
Thus, after a brief overview of the standard, we will discuss a new approach for the bit-allocation that seems to make more sense where theory is concerned.
Then we will talk about some implementation aspects, particularly for the new entropy coder and the features that allow other applications than fingerprint image compression.
Finally, we will compare the performances of the new encoder to those of the first encoder.

Chapter 1

The Wavelet-Scalar Quantizer



As we have already mentioned, the scheme of compression is decomposed in three steps.
The input is usually an image digitized with 8 bits. The size is not specified and it does not have to be a square image.
The first step consists in a wavelet transform performed by a multirate filter bank of depth 5. The result is a decomposition in 64 subbands representing the 2D-spectrum of the image.

The coefficients of the DWT are then quantized to integer values. This is where the lossy compression is performed.
Each subband is allocated a bin width : the quantization is scalar uniform. Except for the zero bin, all bins of a subband have the same size.

Finally, subbands are gathered into 3 to 8 blocks, huffman tables are computed for these blocks and each block is entropy coded using one of these tables and the alphabet specified by the FBI.
Let us also mention that this alphabet includes symbols for run-length encoding, which means that the lossless compression obatined by the entropy coding can be huge, particularly for higher subbands where most of the coefficients of the DWT are quantized to a zero value.

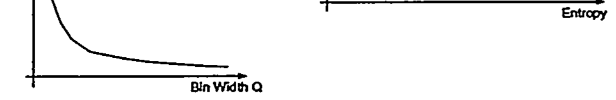
The structure of the compressed data is much inspired from the JPEG standard. It contains a header for the transform table, with the filter coefficients, a quantization table with the bin widths for all subbands and the different Huffman tables.

Chapter 2

Entropy-Constrained Scalar Quantizer

2.1 Introduction

We want to improve the quantization in the present scheme of compression. This new method will be based on the observation that quantization and entropy coding are strongly correlated : The lossy compression obtained from the quantization must be adapted to the huffman coding that follows. The next figure represents our assumption concerning the behaviour of the distortion and the entropy with the scaling factor Q.
The factor that minimizes the distortion can be inappropriate for entropy.



The optimization problem, which consists in minimizing the distortion due to the quantization will be derived with a new constraint on the entropy. We want, for each subband, to go as far as possible towards the lower entropys in the flat region of the rate-distortion curve. In other words, we try to gain as much as possible on the entropy without losing much quality. We can write the optimization problem as the following system of equations :

$$Min\ D = \sum_{i=0}^{i=63} w_i \frac{D_i(Qi)}{4^{d_i}} \tag{2.1}$$

$$H = \sum_{i=0}^{i=63} \frac{H_i(Qi)}{4^{d_i}} \tag{2.2}$$

where w_i is a weight for the subband i and d_i the depth of the subband.

2.2 Data Modeling

The first step is the modeling of the subband data. We assume that the different subband pdf can be modelized by a generalized gaussian distribution (GG), that is to say :

$$\mu_x(.) = C_1 * e^{-C_2 * |x|^{\nu}} \tag{2.3}$$

with

$\nu = \frac{1}{\sigma} * \sqrt{\Gamma(\frac{3}{\alpha})/\Gamma(\frac{1}{\alpha})}$

$C_1 = \frac{\alpha * \nu}{2 * \Gamma(1/\alpha)}$

$C_2 = e^{\alpha * log(\nu)}$

To confirm this assumption, we do a χ^2 test. Plotting χ^2 against ν, we can also determine what parameter for the GGD gives the best fit.

2.2.1 The χ^2 Test

We would have liked to generate random variables with a generalized gaussian pdf so that we could have mixed them with the data from the subbands of fingerprint images. Nevertheless, solving this problem basically means being able to integrate $e^{-C_2 * |x|^{\nu}}$ for any value of $\nu \in]0;2]$, which seems to be inextricable.

Hence we will do the test only on the basis of real data from images.

Pdf estimate

There are two options for the pdf estimate :

- Using a normal histogram where all bins have the same width.
- Using a modified histogram where all bins have the same area.

The modified histogram seems to have several advantages : First, it really draws an histogram whose total area is 1, which is closer to the usual idea of the pdf. Secondly, it takes more significant values in account. As a matter of fact, in the normal histogram, with a gaussian distribution, about 50 percents of the bins have a height smaller than 5 percent of the maximum height. On the contrary, the modified histogram has almost all its bin heights bigger than 5 percent of the maximum height, which means that the values estimated are more significant.

Then, we must choose an accurate number of bins for the Chi-square test. Some tests on a Laplacian distribution (GGD with $\nu = 1$), easy to generate, show that choosing between 10 and 30 bins can be adapted to our needs. Actually, it is not easy to determine exactly the best number since we always have a large confidence interval and the values of χ^2 do not always appear in the tables (see [3], page 537, for instance).

ν estimate

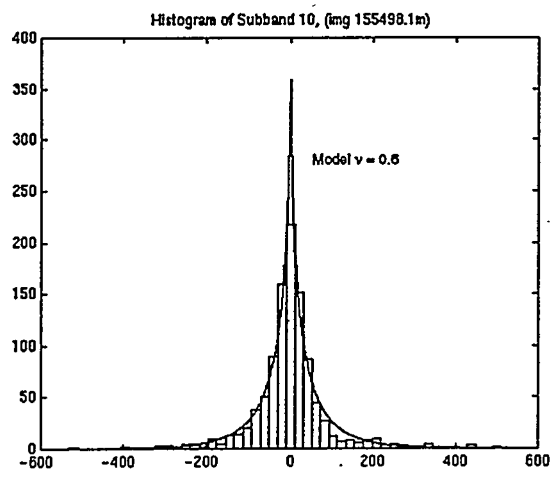
To estimate ν, we draw the function $\chi^2 = f(\nu)$ and determine where the minimum of f is reached.
To compute χ^2, we use the formula :

$$\chi^2 = \sum_{i=1}^{i=K} \frac{(P_{X_{actual}}(i) - P_{X_{model}}(i))^2}{P_{X_{model}}(i)} \tag{2.4}$$

where K is the number of bins.

Doing the Chi-square test on laplacian distributed data shows that the ν estimate made with the modified histogram is more accurate than those of the normal histogram.

2.2.2 Results



We conducted χ^2 tests on different subbands of cropped 512x512 images (only the center of the image that contains the fingerprint is kept : background blank spaces and text have disappeared).

The conclusion is that GG distributions fit the data very well, with more then 99.99% of certitude that the distribution of the model is the same as the distribution of real data.
The parameter ν, depending on the subband and the image, can be anywhere between 0.5 and 1.

Also, experimentations on matched cards reveal that this parameter ν does not seem to depend on the type of images : scans of paper cards or live scans. The difference in ν parameters between a live scan and its matched paper card was always within the range of incertitude.

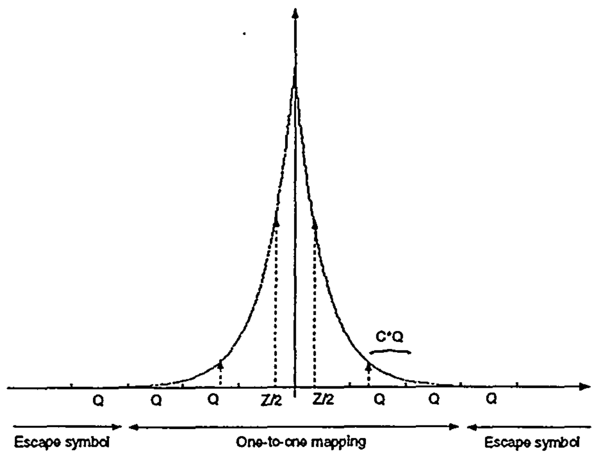
Chi-square tests have been conducted both with equal-bin-width histograms and equal-bin-population histograms. The incertitude on ν was around ± 0.03. It is due to the fact that we do not have many samples per bin in the histogram.
We also used the method of the relative entropy to estimate ν. It consists in finding the minimum of $D(p\|q) = \sum p(x) \log \frac{p(x)}{q(x)}$, D measuring the inefficiency of assuming that the distribution is q when the true distribution is p.

The results were comparable although the number of samples per bin seemed to play too much of an important role to rely on this method.

2.3 Distortion, Entropy calculation

We now derive the expressions of distortion and entropy for a subband i.

2.3.1 Models



We use two different models for distortion and entropy.
The model for distortion assumes that we have an infinite number of bins of width Q, except for the zero bin which has a width Z. The reconstruction bin center is given by C.
On the contrary, Entropy calculation assumes that there is a finite number of quantized values and that they can be coded with a certain number of bits. Then we make the difference between values that will be mapped one-to-one to symbols (for these, there is a bijection between the quantized values and the symbols) and those that will be coded as escaped symbols followed by the value as an 8 or 16 bit integer.

2.3.2 Distortion

For a subband i,

$$D_i = \sum_j \int_{X_j - a_j}^{Xj+b_j} |x - X_j|^2 \mu_X(x) dx \tag{2.5}$$

where $\{X_j\}$ are the quantized values and $\{a_j + b_j\}$ the bin widths.
We want D_i to be a function of Q_i, the scaling factor of the subband i.

preliminaries

Deriving the expression of D_i requires being capable of integrating $x^n e^{-C_1 * |x|^\nu}$, with n integer. We can do it in the special case where the inverse of ν is an integer.

$$F_n(x) = C_1 \int_0^x u^n e^{-C_2|u|^\nu} du \tag{2.6}$$

Thanks to the change of variable $t = C_2 u^\nu$, we obtain :

$$F_n(x) = \frac{C_1}{\nu C_2^{\frac{n+1}{\nu}}}(m! - b_{n,\nu}(x)) \tag{2.7}$$

with

$$b_{n,\nu}(x) = exp(-C_2 x^\nu) \sum_{k=0}^{k=m} \frac{m!}{(m-k)!}(C_2 x^\nu)^{(m-k)} \tag{2.8}$$

and $m = \frac{n+1}{\nu} - 1$

Expression of the distortion

Let us rewrite formula (2.5) using the symmetry of the distribution :

$$D_i = D_i^0 + 2\sum_{j\geq 1} \int_{\beta_{j-1}}^{\beta_j} |x - X_j|^2 \mu_X(x)dx \tag{2.9}$$

with $\beta_i^j = jQ_i + Z_i/2$ and $X_i^j = \beta_i^j - CQ_i$, where $C \in [0;1]$

$$D_i = D_i^0 + 2\sum_{j\geq 1} D_i^j \tag{2.10}$$

For the zero bin,

$$D_i^0 = 2\int_0^{Z_i/2} x^2 \mu_X(x)dx \tag{2.11}$$

$$D_i^0 = 2(F_2(Z_i/2) - F_2(0)) \tag{2.12}$$

For the bin j, $1 \leq j \leq \infty$

$$D_i^j = (F_2(\beta_i^j) - F_2(\beta_i^{j-1})) - 2(\beta_i^j - CQ_i)(F_1(\beta_i^j) - F_1(\beta_i^{j-1}) + (\beta_i^j - CQ_i)^2(F_0(\beta_i^j) - F_0(\beta_i^{j-1})) \tag{2.13}$$

Finally, after simplification,

$$D_i^j = \sigma^2 + 2\sum_{j\geq 1}(\beta_i^j - CQ_i)[(\beta_i^j - CQ_i)[F_0(\beta_i^j) - F_0(\beta_i^{j-1})] - 2[F_1(\beta_i^j) - F_1(\beta_i^{j-1})]] \tag{2.14}$$

For the calculation of the distortion, we have supposed that we have an infinte number of bins. We will see later that this assumption makes sense since we can code our values with 16 bits. Nevertheless, if the number of bins was finite and small, we would have to add another term referred to as overload distortion.

2.3.3 Entropy

Since the entropy is always a lower bound, the real compression rate is not as good as the entropy. Nonetheless, the use of run length coding makes it possible to be closer to this lower bound. So we will suppose that the entropy is a good estimate for the final rate :

$$Rate_{(Runlength+Huffman)} \simeq Entropy + \sum_{S_j \in Esc} NbBits(S_j) * P_S(S_j) \tag{2.15}$$

where S_j is the set of symbols we can use. From now on, we will call this set an alphabet. We use the same alphabet for all subbands.
The computation of the entropy requires the knowledge of the probability of every symbol.
For the subband i,

$$H_i = H_i^0 - \sum_j P_S(S_j) \log_2(P_S(S_j)) + \sum_{S_j \in Esc} NbBits(S_j) * P_S(S_j) \tag{2.16}$$

$$zero\ bin, \quad H_i^0 = -2P_X(Z_i/2) * \log_2(2P_X(Z_i/2)) \tag{2.17}$$

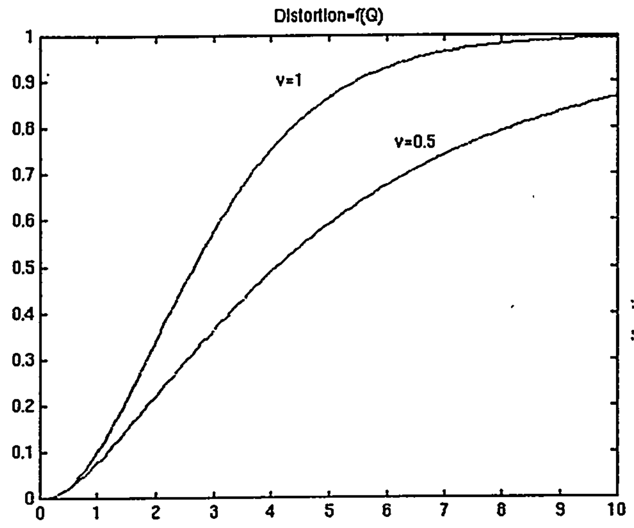
$$Xi\ non\ escape\ symbol, \quad P_S(S_j) = \int_{(j-1)Q_i+Z_i/2}^{jQ_i+Z_i/2} \mu_X(x)dx \tag{2.18}$$

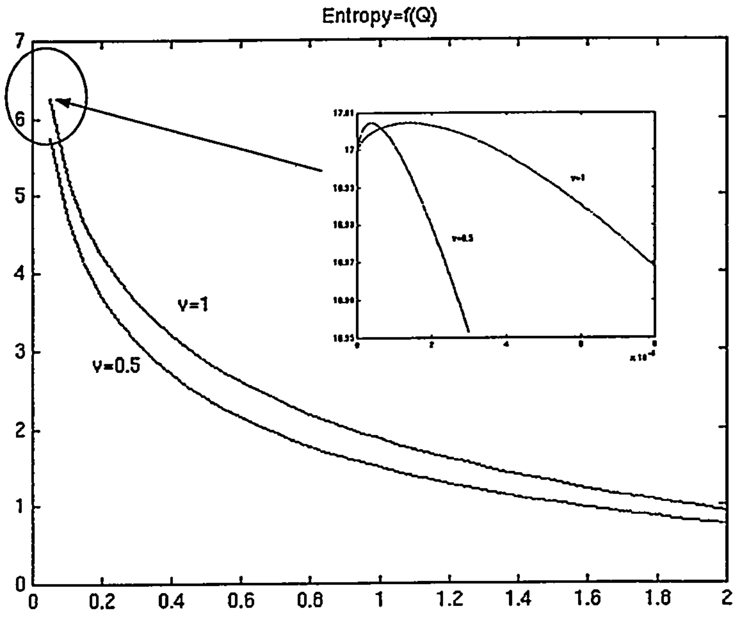
$$Xi\ escape\ symbol, \quad P_S(S_j) = \int_{MinVal_j}^{MaxVal_j} \mu_X(x)dx \tag{2.19}$$

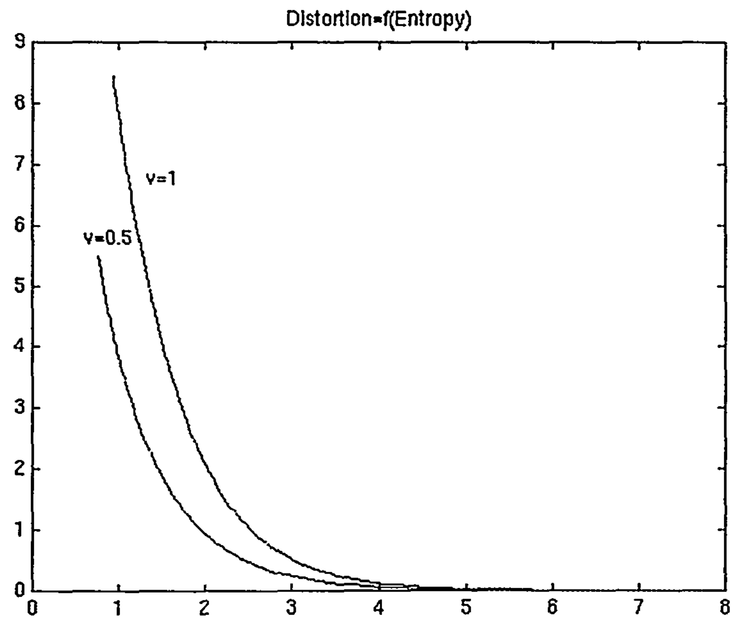
The last $MaxVal_j$ is actually considered as being ∞.

2.4 Analysis of the models

For two distributions $\nu = 0.5,\ \sigma = 1$ and $\nu = 1,\ \sigma = 1$, we obtain the following graphs of Distortion and Entropy.
The graph Distortion=f(Entropy) corresponds to $\sigma = 5$.


Distortion=f(Q)
v=1
v=0.5


Entropy=f(Q)
v=1
v=0.5



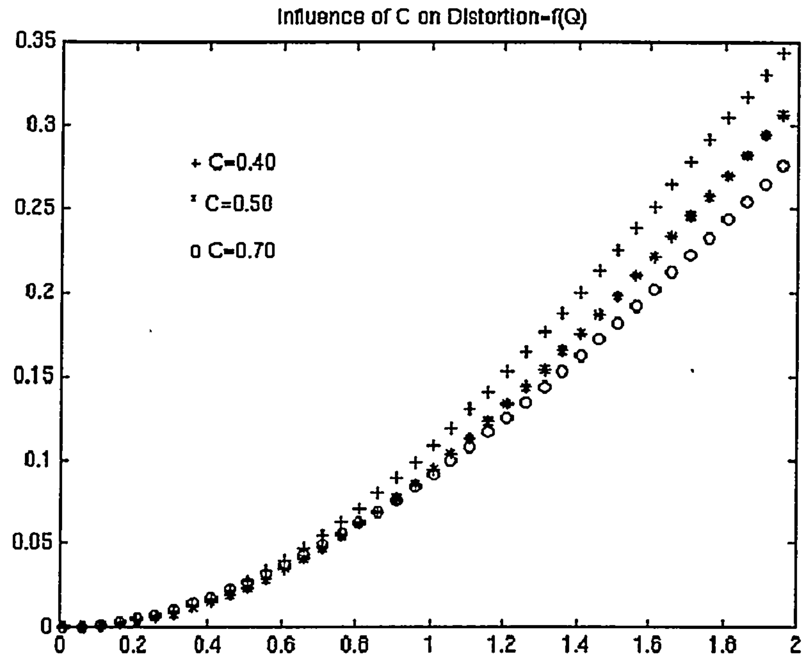
2.4.1 Behaviour around 0 and ∞

When Q = 0, since we assume that we can have as many symbols as possible for the distortion calculation, the distortion is zero.
Where the entropy is concerned, we have only two escape symbols whose probabilities are 0.5 and length is 16 bits. So the entropy is 17.

When $Q \to \infty$, we have only the value 0. So the distortion is 0, supposing an infinity of symbols (and σ^2 if the number of symbols is finite) and the entropy 0.

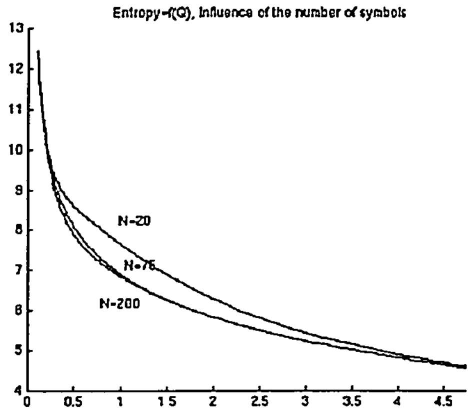
2.4.2 Influence of C

Generating graphs for values C=0.4, C=0.5 and C=0.7, we can see an interesting behaviour of the distortion with C. Whereas the encoder number one used a value of C = 0.44, we realize here - assuming that our model is accurate- that this is not an optimal value. When Q is large, values of C between 0.5 and 0.7 can give a better result. We can explain it by the fact that the best reconstructed value would be the centroid of the bin. Since (for positive values), the slope of the distribution is negative, it has to be on the left side of the bin center, which means that we should have $C > 0.5$. The value of C has no influence on the entropy.



2.4.3 Influence of the number of symbols in bijection

The number of symbols mapped one-to-one does not have any influence on the distortion. On the contrary, we want to find the alphabet that minimizes the entropy.
For $\sigma = 30$, we obatin the following curve :

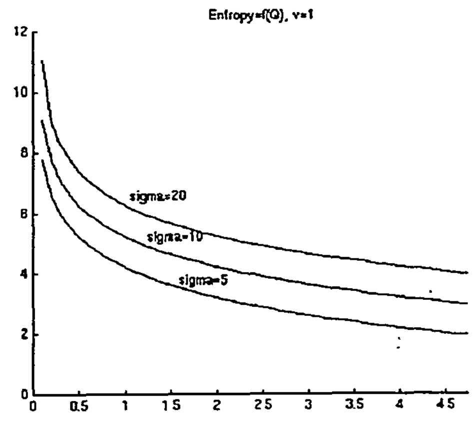


We see that after a certain number N_0 of symbols, there is almost no difference. So the present value N=75 seems to be a good choice since a greater number of symbols also means a bigger huffman table.

When there are almost no escape symbols ($N \geq 1000$) , there is a significant gain for the small values of Q. But this is an area where we should not be because the entropy is always large (12 bits/pixel, typically).

2.4.4 Influence of the variance

The greater the variance, the greater the entropy. But the shape of the graph remains the same. The distortion grows a little bit faster with a smaller σ but there is not much change.

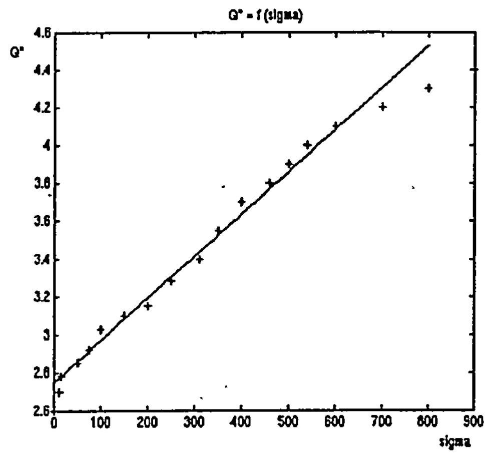


The most important thing is to be able to find the breakpoint on the graph Distortion=f(Entropy). After this point, the distortion begins to grow quickly and the entropy is not much improved when Q increases.
We want to have access to the breakpoint Q* in function of σ.

2.4.5 Estimate of Q* in function of σ

We need a criterion to decide how to choose the breakpoint Q* for one subband. Also, we would like to make it depend on the variance of the subband.
The idea here is to decide not to go farther than the point on the rate-distortion curve where the derivative gets to small. Because it means that we are losing much coding gain without improving much the quality.
We decide that we will choose this breakpoint by finding the value of Q for which $\frac{\nabla D}{\nabla H} = 1$ since it seems to be where the derivative takes a turn from a steep region to a flat region.

And we plot Q* versus the standard deviation, finding the breakpoint with the Laplacian distribution :

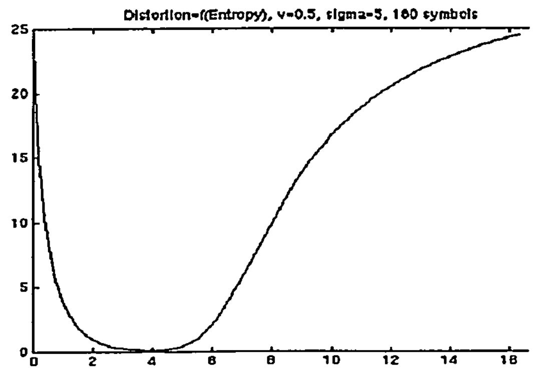


We realize we can assume that the relation between Q* and σ is linear.
It is not so true when σ is too small or too large. But when σ is close to zero, we will not allocate the maximum of bits but rather the minimum of bits for the subband. Also, this criterion was subjective and trying to find a better fit for this curve would probably not make sense.

The model is then :

$$Q^* = 0.002\sigma + 2.8 \tag{2.20}$$

We should also mention that before choosing this criterion, we had another criterion consisting in taking the minimum of an overloaded distortion :



But the problem of this criterion was that the minimum depended too much on the number of symbols used to reveal this minimum.
It then gave a perfectly linear relation

$$Q^* = 0.065\sigma \tag{2.21}$$

Which was somewhat suspicious. We will see later that this was not a good choice.

2.5 Accuracy of Estimates

We now compare the distortion and the entropy of real data subband by subband to the estimates based on the generalized gaussian distribution modeling.
The following curves -if not specified- are obtained for 512x512 cropped images, which means that most of the image is occupied by the fingerprint.

2.5.1 Variance Computation

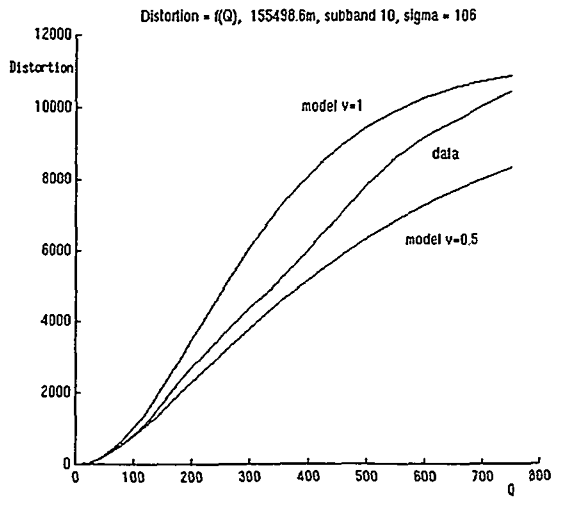
The computation of the variance of every subband is decisive for the estimates since we will fix the parameter ν to 0.5 or 1 and then use the standard deviation to model the distribution of a subband.

To compute the variance, we do as encoder 1 did (see [4]), which is simply taking a subregion of the image where the fingerprint is usually located and then using unbiased estimates of the mean of the subregion and the variance of the subregion.

For cropped images, the variance of the subregion is basically the same as the variance of the whole subband since both of them contain only fingerprint information.

On the contrary, taking a subregion on 768x768 images allows to get rid of most of the blank spaces and the text.
We were nevertheless surprised to see that variance of the subregions were smaller than variance of the whole regions since we simply removed white spaces (and text as well). Yet it seems to be better to compute variance over a subregion since the model is then adapted to the fingerprint and not to the whole image. It is more important to have good quality on the print than on the background.

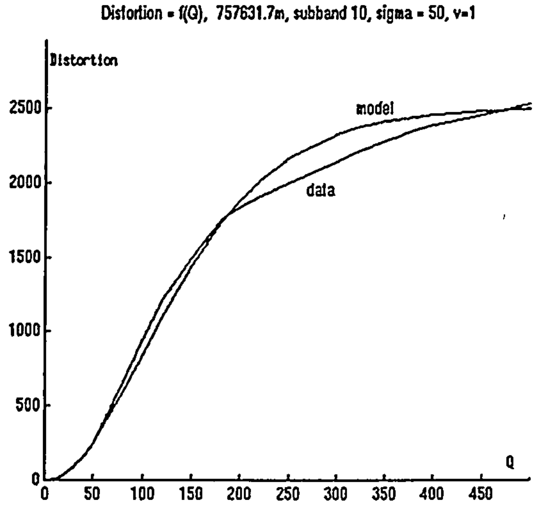
2.5.2 Distortion

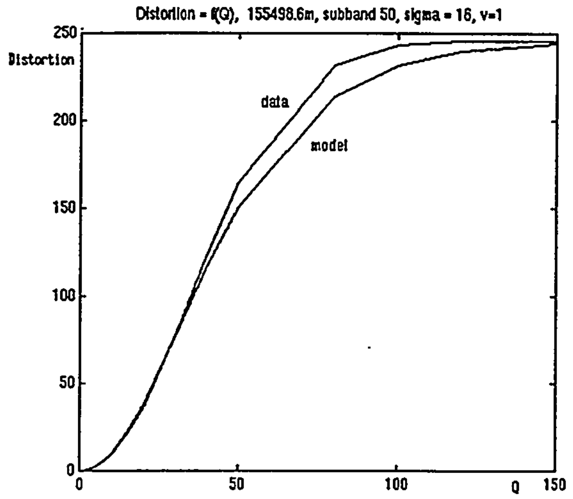


If we look at this graph, we can see that for this subband of this image, the distortion curve of the real data is really between the distortion curves obtained for a laplacian model and a GGD($\nu = 0.5$) model.

Laplacian Model

Since we can hardly do the chi-square test and determine the ν parameter for each subband, we would like to choose one value rather than another for ν: either 0.5 or 1 for simplicity of calculation.
In most cases, we observe that the curve χ^2 versus ν is not so steep after the minimum. Hence we decide to use the Laplacian model ($\nu = 1$) and look more precisely at the distortion curves.
We will see later if a Laplacian is good enough for entropy estimates.

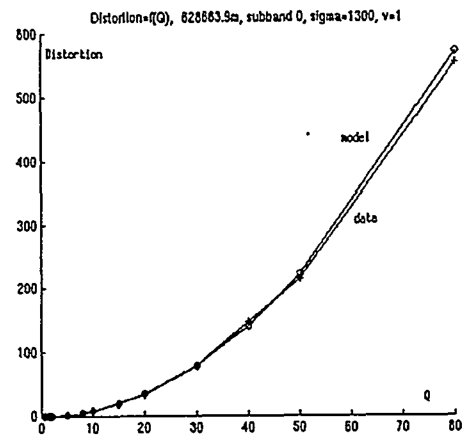




These two graphs represent the fit of the Distortion estimate based on a Laplacian model for a subband 10 and a subband 50 of two different images.
We can see that the fit is very good in the region of the lower Qs, which is the most important since we want to stay at high target bit rates for this application (typically 0.75 bits/pixel).
Other experiments have shown that the fit was not always perfect even in this region but we are always within 5% of difference in this region on the left of the inflexion point.
Hence, we will then assume that Distortion estimates computed with $\nu = 1$ is good enough for our application.

DC band

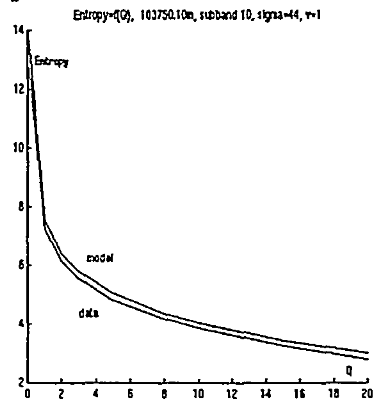
We have not decided what to do with the subband 0, yet. Actually it does not seem to make sense to try to model this subband with a Laplacian distribution since it looks like two normal distributions. Nevertheless, we are going to use a very small value of Q compared to the standard deviation. Hence we can use any model : a uniform model would fit as well. But for us, it is simpler to do the assumption of a Laplacian distribution again so as not to have to process the DC band separately.
We can see on the following curve that the fit is perfect, even with this non-sense model, for the range of bin widths we will use.

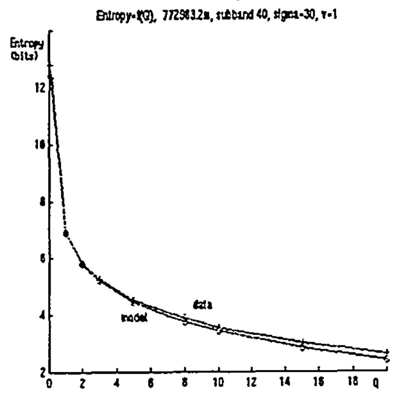


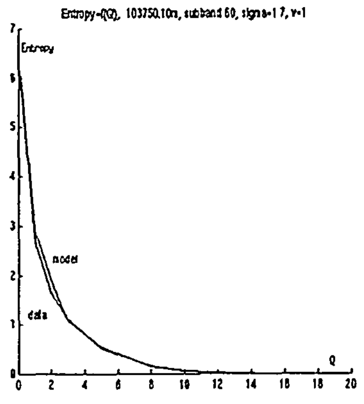
2.5.3 Entropy

Cropped Images

We plot curves of Entropy versus Q for 512x512 image data and for the estimate obtained with a Laplacian model.



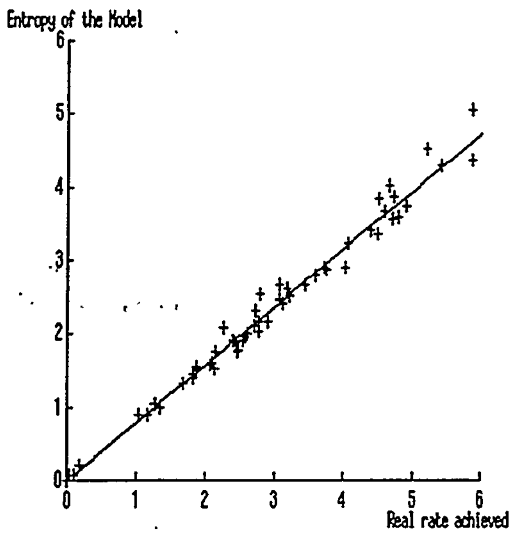




We observe that the fit is always very good. This is independent from the picture and the subband : the entropy estimate is definitely accurate for cropped images.
And this is all the more interesting that we did not take account, in the model, of the fact that we used run-length coding for real compression. Here, we can see on the graph for a subband 60 that the fit is still very good when the variance is small (which means not much information and much lossless coding gain from run-length).

768x768 Images

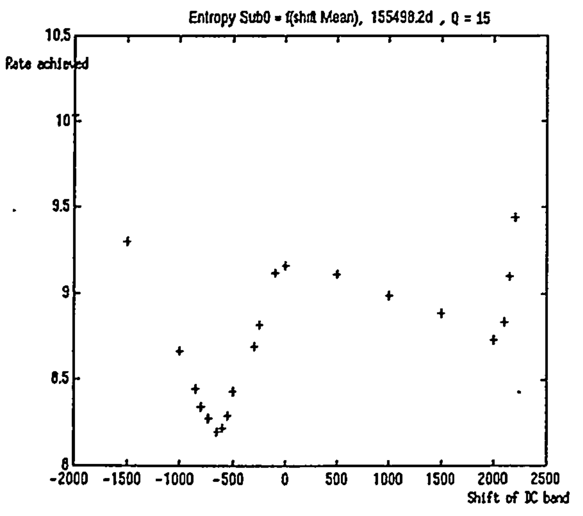
768x768 images have the peculiar property to have lots of blank spaces from the background. Only part of the image is covered with the print. The rest of it corresponds to the white card and text annotations.
For these images, run-length coding is all the more efficient.
This is why we obtain a real compressed subband with less entropy than what expected :

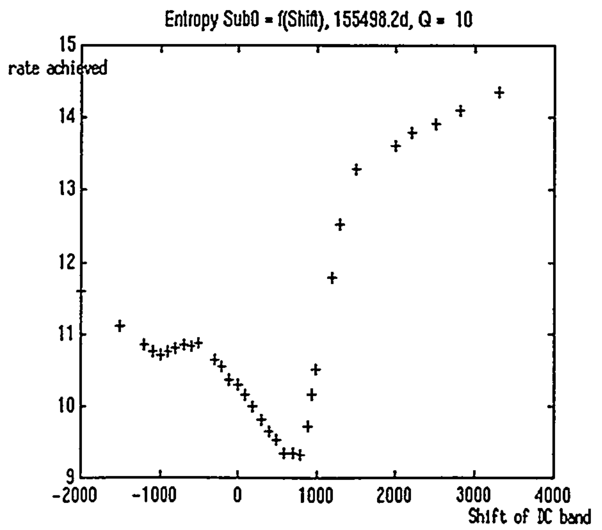


This graph is obtained from the 64 subbands of one 768x768 image. We can see that the rate achieved is about 20% less than expected but the relation between entropy estimate and real entropy is still linear. And it is the most important thing since we are going to feed this model to a Lagrangian optimizer that is somewhat based on comparisons between entropys of subbands. In that respect, a multiplicative factor in the entropy should not be too much of a problem.
Nevertheless, we then have a much worse target rate control : the rate achieved is about 20% smaller than the overall target bit-rate.

Shift of the image mean

We now concentrate only on the DC band. This subband usually gets 8 to 10 bits per pixel. And depending on what is the shift of the image mean (which is simpy 1/32 times the shift of the DC band), we will not have the same number of escape symbols.
There are usually two peaks in the histogram of this band : one for the white background and one from the fingerprint.





Here the x values are the values affected to the mean. The image we are looking at is a 768x768 image since we want an image with blank spaces and its standard deviation is about 1600.

We can see on these two curves obtained for two typical values of Q (Q=10 and Q=15) that there are two local minima. Other experiments have shown that these two minima always exist even though the difference between the two minimal rates achieved can vary.

We can explain these two local minima as follows :

- The minimum obtained for a negative value of the mean corresponds to the encoding of all zero values as one-to-one mapped symbols. Its position can be computed with the formula :

$$Mean = -Q * \#Positive\_Symbols\_in\_bijection \tag{2.22}$$

- The minimum obtained for a positive value of the mean corresponds to the attempt to map most of the quantized values from the fingerprint to symbols without using escape symbols. Experiments on other images, when Q = 12.5, show that this position usually give a better rate than the negative mean.
  Nevertheless, it is also more dangerous since the slope gets steeper after this minimum.
  As a rule of thumb that would have to be better tuned, we can use :

$$Mean = +0.5\sigma\sqrt{\frac{Q}{12.5}} \tag{2.23}$$

  The 0.5 allows us to be in the flat region before the minimum and prevents us from being in this steep region where the rate gets worse quickly. We use $\frac{Q}{12.5}$ since we have done many experiments on this value and it is a good reference value. Finally, the square root is due to the fact that the progression seems to be slower than linear and the shift would be too important for smaller target rates.
  This formula has been tested on 512x512 images and seem to work properly. Nonetheless, it is really empirical and there would probably be a better and simpler decision to find.

2.6 Optimization with Lagrange multipliers method

The optimization problem can be written as :
Find $\{Q\}_{0\leq i\leq 63}$ that solves

$$\begin{cases} Min\ D \\ H \leq Rh \end{cases}$$

$$D = \sum_{i=0}^{i=63} w_i \frac{D_i(Q_i)}{4^{d_i}}$$
$$H = \sum_{i=0}^{i=63} \frac{H_i(Q_i)}{4^{d_i}}$$

Q_i, Scaling factor of subband i
D_i, distortion of subband i
d_i, depth of subband i
w_i, weight factor for subband i
H_i, entropy of subband i

The idea of the algorithm is to have, for each subband, a lower value Q_{lower} and an upper value Q_{upper} that define the range in which to look for the optimal value. The goal is to make these two 64D-vectors converge to the optimal Q values where the overall rate achieved is the target bit-rate.

We are going to use the fact that $D_i = f(H_i)$ are decreasing functions on the restricted intervals $[Q_i^*, \infty[$.

2.6.1 Global picture of the algorithm

- Initiate $\{Q^0_{upper}\}_{0\leq i\leq 63}$, $\{Q^0_{lower}\}_{0\leq i\leq 63}$
  Compute $\lambda^0_{upper} = Min_i \frac{w_i \nabla D_i((Q^0_{upper})_i)}{\nabla H_i(Q^0_{upper})_i}$,
  $\lambda^0_{lower} = Max_i \frac{w_i \nabla D_i((Q^0_{lower})_i)}{\nabla H_i(Q^0_{lower})_i}$
- Step $k \geq 1$:
  - Set $\lambda^k = \frac{\lambda^{k-1}_{upper}+\lambda^{k-1}_{lower}}{2}$
  - $\forall\ 0 \leq i \leq 63$, Find Q_i that minimizes $Lag_i = w_i D_i - \lambda^k H_i$.
  - Compute $H^k = \sum_{i=0}^{i=63} \frac{H_i(Q^k_i)}{4^{d_i}}$, compare to Rh :

    $$\begin{array}{ll} If\ H^k \leq Rh,\ then & \{Q^k_{upper}\} = \{Q^k\},\ \lambda^k_{upper} = \lambda^k, \\ & \{Q^k_{lower}\} = \{Q^{k-1}_{lower}\},\ \lambda^k_{lower} = \lambda^{k-1}_{lower}. \\ If\ H^k > Rh,\ then & \{Q^k_{lower}\} = \{Q^k\},\ \lambda^k_{lower} = \lambda^k, \\ & \{Q^k_{upper}\} = \{Q^{k-1}_{upper}\},\ \lambda^k_{upper} = \lambda^{k-1}_{upper}. \end{array}$$
  - Test convergence on λ and H^k.

This algorithm always converge if initial bounds have been correctly chosen, that is if the target rate can be reached in this range of Q values.

The algorithm indicates *minimize the Lagrangian function since finding an extremum for this function is the same as finding a minimum. As a matter of fact, distortion and entropy are always positive and λ negative so the Lagrangian function is positive; distortion is a decreasing function and entropy and increasing function; so the lagrangian function will first decrease with distortion and then increase with entropy.*

Tets of convergence are made on both λ and H since one or the other can converge faster depending on the slopes of rate-distortion curves.

2.6.2 Implementation

For reasons of efficiency, the algorithm implemented is not exactly the same as the one described in the previous section.
We still use a bisection method to define the next λ in the main loop. Nevertheless, the first values of λ give a too large range of values. Starting with these values, we then need 20/25 steps in the process. And every step requires new estimates of distortion and entropy that cost much time. This is also why we do not use a Newton method that would require extra estimates even though the number of steps would be smaller.

So we replace the initiation step by a first set of iterations as follows :

- *Initiate* $\{Q^0_{upper}\}_{0\leq i\leq 63}$, $\{Q^0_{lower}\}_{0\leq i\leq 63}$

- $Q^0 = \frac{\{Q^0_{lower}\}+\{Q^0_{upper}\}}{2}$

```
• $\lambda^0 = Mean_i \frac{w_i \nabla D_i((Q^0)_i)}{\nabla H_i((Q^0)_i)}$
• Set $H_{upper} = 0$, $H_{lower} = \infty$, $k = 0$
• While $(H_{upper} > Rh)$ or $(H_{lower} < Rh)$
    – $\forall\ 0 \leq i \leq 63$, Find $Q_i$ that minimizes $Lag_i = w_i D_i - \lambda^k H_i$.
    – Compute $H^k = \sum_{i=0}^{i=63} \frac{H_i(Q_i^k)}{4^{d_i}}$, compare to $Rh$ :
      If $H^k \leq Rh$, then $\{Q_{upper}^k\} = \{Q^k\}$, $\lambda_{upper}^k = \lambda^k$,
                             $\{Q_{lower}^k\} = \{Q_{lower}^{k-1}\}$, $\lambda_{lower}^k = \lambda_{lower}^{k-1}$,
                             $\lambda^{k+1} = 0.1\lambda^k$
      If $H^k > Rh$, then $\{Q_{lower}^k\} = \{Q^k\}$, $\lambda_{lower}^k = \lambda^k$,
                             $\{Q_{upper}^k\} = \{Q_{upper}^{k-1}\}$, $\lambda_{upper}^k = \lambda_{upper}^{k-1}$,
                             $\lambda^{k+1} = 1.9\lambda^k$
    – Increase $k$
```

Also in the implementation, we verify that $(Q_{upper})_i - (Q_{lower})_i$ is greater than a certain tolerance. Otherwise we assume that this subband has converged and we do not work on it anymore.

For initial values of Q_{upper} and Q_{lower}, we choose respectively $Cst1 * \sigma$ and the breakpoint as defined in 2.20. Cst1 is typically 10 or 15. It seems a pretty large value but we can see otherwise that the last subbands get the minimum of bits even at a target rate of 0.75 bit/pix. Also, the encoder 1 with the high-rate assumption, could sometimes define a value $Q = 30\sigma$.
These initial values have much influence on the behaviour of the quantizer.
A Q_{lower} too large can have the effect of producing an artefact such as bars in the image : we experienced this on one image that had a subband where the standard deviation was exceptionnaly high. We used the criterion 2.21 for Q_{lower}. This subband had much more distortion because the maximum of its entropy was set too low because of the value of Q_{lower}.

The weights w_i we use here have been computed to optimize the PSNR, taking account of the biorthogonality of the filters. They do not have much influence on the final result since they vary between 0.5 and 1.5 and most of them are very close to 1.

Chapter 3

Huffman Coding

3.1 Introduction

An implementation of the Huffman encoding/decoding process as specified by the FBI already exists. We want to add new features that will make it possible to conduct new experiments in the future. The program has also to be flexible enough to be easily adapted to other kinds of compressions like video compression.

3.2 The Huffman coding adopted by the FBI

We will not try to explain the huffman coding which is a well-known scheme of lossless compression, but will emphasize its particularities in the FBI specification.

3.2.1 The alphabet

The output of the quantizer is a set of signed integer values coded with 16 bits. The role of the alphabet is to map these integer values to a smaller number of symbols (254).
These symbols will then be attributed codewords and the concatenated codewords make the entropy coded segment.

Since most of the subbands have a large number of zeros, the symbol mapping also consists in a zero run-length encoding. Large integer values are mapped with escape symbols : an escape sequence is made of the escape symbol and the integer value that is not entropy encoded.

The next table represents the FBI alphabet (as in [1]).

Position	Value
1	zero run length 1
...	...
100	zero run length 100
101	esc for pos 8 bit coeff
102	esc for neg 8 bit coeff
103	esc for pos 16 bit coeff
104	esc for neg 16 bit coeff
105	esc for run - 8 bits
106	esc for run - 16 bits
107	coeff value - 73
...	...
254	coeff value 74

A new feature would be the possibility of having any kind of alphabet with either less than 255 symbols (so that can be represented with one byte) or more than 255 symbols (two bytes).

The idea is that the different subbands do not have the same kind of data : whereas the lower subbands have significant integer values, the higher subbands have mostly zeros.
Moreover, the specification indicates that there should be at least three different blocks for the Huffman encoding (and not more than eight) so these three different encoders could use different alphabets.

3.2.2 Adàptative coding

The FBI entropy encoder is adaptative. That is to say that for every image, new tables are built on the basis of its data. The scheme consists in counting frequencies of symbols, then constructing the tables and the correspondance betwen a symbol and its codeword ([2], Annex C and K).

Nevertheless, all fingerprint images have some similar properties and we would like to be able to compute a codebook for one block with a great number of images and then reuse it to encode all images. It saves time and allows not to transmit the tables (BITS and HUFFVAL like in the JPEG standard).

3.2.3 File structure

The file structure of the FBI specification is once again based on the JPEG format.
The compressed image is divided in headers and blocks. Every header or block begins with a marker.
Where huffman process is concerned, we need to know the meaning of the following markers :

Block header :

SOB	Ls	Td

Ls(16 bits) is the length of the marker and Td(8 bits) is the Huffman table selected.

Huffman table header :

DHT	Lh	Th	L_i	...	L_{16}	$V_{1,1}$	...	$V_{16,L_{16}}$

Th(8 bits) is the Huffman table identifier. The L_i's(8 bits) are the elements of the list BITS and the V_{ij}'s are the elements of the list HUFFVAL.

We clearly see that there is no variable for the identification of the alphabet (since there is only one in the FBI specification) and the fact that L_i's and V_{ij}'s are coded with 8 bits do not allow us to use more than 255 symbols.
Also, it is not possible to have a reduced segment in the case of an external codebook that does not need to specify BITS and HUFFVAL in the file.

Hence we will have to change the file format to be able to include the new features and particularly so that the decoder will be able to reckognize the different options.
Nevertheless, we always want to have the possibility to comply with the old FBI specification since it has been published and it is not bound to change.

3.3 The new encoder

In this section, we will try to stick to the description of the new features, without alluding to the implementation aspects.

3.3.1 Creating an alphabet

The main issues of the alphabet creation are the possibility of having run length on other values than zeros, the possibility of using different alphabets and the possibility of having alphabets with more than 255 symbols.

So we define a new variable Tc (inspired from the Tc in the JPEG spec) that is the alphabet number.
The file of specification also contains the symbol representation type : either one unsigned byte or two unsigned bytes.
Finally, we have four ways of mapping an integer value to a symbol : with run length or not, with an escape symbol or not.

Here is the example of the FBI alphabet :

Value_of_Tc: 2
Type_of_SYMBOLS: unsigned char

Run length with explicit symbols

Value	Min_Length	Max_Length	Data_Symbol_Offset
0	1	100	0

Run length coded by escape symbols

Value	Min_Length	Max_Length	Esc_Symbol	#SYMBOLS
0	101	255	105	1
0	256	65535	106	2

Individual values coded with explicit symbols

Min_Value	Max_Value	Data_Symbol_Offset
-73	74	180

Individual values coded by escape symbols

Min_Value	Max_Value	Esc_Symbol	#SYMBOLS
75	255	101	1
-255	-74	102	1
256	32756	103	2
-32756	-256	104	2

The format seems to be self-explanatory but let us precise some points :

When values are mapped with explicit symbols, the value is converted to a symbol shifted from the offset Data_Symbol_Offset..
For example, a run length of 45 zeros will be mapped to the symbol 45.

When values are mapped with escape symbols, we first write an Escape symbol Esc_Symbol and then the integer value coded with a number of symbols #SYMBOLS.
For example, the value 300 will be attributed the symbol 103, followed by two bytes representing 300 (0x01, 0x2C), if Type_of_SYMBOLS is unsigned char and only one byte otherwise.

To test a new alphabet, we implemented two tools : testAlp and testMap. The first one uses a test file to verify that some key values are correctly mapped.
The second one maps a whole file and does the invert transform, then compares the input and the output : if there are differences, there are mistakes in the alphabet.

3.3.2 External codebooks

We now can build external codebooks that are saved in a file and can be reused for any image, without creating adaptative tables.

The tool that enables building a codebook with one alphabet and one or several blocks, StatDriver, also returns the entropy of the source.

The format of this file is not important for the user. It is readable, though.
It contains the Th value and all the symbols, the length of their codewords, the value of the codeword in decimal and the codeword in binary.

3.3.3 The file format

We have already seen that we need a new format for the huffman header.

On the contrary, the block header stays the same (see 3.2.3) since an huffman table can use only one alphabet, which means that the Td (or Th) determines the Tc.

When the option *FBI format* is not on, the header becomes :

DHT	Lh	Tc	Th	L_i	...	L_{16}	$V_{1,1}$	...	$V_{16,L_{16}}$

with the L_i's and V_{ij}'s written with 8 or 16 bits depending on the number of symbols in the alphabet.

In the case of a reduced header (for an external codebook), we obtain :

DHT	Lh	Tc	Th

You can also have a reduced header with the *FBI format*. Then you do not have the Tc variable. When there are multiple external codebooks, we write a new header for each codebook whereas several BITS/HUFFVAL can be specified in the same header.

3.3.4 Options specification

Since the user can now have a certain degree of freedom with multiple options, their specification is specified in a file.

Let us explain with an example :

```
Output_File:       out.enc
FBI_File_Format:   no
Verbose:           no
Run_Length:        yes
Restart_Markers:   0

Encoder(Th)   Alphabet   Ext_Codebk   Use_Blocks
0             fbi.tab    no           0
10            fbi.tab    cdb0         1,2
```

Run_Length allows to enable or disable the run length encoding (for alphabets that can support run length coding).

Restart_Markers indicates the restart interval length. When 0, there is no restart interval. This is also the variable Ri in the Restart interval segment :

DRI	Lr	Ri

Now let us have a look at the table that indicates how to build the different encoders :

An encoder is always referred to thanks to its Th. With a Th, there can be only one alphabet corresponding (one Tc). The user knows the name of the alphabet (and not necessarily its Tc). Then we have to know if the encoder has to be built from input data or if it is an external codebook. If it is, then we must specify the name of the file (cdb0 in the example). Finally, we have to know which blocks are encoded with which encoder (or with which blocks the tables of an encoder is built). This is what Use_Blocks is aimed at.

Note that if the Th of an external codebook does not correspond to the one in the external codebook file, the program will break.

3.4 Implementation

The implementation is made in C++, for its flexibility, the possibility of reusing code and its safety. We use KCC compiler and purify for debugging.

Whereas the memory issues are not a main concern since we are working on workstations, we try to keep the code as fast as possible, optimizing some key portions of the code like the bit-level encoding and decoding process.

The code essentially uses functors since the process is pretty linear. Most of the classes and functions are templated on the type of SYMBOLS. Only highest-level classes are not templated since they must instantiate templated objects.
We also use Standard Templated containers widely for their built-in memory allocation and iterators.

3.4.1 Encoder

Flowcharts of the main highest-level classes can be found in annex.

The encoder only needs to be able to produce one of the different types of files that the decoder can reckognize. So the encoder always gathers the header segments at the top of the file and writes all the blocks after.

After all alphabets and external codebooks are fetched from file, every block is mapped with the correct alphabet then all the internal codebooks are built. Finally, every block is encoded with the right encoder and written to the file.
The bit-level encoder consists in a buffer fed with codewords and whose bytes are popped as soon as there are more than 8 bits in it. The encoder has always to take account of the fact that there can be restart markers and that a 0xFF byte has a special meaning and indicates the beginning of a marker (if not followed by a 0x00 byte).

3.4.2 Decoder

Flowcharts of the main highest-level classes can be found in annex.

The decoder does a one-way parsing of the file. It looks for markers and when it does not reckognize a DHT marker or a SOB marker, it exits.

When a DHT marker is found, the header segment is read and alphabets and external codebooks are fetched.

When a SOB marker is found, a block is read and decoded with the tables already fetched.

The byte-level decoding is made with two objects passing each other messages. The first one has a one-byte buffer and requires new bytes from the second object, which holds a reference to the input block. The first object is responsible for traversing the tree, reading what it finds on the leaves and returning it to the second object. Obviously, markers and special bytes (like the 0xFF) have to be reckognized and processed : this is the job of the second object.

The fact that the buffer can contain only one byte is the result of optimization tests. Shifting information in the buffer is a great loss of time. So the buffer is refilled one byte at a time.

3.4.3 Tests

Tests on the encoder and decoder have been made using all new features and parameters that the user can vary : with or without run-length encoding, with or without restart markers, with external codebooks and internal codebooks in the same compressed file, using different types of alphabets in the same compressed file (and particularly one with less than 255 symbols and another with more than 255 symbols).
We also made tests on compression with encoder 2 and decompression with decoder 1 and compression with encoder 1 and decompression with decoder 2.
This means that we have tested compliance with FBI specification and also the new features added to the FBI specification.

Experiments with the alphabet have shown that the FBI alphabet was probably a very good alphabet for fingerprint images and particularly there is really a need for 100 symbols dedicated to run-length.

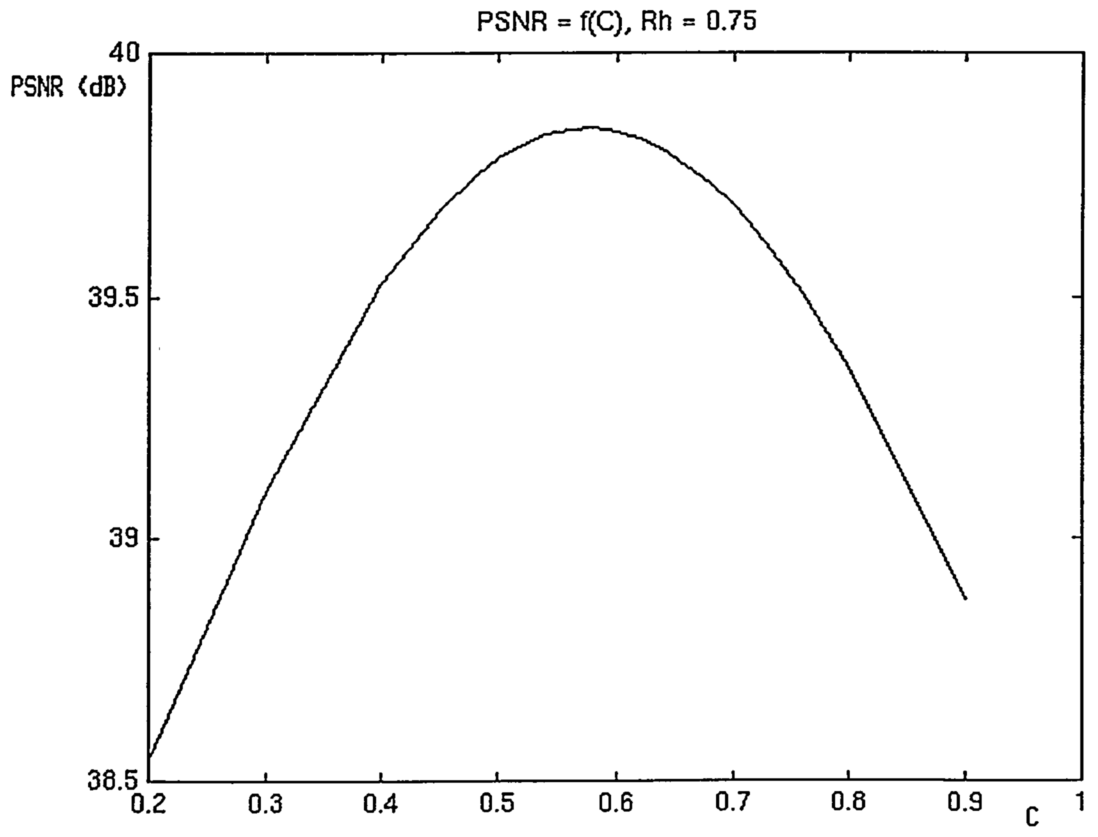
[illegible]

Chapter 4

Experiments

4.1 Reconstructed bin center C

We have seen in 2.4.2 (page 18) that the value C = 0.44 in encoder 1 could not be optimal. So we run our new compression scheme : decomposition with FBI 9/7 filters, Bit Allocation using the Lagrangian Optimizer, entropy coding with 6 Huffman encoders and we vary the parameter C. The following curve is obtained for 17 images (size 768x768) at a target rate of 0.75 (rate achieved around 0.55) and the step of C incrementation was 0.02 around the peak.



The peak signal-to-noise ratio here is computed as follows :

$$PSNR = 20\log_{10}\left(\frac{255}{\sigma_{error}}\right) dB \tag{4.1}$$

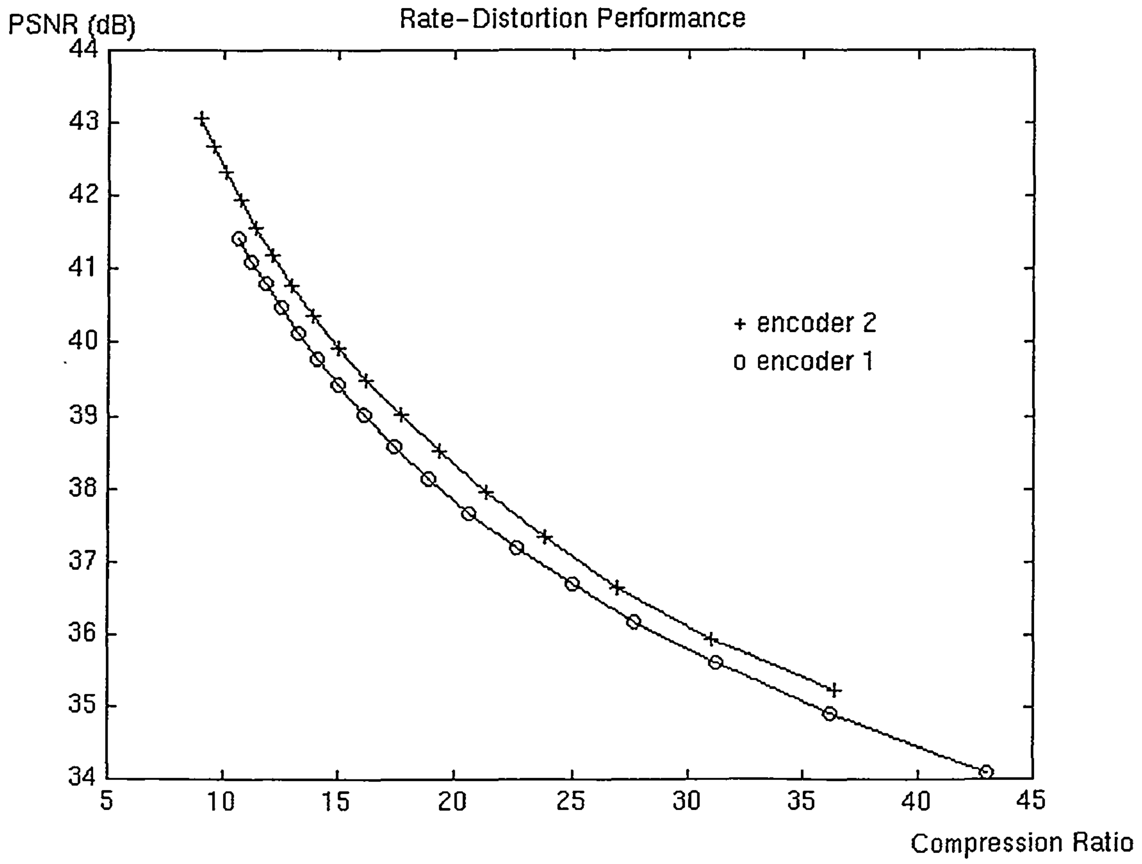
We will see in the next section that this definition of the PSNR, pretty unusual, is maybe not the best. It gives PSNRs around 39dB, which seems high even at this rate. But here, we simply want to know the behaviour of the curve.

We can observe a maximum around 0.58. Again, this can be explained by the fact that we have generalized gaussian distributions for which the centroid of each bin is shifted towards the y axe from the bin center. And this causes a 0.25 dB improvement over the former encoder.

We tried also this value on a few cropped images and 832x768 images from the NIST database and it also seems to be the best choice in terms of distortion.

4.2 Rate-Distortion Curves

The following experiment has been conducted on 32 images (size 768x768), target rates were in the range 0.35 to 1.10 with a step of 0.05. We compare encoder 2 with C=0.58, 6 blocks and the lagrangian optimizer based on Laplacian estimates versus encoder 1. Nothing has been modified in the DWT : filters are the same.



This curve is using 4.1 as a measure of the PSNR and we can see that the results obtained are much higher than expected (about 4dB) if we compare to the rate-distortion curve published in [4] using $PSNR = 20\log_{10}\left(\frac{255}{RMSE}\right) dB$, where RMSE is the root mean square error.
Actually this comes from the fact that RMSE is usually 2 or 3 times as big as the standard deviation, which means that the mean of the error is not negligeable before the standard deviation of the error.

We can see that we have an improvement of 0.5dB. This is due to the parameter C (0.2dB) and the use of 6 blocks for Huffman coding instead of 2 (about 0.2/0.3 dB usually).
Which means that we have no gain (but no loss) in the bit allocation itself.

Another experiment with 60 images of the NIST database (832x768) with the same number of blocks (2) and the same value C (0.44) resulted in two rate-distortion curves absolutely undifferentiated. Hence, the bit-allocation of the entropy-constrained quantizer does not make the difference and probably needs better tuning.

4.3 Quality of decompressed images

The quality of decompressed images vary with the type of image.

832x768 images of the NIST database show artefacts in the background even with 1.5 bit/pixel. This is true for both encoders 1 and 2, for both live scans and paper scans.
And the most curious thing is that it sometimes appears with good quality images.

512x512 images still have a very good quality with a compression ratio of 15:1 although they are difficult to compress since they contain more information.

768x768 images are easy to compress and ringing artefacts only appear around a compression ratio of 30:1.

Conclusion

As a conclusion, we could say that we have implemented a complete encoder and a complete decoder. The flexibility from the C++ language makes it possible to replace one box of the process by another and particularly to compare two steps of the process in details. For example, the software can now use either the box doing the high-rate bit-allocation or the entropy-constrained quantizer.

This is all the more important that, as we have seen previously, the performance is not dramatically improved compared to the first encoder.
Other experiments have now to be conducted to discover the exact influence of each method and each parameter. Which is possible thanks to the software.

Bibliography

[1] CJIS, FBI. WSQ Gray-scale Fingerprint Image Compression Specification, Feb 1993.

[2] W.B. PENNEBAKER, J.L. MITCHELL. JPEG Still Image Data Compression Standard.

[3] W.H. BEYER. CRC Standard Mathematical tables, 25th edition.

[4] J.N. BRADLEY, C.M. BRISLAWN. Proposed first-generation WSQ bit allocation procedure, Sep 1993.

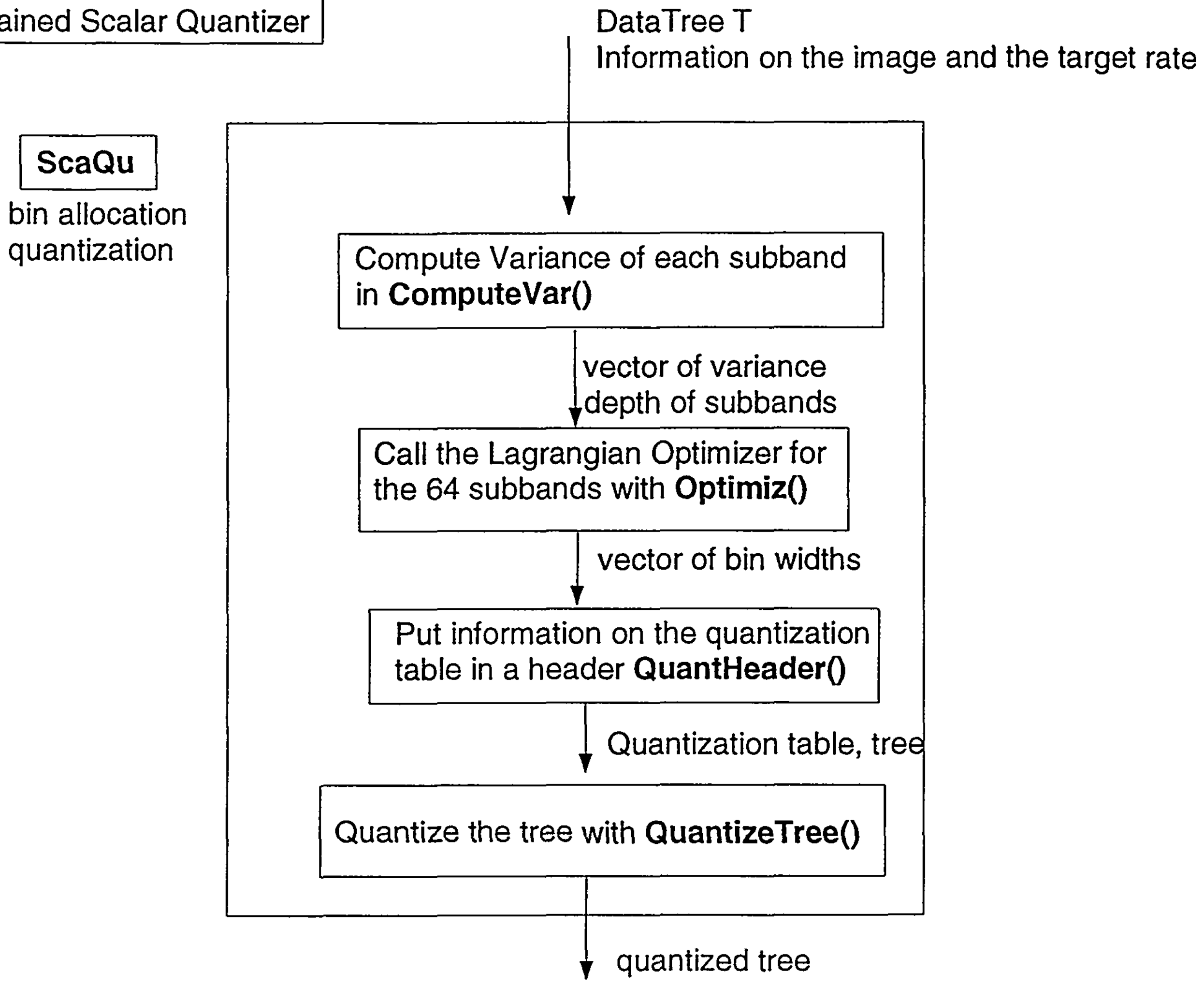
[5] J.N. BRADLEY, C.M. BRISLAWN, T. HOPPER. The FBI Wavelet/Scalar Quantization Standard for gray-scale fingerprint image compression, Visual Information Processing II, April 1993.

[6] C.M. BRISLAWN, J.N. BRADLEY. The FBI Compression Standard for Digitized Fingerprint Images, Conference on Apllications of Digital Image Processing, Aug. 1996.

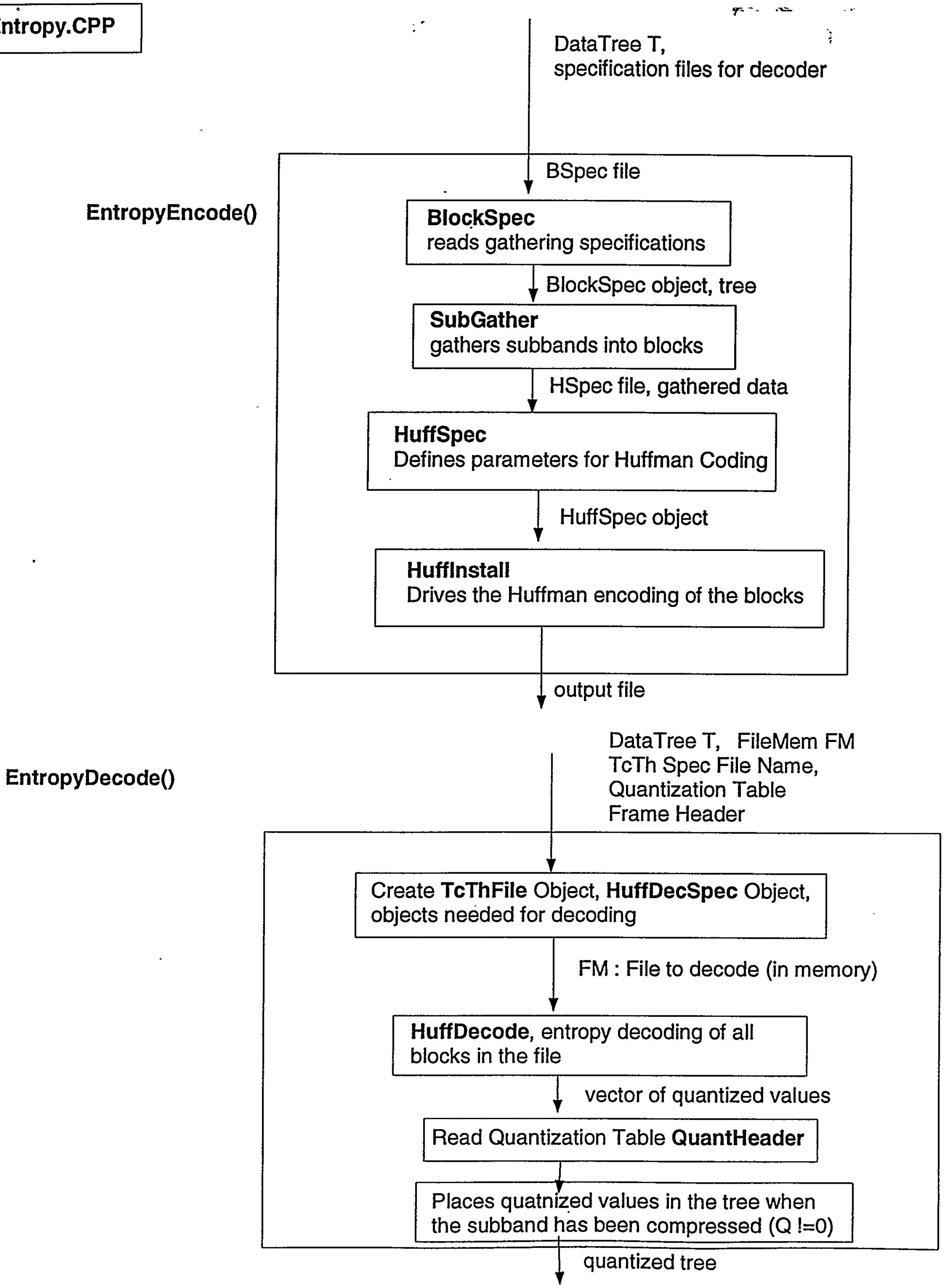
[7] C.I. WATSON. NIST Special Database 9, Mated Fingerprint Card Pairs, Feb 1993.

Appendix A
FlowCharts

ECSQ.CPP
Entropy Constrained Scalar Quantizer

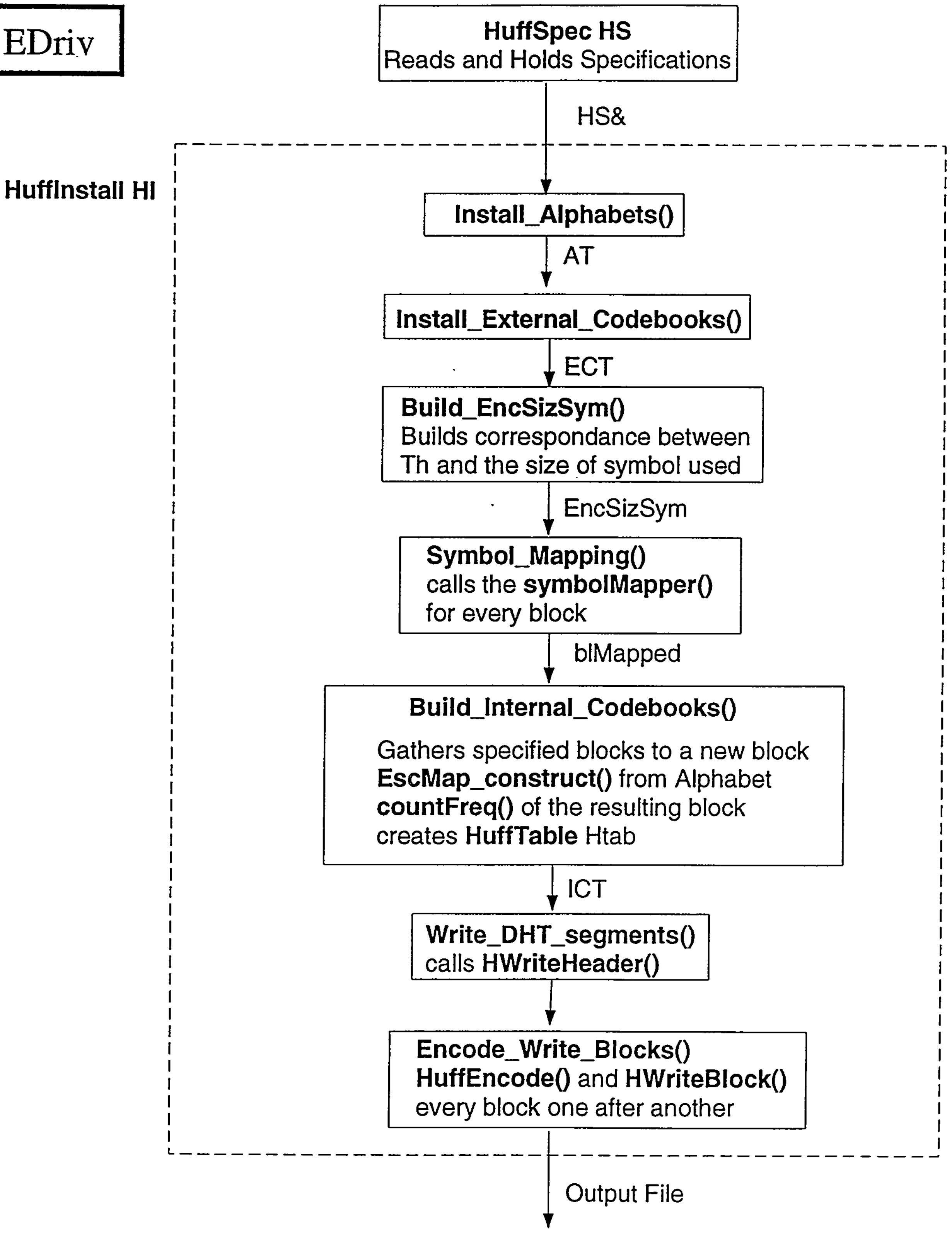

DataTree T
Information on the image and the target rate
ScaQu
bin allocation
quantization
Compute Variance of each subband in **ComputeVar()**
vector of variance
depth of subbands
Call the Lagrangian Optimizer for the 64 subbands with **Optimiz()**
vector of bin widths
Put information on the quantization table in a header **QuantHeader()**
Quantization table, tree
Quantize the tree with **QuantizeTree()**
quantized tree

Entropy.CPP



EntropyEncode()
DataTree T,
specification files for decoder
BSpec file
BlockSpec
reads gathering specifications
BlockSpec object, tree
SubGather
gathers subbands into blocks
HSpec file, gathered data
HuffSpec
Defines parameters for Huffman Coding
HuffSpec object
HuffInstall
Drives the Huffman encoding of the blocks
output file
EntropyDecode()
DataTree T, FileMem FM
TcTh Spec File Name,
Quantization Table
Frame Header
Create TcThFile Object, HuffDecSpec Object,
objects needed for decoding
FM : File to decode (in memory)
HuffDecode, entropy decoding of all
blocks in the file
vector of quantized values
Read Quantization Table QuantHeader
Places quatnized values in the tree when
the subband has been compressed (Q !=0)
quantized tree

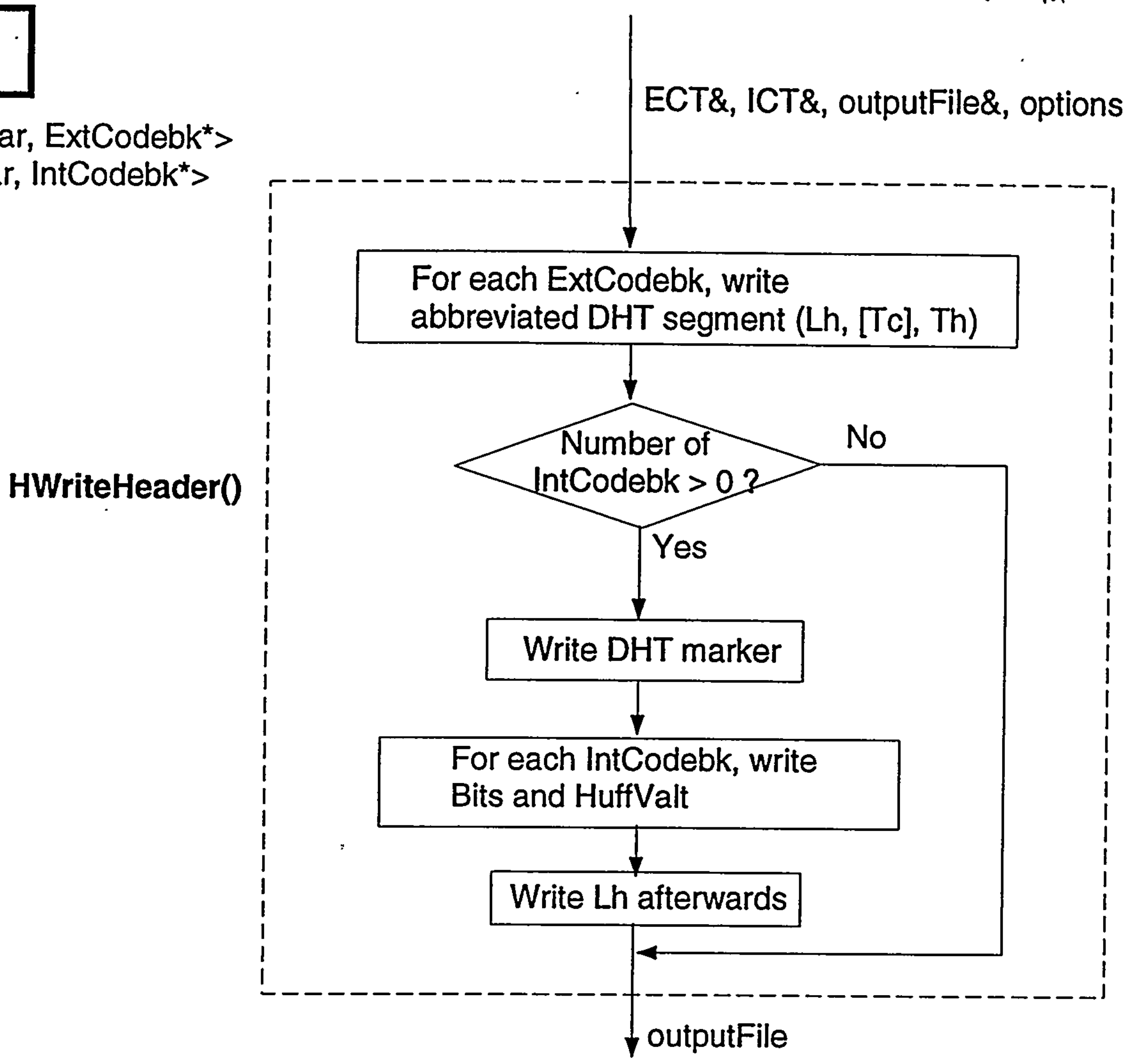
EDriv

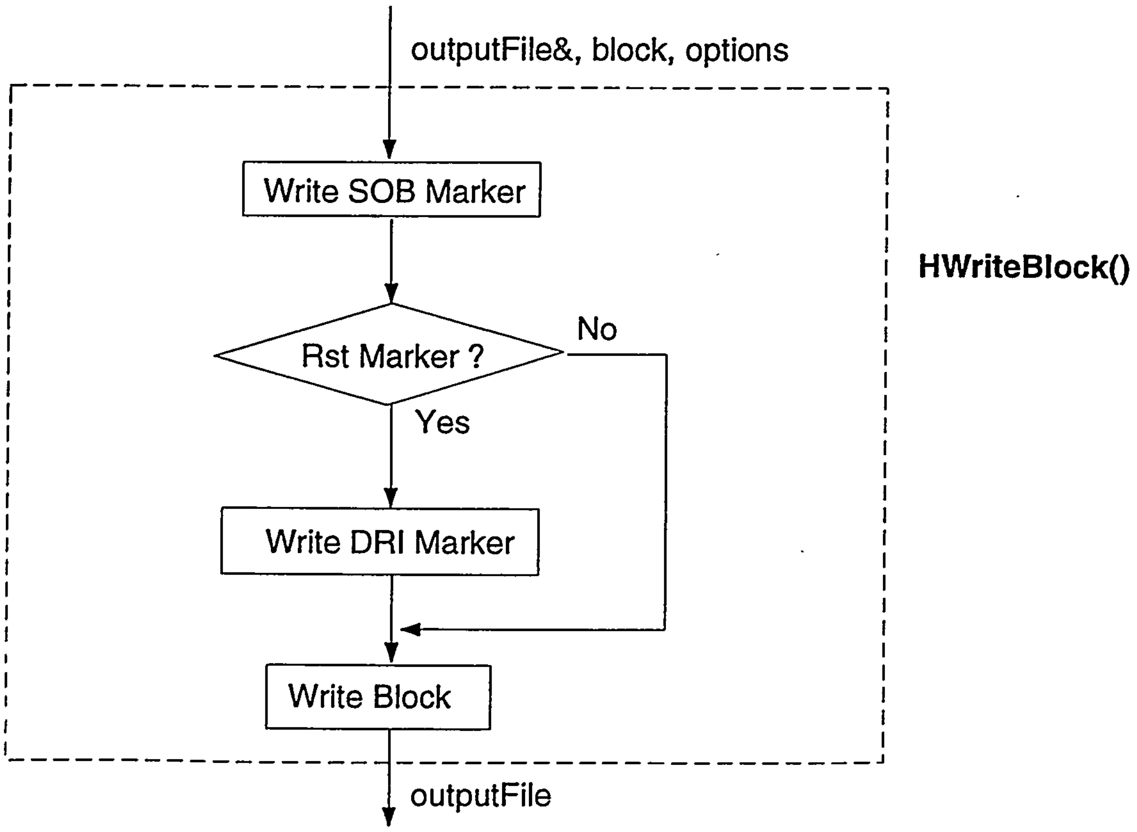

HuffSpec HS
Reads and Holds Specifications
HS&
HuffInstall HI
Install_Alphabets()
AT
Install_External_Codebooks()
ECT
Build_EncSizSym()
Builds correspondance between
Th and the size of symbol used
EncSizSym
Symbol_Mapping()
calls the symbolMapper()
for every block
blMapped
Build_Internal_Codebooks()
Gathers specified blocks to a new block
EscMap_construct() from Alphabet
countFreq() of the resulting block
creates HuffTable Htab
ICT
Write_DHT_segments()
calls HWriteHeader()
Encode_Write_Blocks()
HuffEncode() and HWriteBlock()
every block one after another
Output File

AT : map<unsigned char, AlphTable*>
ECT : map<unsigned char, ExtCodebk*>
ICT : map<unsigned char, IntCodebk*>
EncSizSym : map<unsigned char, unsigned short>
blMapped : vector<VecMap>

HWriteFile

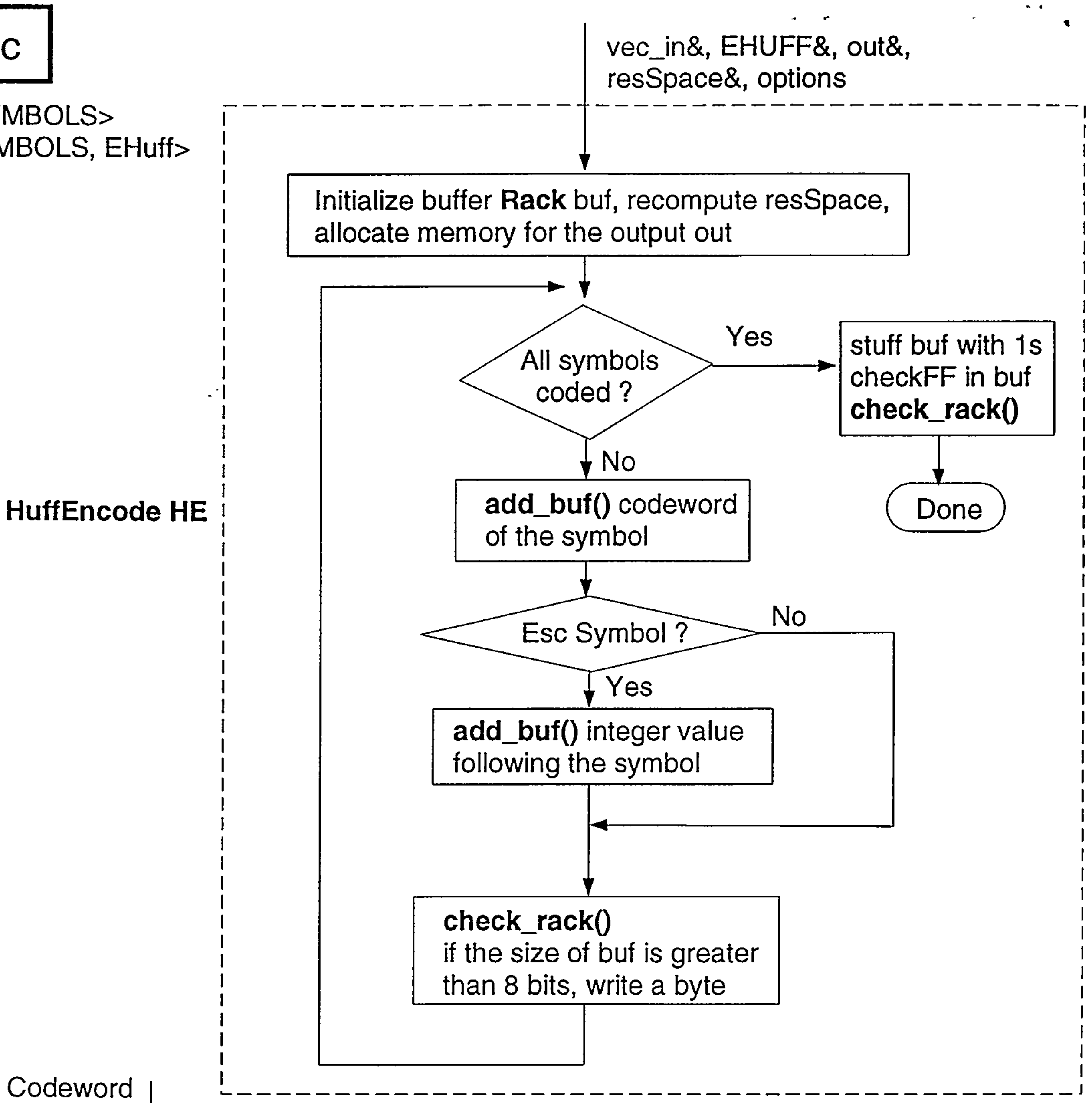
ECT : map<unsigned char, ExtCodebk*>
ICT : map<unsigned char, IntCodebk*>


ECT&, ICT&, outputFile&, options
HWriteHeader()
For each ExtCodebk, write abbreviated DHT segment (Lh, [Tc], Th)
Number of IntCodebk > 0 ?
No
Yes
Write DHT marker
For each IntCodebk, write Bits and HuffValt
Write Lh afterwards
outputFile

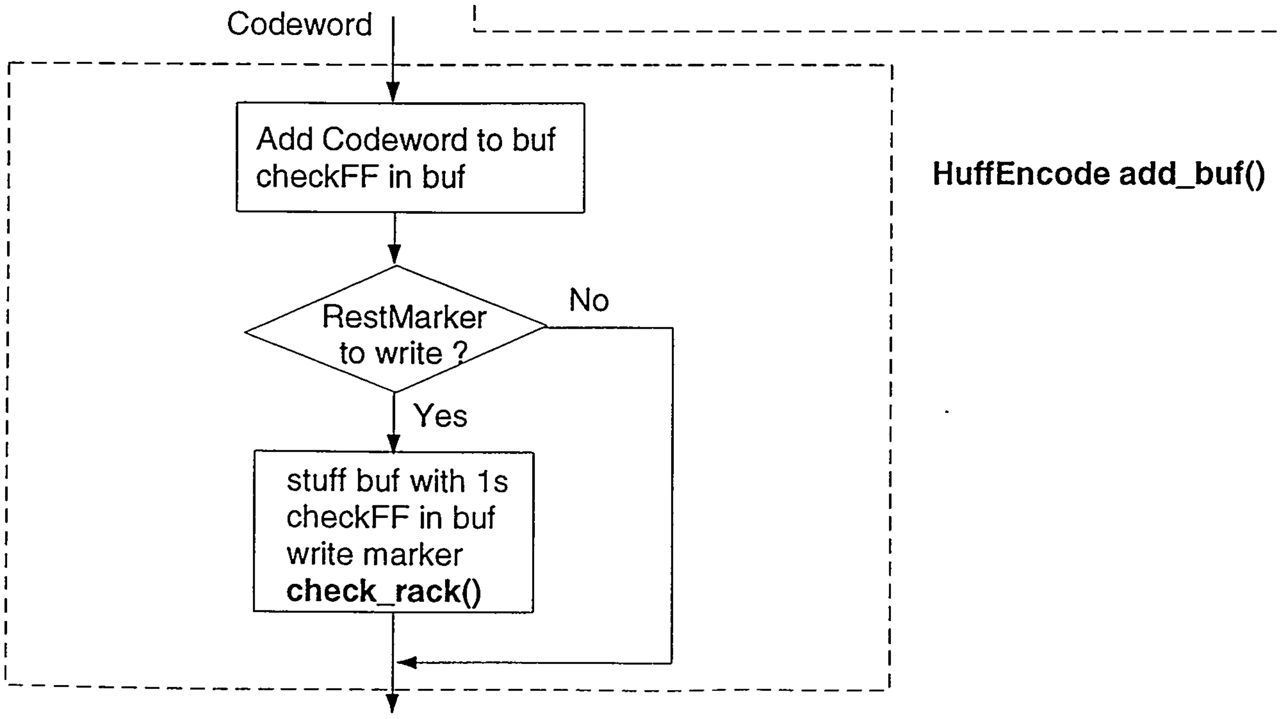


HuffEnc

vec_in : vector<SYMBOLS>
EHUFF : map<SYMBOLS, EHuff>
out : char*

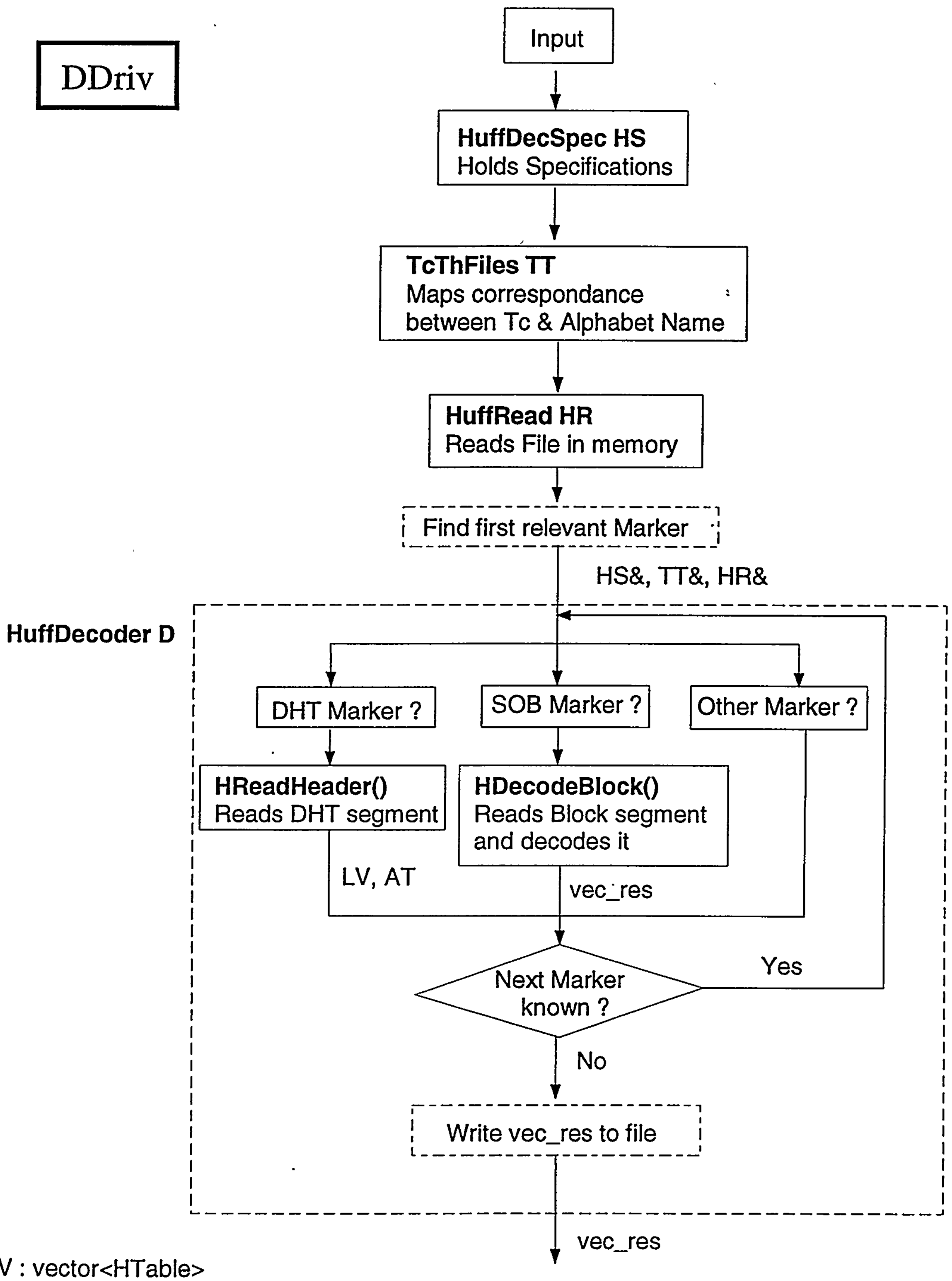


vec_in&, EHUFF&, out&,
resSpace&, options
HuffEncode HE
Initialize buffer Rack buf, recompute resSpace, allocate memory for the output out
All symbols coded ?
Yes
No
stuff buf with 1s checkFF in buf check_rack()
Done
add_buf() codeword of the symbol
Esc Symbol ?
Yes
No
add_buf() integer value following the symbol
check_rack() if the size of buf is greater than 8 bits, write a byte



Codeword
HuffEncode add_buf()
Add Codeword to buf checkFF in buf
RestMarker to write ?
No
Yes
stuff buf with 1s checkFF in buf write marker check_rack()

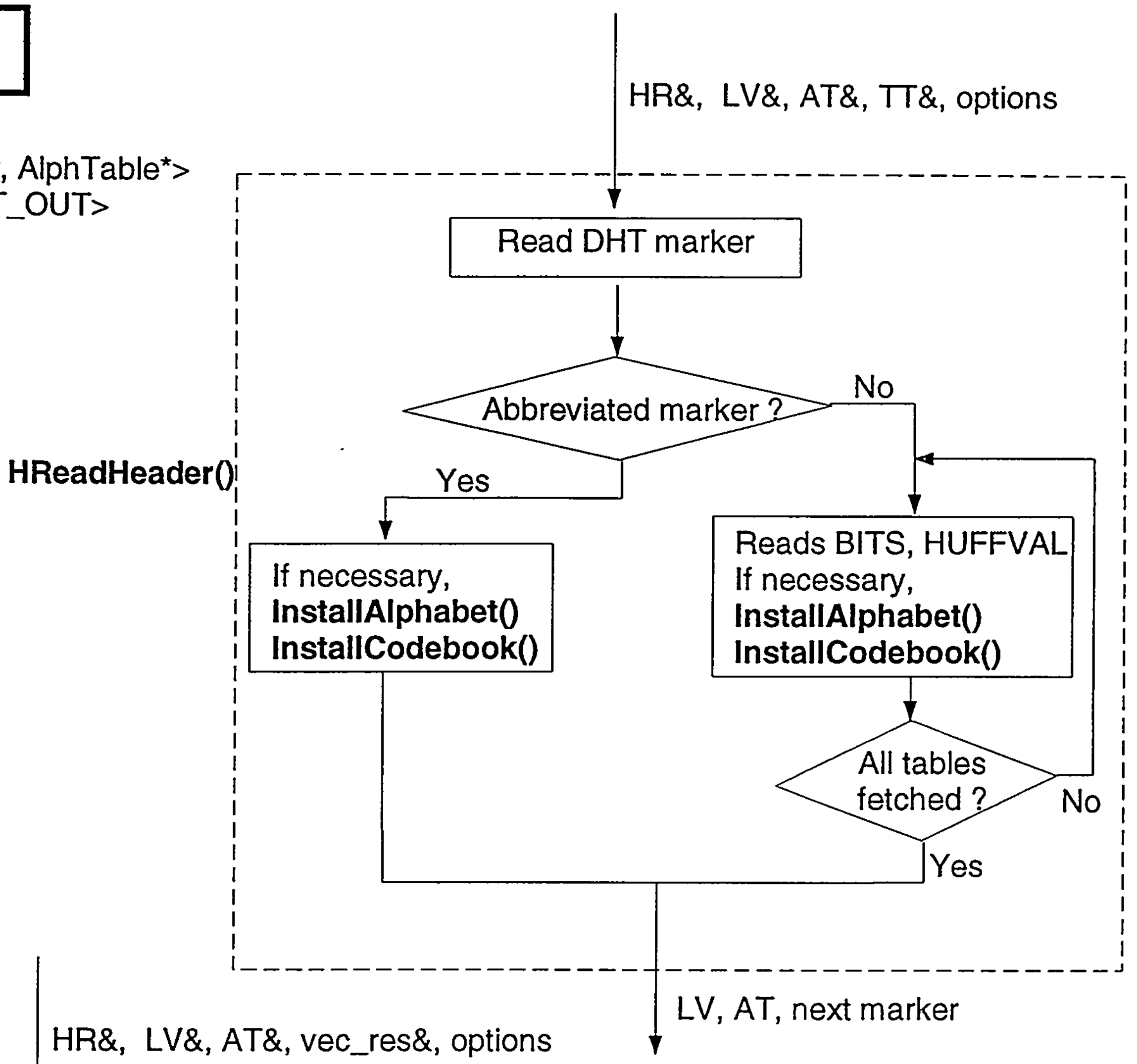
DDriv

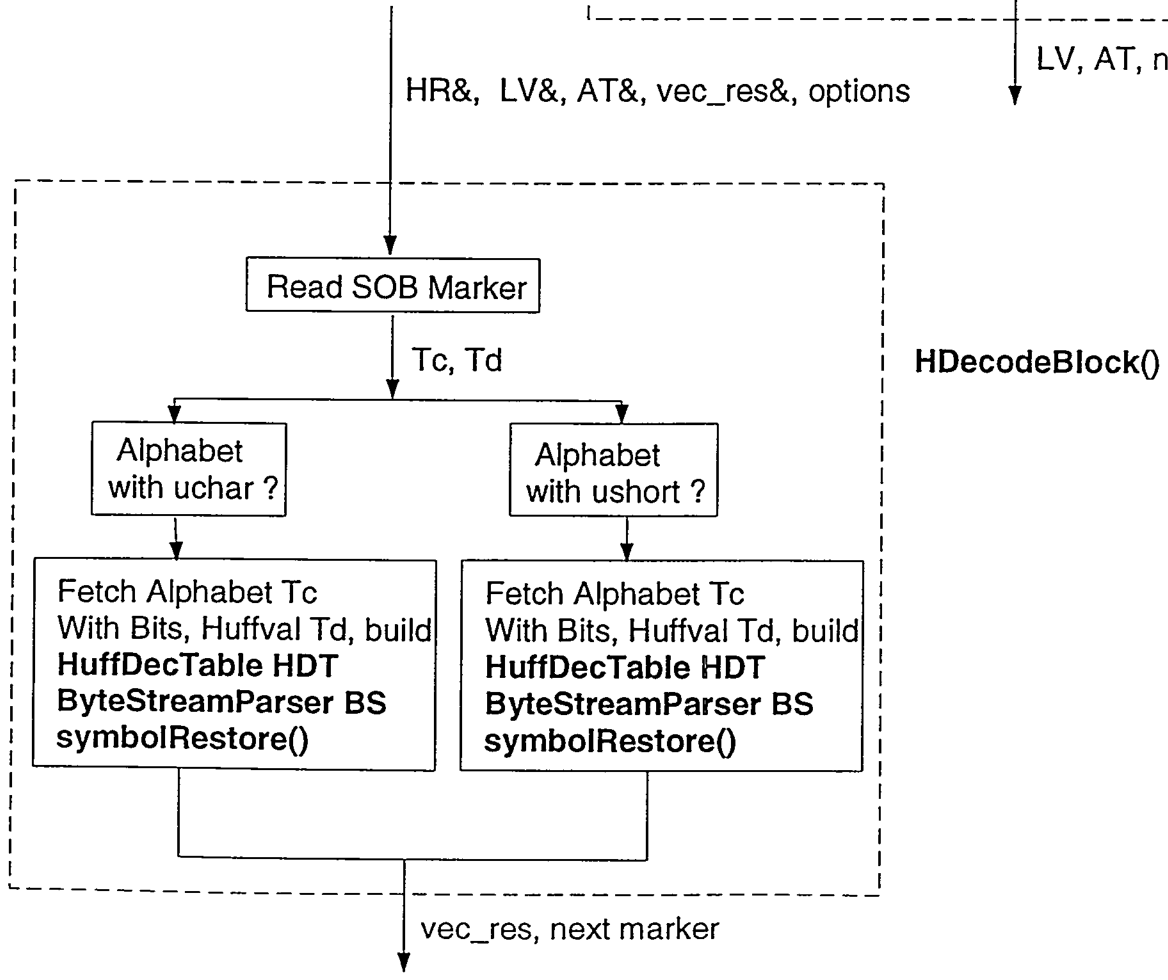


LV : vector<HTable>
AT : map<unsigned char, AlphTable*>
vec_res : vector<QUANT_OUT>

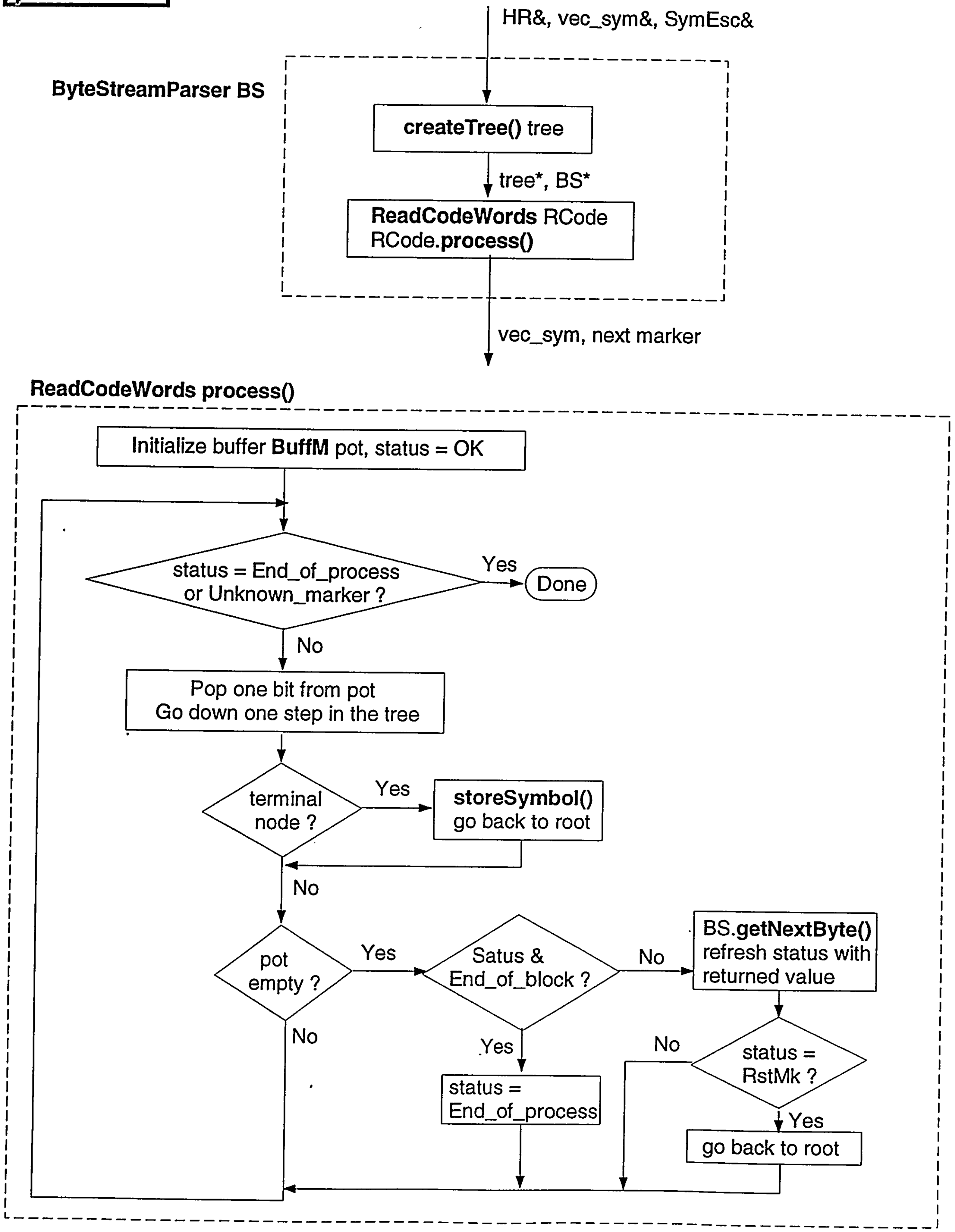
HParseFile

LV : vector<HTable>
AT : map<unsigned char, AlphTable*>
vec_res : vector<QUANT_OUT>





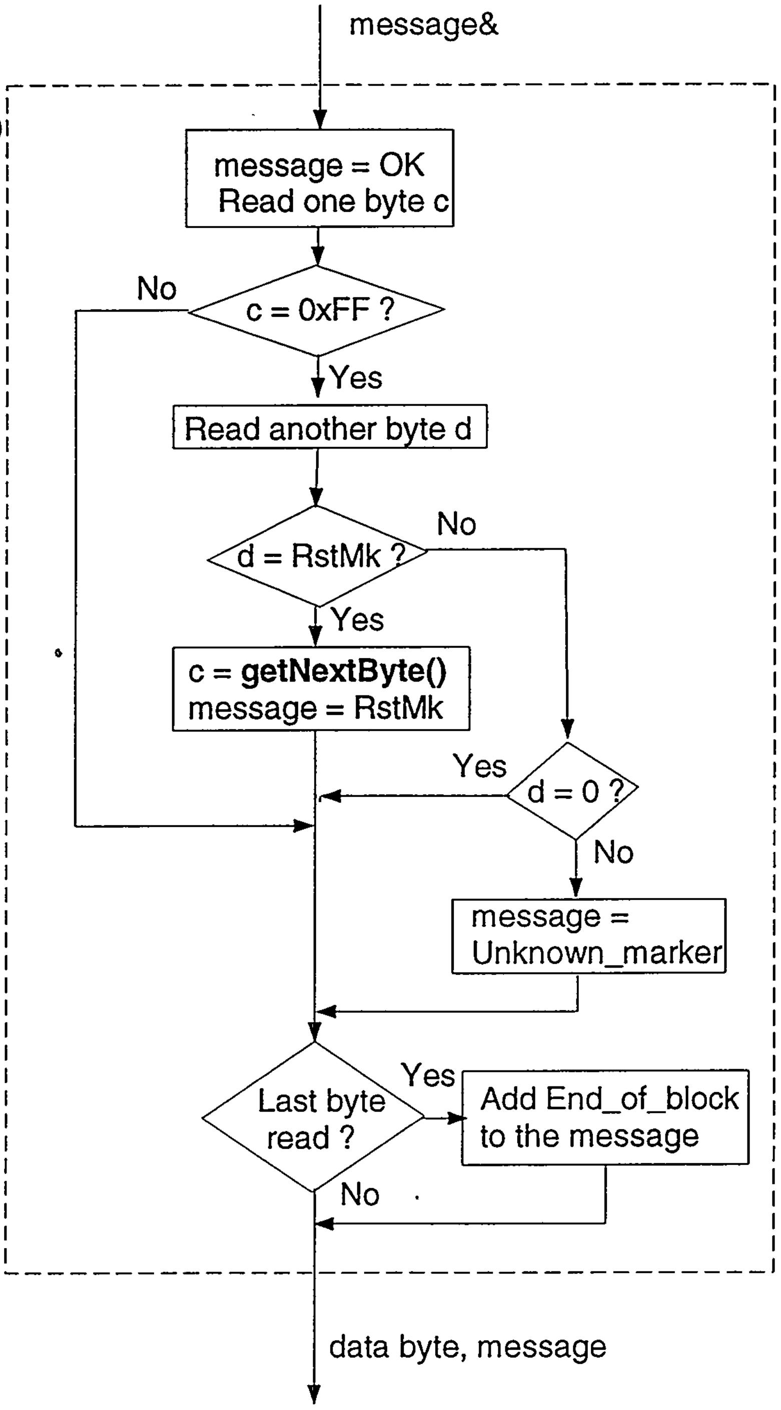
ByteParser



HR&, vec_sym&, SymEsc&
ByteStreamParser BS
createTree() tree
tree*, BS*
ReadCodeWords RCode
RCode.process()
vec_sym, next marker
ReadCodeWords process()
Initialize buffer BuffM pot, status = OK
status = End_of_process or Unknown_marker ?
Yes
Done
No
Pop one bit from pot
Go down one step in the tree
terminal node ?
Yes
storeSymbol()
go back to root
No
pot empty ?
Yes
Satus & End_of_block ?
No
BS.getNextByte()
refresh status with returned value
No
Yes
status = End_of_process
No
status = RstMk ?
Yes
go back to root

ByteParser (2)

ByteStreamParser getNextByte()



message&
message = OK
Read one byte c
c = 0xFF ?
No
Yes
Read another byte d
d = RstMk ?
No
Yes
c = getNextByte()
message = RstMk
d = 0 ?
Yes
No
message =
Unknown_marker
Last byte
read ?
Yes
Add End_of_block
to the message
No
data byte, message