1-1-2007

Simultaneous approximation of images applications to image and video compression

Ariel Juan Bernal
Ryerson University

SIMULTANEOUS APPROXIMATION OF IMAGES APPLICATIONS TO IMAGE AND VIDEO COMPRESSION

by

ARIEL JUAN BERNAL

BSc., CAECE University,
Mar del Plata, Argentina, 2003

A thesis presented to Ryerson University
in partial fulfillment of the requirement
for the degree of Master of Applied Science
in the Program of
Electrical and Computer Engineering.

Toronto, Ontario, Canada, 2007

Author's Declaration

I hereby declare that I am the sole author of this thesis.
I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.
Signature

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.
Signature

Instructions on Borrowers

Ryerson University requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

Title of Thesis:

SIMULTANEOUS APPROXIMATION OF IMAGES.
APPLICATIONS TO IMAGE AND VIDEO COMPRESSION.

Ariel J. Bernal, Master of Applied Science, 2007

Department of Electrical and Computer Engineering,
School of Graduate Studies, Ryerson University

Given a set of images we propose an algorithm that approximates all images simultaneously. The algorithm finds the best common partition of the images' domain at each step, this is accomplished by maximizing an appropriate inner product. The algorithm is a pursuit algorithm constrained to build a tree, the optimization is done over a large dictionary of wavelet-like functions. The approximations are given by vector valued discrete martingales that converge to the input set of images. Several computational and mathematical techniques are developed in order to encode the information needed for the reconstruction. Properties of the algorithm are illustrated through many examples, comparisons with JPEG2000 and MPEG4-3 are also provided.

Acknowledgments

I would like to express my gratitude to my supervisor, Dr. Sebastian Ferrando, who guided me along the way. Without his help and advice it would not have been possible to complete this project.
I would also like to thank my family for their constant support during my time at Ryerson University.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In image processing we usually make a distinction between two different aspects of images, the first group is called "trends" represented by areas of high statistical spatial correlation; and the second group called "anomalies" represented by edges or object boundaries. Although an image could take any state of all possible states, defined as all possible pixel combinations, human vision seems to have evolved to differentiate these two groups.

Normally, most of the image area is represented by the first group; however the perceptual significance of the second group is far greater than their numerical energy contribution. This effect can be explained in the same way as many others where the brain tends to capture changes and to eliminate constant states due to the fact that they do not contribute new information for the current state (e.g., a constant tone for ear, or in a cornea injury where rays or dots are constantly seen; after a period of time the brain partially eliminates those constant states).

The thesis describes a new algorithm for the simultaneous approximation of a given collection of images defined on a common domain Ω. The new algorithm is based on an optimized construction of basis functions adapted to arbitrary geometrical "anomalies" of this input set of images. A natural application of the algorithm is the case when the collection of images is given by a sequence of frames from a video. The algorithm constructs a tree which is associated to a partition of Ω (more precisely: a sequence of partitions). In several instances in this Introduction we may speak loosely and will not distinguish between the tree and the associated partition. Elements of a partition of Ω will be called *atoms*.

This thesis is based on an algorithm to approximate several images at the same time and a natural application is to image compression. Therefore, we describe some comparisons with existing algorithms in this field, the reader should keep in mind that we do not provide a thorough comparison as we have not developed our algorithm up to

the standards of a commercial software application. We provide a chapter describing a literature review for image compression techniques related to our algorithm. The present thesis describes an algorithm that has been studied previously in different contexts and with different aims in [14] and [2].

Several algorithms in the image compression field have been developed during the last twenty years, using different technics like the discrete cosine transform (DCT) or wavelets transform. References [22], [1], [9] and [19] provide examples of adaptive trees for image compression. In general, the tree construction is associated to a partition of the base domain which in turn is dependent on a given *single* input image. It follows that it is critical to keep the storage cost of the partition low as it adds to the total storage cost of the compressed image. Therefore, algorithms which partition a given image domain, with the purpose of compressing an image, need to impose strong geometrical constraints on the partition atoms. In particular, [1] only allows atoms which are polyhedra, further partitions of these atoms can only be done using line cuts. The reference [9] presents a global optimization algorithm which restricts the partition's atoms to be rectangles.

Analyzing these algorithms in terms of the partition cost it is possible to classify the Zero-tree algorithm [22] in one extremum, where there is no need to store a partition information, and our algorithm in the other extremum, where we have to store the partition completely. The geometric wavelets algorithm [1] is somewhere in between.

As an alternative to the above described situation, the approach introduced in this thesis allows for arbitrary partitioning of a given image domain and, hence, we deal with arbitrary atoms. In order to offset the relatively high cost of the resulting adapted partition we consider the case where we have a collection of d images, defined on a common domain Ω. This creates a trade-off as, on the one hand, the relative cost of storing the partition diminishes when we increase d and, on the other hand, the quality of the approximation degrades as d is increased.

The present thesis describes a construction (which we will call the Vector Greedy Splitting Algorithm, VGS for short) of an adapted partition of Ω, this partition is common to the given collection of images. Given a certain amount of similarity among the images in this collection our construction provides associated compression improvements when the common adapted partition is used to compress the collection of images as a single entity.

Despite the fact that the VGS algorithm deals with arbitrary geometrical regions it is a computational efficient algorithm. The reason for this is that the atoms processed by the algorithm are level sets of the input data and they can be efficiently manipulated with a computational cost proportional to the size of the range of values of the data. Details on computational costs are provided in Chapter 4.

Next we will describe the fundamental steps of the algorithm by using a simple

example: Figure 1.1 shows two images containing geometrical objects. Notice that

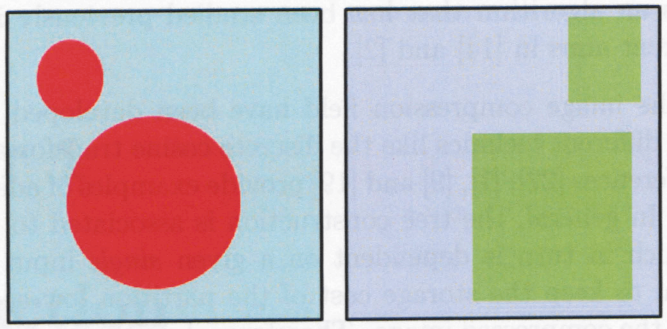


Figure 1.1: Two geometrical Images

there exists some common information between these images namely where both images are constat.

The algorithm starts constructing a partition based on this common information by maximizing the inner product with our wavelets functions [12]. Figure 1.2 shows the first partition in the top of the tree, it is possible to observe that both images are constant in the right child, therefore it should not be subdivided. In the left child the images are not constant then this child is subdivided again in two children, this process is continued (in the present case only a single extra step is needed) until both images are constant in the final partition.

Another important aspect of the algorithm is the tree structure that is possible to observe from the procedure. Finally the associated partition to this tree is also obtained. The number of atoms (or possible final values) in this partition is related to the possible combinations of the values taken by the original images. Usually the number of possible combinations increases with the number of different images and also as the complexity of the final partition increases, the number of coefficients needed to represent such combination increases too, this is related to the trade-off, mentioned above in this Introduction.

1.1 Organization of the Thesis

For the reader's convenience, Chapter 2 describes with technical details the main construction of the thesis for a special case (Haar case). We also take the opportunity to introduce some of the notation to be used in the remainder of the thesis. Chapter 3 presents the mathematical setup and describes the optimization problem solved in this thesis. This optimization is used to define the VGS algorithm. Chapter 3 also describes many possible variations in the setup and describes the different ways

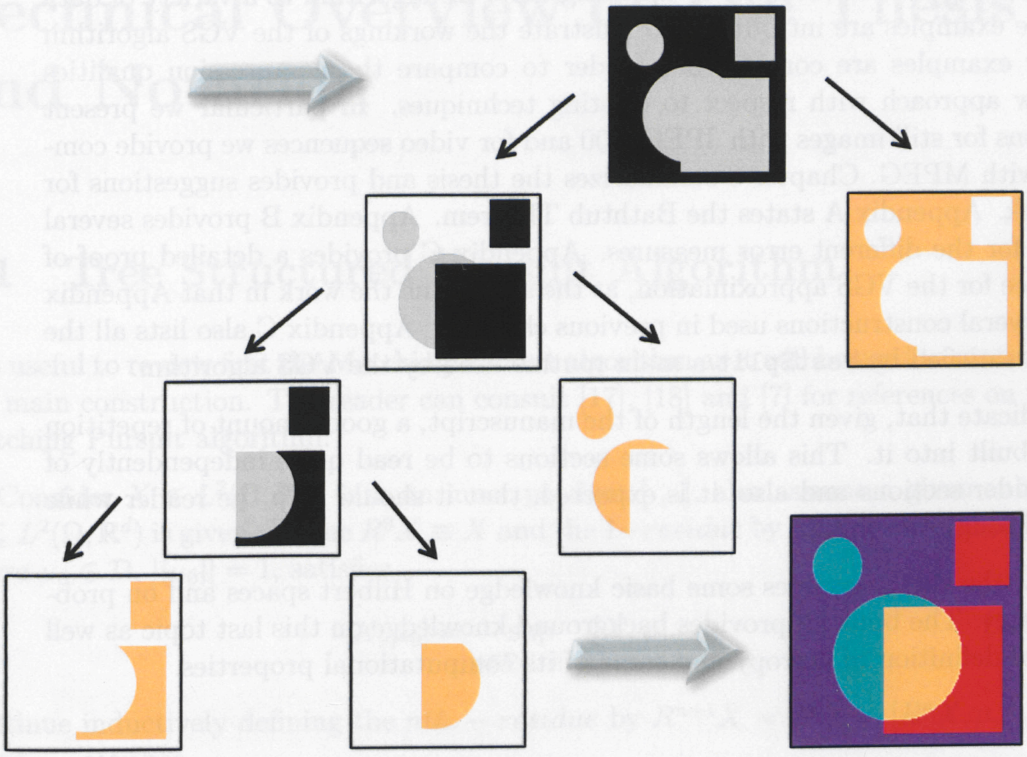


Figure 1.2: Approximating Tree

(vector and scalar) in which the VGS can be interpreted when it is being applied to image compression. Chapter 4 specializes the setup in Chapter 3 to a discrete setup and hence prepares the way for a software implementation. In fact, Chapter 4 does provide several details on the software implementation and describes the associated computational costs. Chapter 5 describes in detail the different steps to encode the VGS approximation in a way that a decoder can reconstruct the VGS approximation. Different quantization techniques are introduced and discussed in the context of the cases treated in this thesis (Haar case, Full Bathtub Case and Martingale Differences case). Chapter 6 describes some related techniques in the existing literature. Chapter 7 provides many results obtained by applying the VGS algorithm to a variety of data sets. Some examples are introduced to illustrate the workings of the VGS algorithm and other examples are considered in order to compare the compression qualities of the new approach with respect to existing techniques. In particular we present comparisons for still images with JPEG2000 and for video sequences we provide comparisons with MPEG. Chapter 8 summarizes the thesis and provides suggestions for future work. Appendix A states the Bathtub Theorem. Appendix B provides several equations for the different error measures. Appendix C provides a detailed proof of convergence for the VGS approximation, at the same time the work in that Appendix justifies several constructions used in previous chapters. Appendix C also lists all the properties satisfied by `bestSplit` a main routine used by the VGS algorithm.

We indicate that, given the length of the manuscript, a good amount of repetition has been built into it. This allows some sections to be read quite independently of the remainder sections and also, it is expected, that it should help the reader while working through the thesis.

Finally, the thesis assumes some basic knowledge on Hilbert spaces and on probability theory. The book [3] provides background knowledge on this last topic as well as a formal definition of entropy and some of its computational properties.

Chapter 2

Technical Overview of The Thesis and Notation

2.1 Tree Structured Pursuit Algorithm

It is useful to review first the Matching Pursuit algorithm as it will be used to motivate our main construction. The reader can consult [17], [18] and [7] for references on the Matching Pursuit algorithm.

Consider $X \in L^2(\Omega, \mathbb{R}^d)$ with an inner product $[\ ,\ ]$, also assume a given subset $\mathcal{D} \subseteq L^2(\Omega, \mathbb{R}^d)$ is given. Define $R^0X \equiv X$ and the $1-$*residue* by $R^1X = X - [X, \mu_0]\ \mu_0$ where $\mu_0 \in \mathcal{D}$, $||\mu_0|| = 1$, satisfies

$$[X, \mu_0] = \sup_{\psi \in \mathcal{D}, ||\psi||=1} [X, \psi]. \tag{2.1.1}$$

Continue inductively defining the $nth. - residue$ by $R^{n+1}X = R^nX - [R^nX, \mu_n]\ \mu_n$, where μ_n satisfies

$$[R^nX, \mu_n] = \sup_{\psi \in \mathcal{D}, ||\psi||=1} [R^nX, \psi]. \tag{2.1.2}$$

Notice that

$$X = \sum_{k=0}^{n} [R^kX, \mu_k]\mu_k + R^{n+1}X \tag{2.1.3}$$

and

$$||R^{n+1}X||^2 = ||R^nX||^2 - |[X, \mu_n]|^2.$$

Lossy transform compression is based on retaining a few terms in the first term of the right hand side of (2.1.3) (and dropping the remainder term $R^{n+1}X$). For the

dictionary $\mathcal{D}$ that will be considered in the present thesis the cost of storing each element μ_k in the pursuit expansion will be too high and for this reason we will impose a tree structure on the pursuit. This type of approach can be considered as a constrained non-linear approximation as described in [4]. As we will show later, one consequence of the tree structure is that $[\mu_i, \mu_j] = 0$ for $i \neq j$, this will imply

$$[R^n X, \mu_n] = [X, \mu_n].$$

Therefore, in our thesis, the pursuit algorithm can be described by indicating that it maximizes $|[X, \mu_n]|^2$, or equivalently, it minimizes $||R^n X||^2$, under the constraint of constructing a tree. The maximization in (2.1.2) is *greedy* because it is only one look ahead, namely it searches for one function at a time. In general, unless $\mathcal{D}$ has a special structure, it is expected that the maximization of $|[X, \mu_n]|^2$ requires an impractically large number of computations. A main contribution of the construction described in this thesis is a practical and insightful approach to handle (2.1.2) for a very large dictionary $\mathcal{D}$.

In this thesis we will use the following setup, consider a collection of d given images $X[i]$, $i = 1, \ldots, d$. We will treat them as random variables $X[i] : \Omega \to \mathbb{R}$ on a probability space $(\Omega, \mathcal{A}, P)$. We collect the d images into a vector valued random variable $X : \Omega \to \mathbb{R}^d$, and will consider the following inner product for $Y, Z \in L^2(\Omega, \mathbb{R}^d)$

$$[Y, Z] \equiv \int_\Omega \langle Y(w), Z(w) \rangle \, dP(w),$$

with $\langle Y(w), Z(w) \rangle = \sum_{i=1}^d Y[i](w) \, Z[i](w)$. Notice that in the case of $d = 1$ we have

$$[X, Y] = \int_\Omega X(w) Y(w) \, dP(w) \tag{2.1.4}$$

so the notation [,] does not indicate the value of d explicitly, therefore, readers will need to determine it from the context. In some cases, to emphasize the fact that $d = 1$, we will write (of course, in this case X and Y are scalar random variables) $[X, Y]_1$ to denote the right hand side of (C.1.2).

Consider the following dictionary of vector valued functions

$$\mathcal{C} = \{\psi : \text{ there exists } \; 0 \leq \varphi_i \leq 1, \quad i = 1, 2, \; a, b \in \mathbb{R}^d, \psi = a\varphi_1 + b\varphi_2, \tag{2.1.5}$$

$$\text{and } \int_\Omega \psi(w) \, dP(w) = 0, \int_\Omega ||\psi(w)||^2 \, dP(w) = 1\}.$$

A moments' reflection indicates this is a a very large collection of functions. If we expand X over this dictionary using a pursuit algorithm, we would obtain a normalized sequence $\{\mu_0, \mu_1, \ldots\} \subseteq \mathcal{C}$ in such a way that $||X - \sum_{k=0}^n [X, \mu_k] \, \mu_k|| \to 0$ as $n \to \infty$ whenever $\int_\Omega X(w) \, dP(w) = 0$. If the intent is to compress the information in X by

storing some of the μ_k functions, it follows that encoding each function μ_k can be too costly.

To avoid this difficulty we define below a pursuit algorithm constrained by a tree structure, storing the tree information will allow us to reproduce the approximating functions μ_k in an efficient way.

First, we refine (2.1.5) as follows, consider $A \in \mathcal{A}$ and

$$\mathcal{C}_A = \{\psi : \text{ there exists } 0 \leq \varphi_1 \leq 1, \quad \varphi_1(w) = 0 \text{ if } w \notin A, \ \varphi_2 \equiv 1_A - \varphi_1, \ a, b \in \mathbb{R}^d, \tag{2.1.6}$$

$$\psi = a\varphi_1 + b\varphi_2, \text{ and } \int_\Omega \psi(w)\, dP(w) = 0, \int_\Omega ||\psi(w)||^2\, dP(w) = 1\}.$$

Let $u_i \equiv \int_\Omega \varphi_i \, dP$, we will further restrict the elements $\psi \in \mathcal{C}_A$ even more by assuming $||b|| = ||b||(u_1, u_2)$. The resulting restricted dictionary will be denoted by $\hat{\mathcal{C}}_A$.

We indicate that we can solve the following key step to define the pursuit algorithm.

Proposition 1. *Under general conditions on X, there exists $\psi_A^{(0)} \in \hat{\mathcal{C}}_A$ so that*

$$[X, \psi_A^{(0)}] = \sup_{\psi \in \hat{\mathcal{C}}_A} [X, \psi],$$

and the functions $\psi_A^{(0)}$ take only a finite number of distinct values. Moreover, under the hypothesis that X has a continuous cumulative distribution the functions $\psi_A^{(0)}$ take only two non-zero different values on A.

Remark 1. *The functions $\psi_A^{(0)}$ will be called best functions (at A) their explicit form will be given later. In this chapter, the notation $\psi_A^{(0)}$ will be reserved for the functions in Proposition 1. The use of the superscript in $\psi_A^{(0)}$ will be needed later in the thesis, during the present Chapter it can be ignored.*

Remark 2. *For simplicity, on this Chapter only, we will assume X has a continuous cumulative distribution (this will give us what later in this thesis we will call the Haar case). In later chapters of the thesis we will deal with the general case namely, the continuity hypothesis will be removed and φ_2 will be required only to satisfy $0 \leq \varphi_2 \leq 1$.*

Instead of starting with (2.1.1), our vector approximations are always initialized as follows

$$\mu_0 = \psi_\emptyset^{(0)} \equiv c\, 1_\Omega$$

and

$$c[i] = \frac{\int_\Omega X[i]\, dP}{\sqrt{\sum_{k=1}^{d} \left(\int_\Omega X[k]\, dP \right)^2}}.$$

Therefore

$$[X, \psi_\emptyset^{(0)}]\ \psi_\emptyset^{(0)}[i] = \int_\Omega X[i]\, dP. \tag{2.1.7}$$

We need to introduce the following notation. For the range of a random variable Y (scalar valued) on the set A

$$\mathcal{R}_A(Y) \equiv \{y : \exists w \in A \ \text{ such that } \ Y(w) = y\}.$$

As noted, the functions $\psi_A^{(0)}$ take only two values for a given A; in order to completely specify $\psi_A^{(0)} = a\ \varphi_1 + b\ (1_A - \varphi_1)$ ($\psi_A^{(0)}$ as given in Proposition 1) it is enough to provide φ_1 explicitly.

$$\varphi_1(w) = 1_{\{z : X[b'](z) \le y\}}(w) \text{ where}$$

$$X[b'](z) \equiv \langle X(z), b' \rangle \ \text{ for some } \ b' \in S^d \ \text{ and some } \ y \in \mathcal{R}_A(X[b']),$$

moreover $b = ||b||\ b'$ (the expression for $||b||$ is provided in (3.4.12)). The quantities b' and $y = y(b')$ are obtained by an optimization procedure as described in [2] (and also explained later in this thesis). $X[b']$ can be considered as an average image as illustrated in Figure 2.1.

Notice that the two different (non-zero) values taken by $\psi_A^{(0)}$ on A are a and b. We will then define its *best children* by

$$A_0 \equiv \{w \in A : \psi_A^{(0)}(w) = a\}, \quad A_1 \equiv \{w \in A : \psi_A^{(0)}(w) = b\}.$$

Next we define the VGS algorithm, we will indicate how the algorithm constructs a sequence of partitions Π_n indexed by $n = 0, 1, 2, \ldots$. The index n will be referred as the $n-$th. iteration of VGS. The partitions are defined recursively as indicated next. Start by setting $\Pi_0 = \{\Omega, \emptyset\}$ (notice that we explicitly include $\emptyset$ in Π_0, this will include $\psi_\emptyset^{(0)}$ in all our approximations) and assume, inductively, that Π_k, $k \le n$ ($\Pi_k \subseteq \mathcal{A}$) have been constructed and are finite. Now we describe how to generate Π_{n+1}. Consider $A^* \in \Pi_n$ such that it satisfies

$$|[X, \psi_{A^*}^{(0)}]| \ge |[X, \psi_A^{(0)}]| \ \text{ for all } \ A \in \Pi_n. \tag{2.1.8}$$

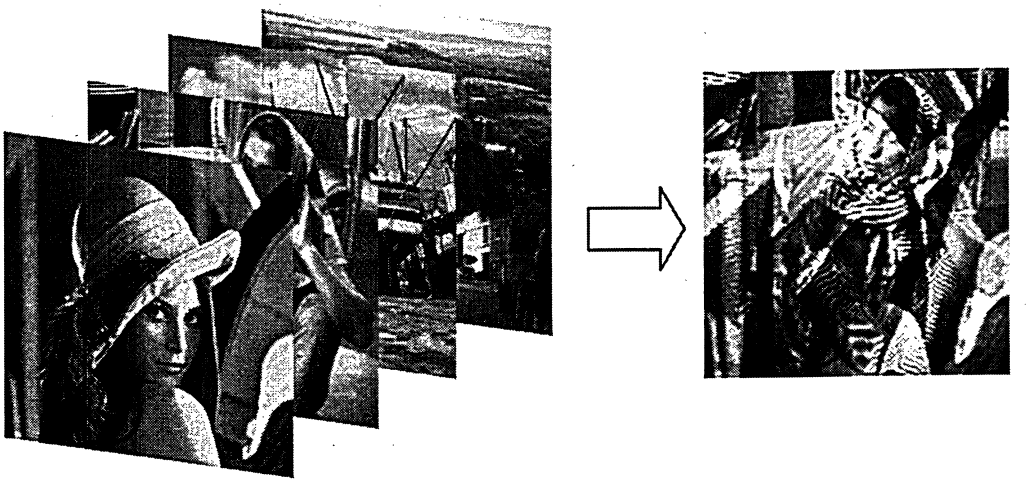


Figure 2.1: Left: set of input images, Right: Linear combination

In the case that

$$[X, \psi_{A^*}^{(0)}] = 0,$$

the algorithm VGS terminates and $\Pi_p \equiv \Pi_n$ for all $p \geq n$. Otherwise, i.e. $[X, \psi_{A^*}^{(0)}] \neq 0$, we set

$$\Pi_{n+1} = \Pi_n \backslash \{A^*\} \cup_{k=0,1} \{A_k^*\}$$

where, as indicate previously, the sets A_k^* are the best children of A^*.

Define the tree $\mathcal{T}_n$ as follows

$$\mathcal{T}_n \equiv \cup_{k=0}^{n} \Pi_k.$$

It follows easily from the construction (shown formally in later chapters) that $[\psi_{A_1}^{(0)}, \psi_{A_2}^{(0)}]$ if $A_1 \neq A_2$, $A_i \in \mathcal{T}_n$.

We then define the associated approximation by

$$X_{\mathcal{T}_n} \equiv \sum_{A \in \mathcal{T}_n} [X, \psi_A^{(0)}] \, \psi_A^{(0)}. \tag{2.1.9}$$

Requiring the appropriate conditions, it can be proven that

$$\lim_{n \to \infty} X_{\mathcal{T}_n}(w) = X(w) \quad \text{for almost all} \quad w \in \Omega.$$

The above limit will actually be finite if X is a simple function, namely if X takes a finite number of values. Full details are provided in Appendix C.

Returning to the expansion (2.1.9), once the tree is built, it is possible to insert nodes into a priority queue (which will be denoted with $\mathcal{Q}_\Lambda$ and introduced later in

this thesis) and order the queue by the absolute value of the inner products $[X, \psi_A^{(0)}]$. This allows to set to zero the smallest nodes and reconstruct the original signal allowing some distortion, of course this method will only provide a lossy compression algorithm.

Setting to zero the smallest inner products is equivalent to calculate the n-term approximation, this is accomplished by re-labeling the elements $\psi_A^{(0)}$ in the expansion (2.1.9) and denote them with $\mu_{h(k)}$ (where h is a re-ordering function) in such a way that:

$$X_n(w) = \sum_{k=0}^{n-1} [X, \mu_{h(k)}]\ \mu_{h(k)} \quad \text{where} \quad |[X, \mu_{h(0)}]| \geq |[X, \mu_{h(1)}]| \geq \dots \tag{2.1.10}$$

In this thesis we will refer many times to the *VGS approximation*, most of the times this will mean the expression given by (2.1.9) but, depending of the context it may as well refer to expression (2.1.10). For the sake of precision, one may call (2.1.10) the *optimized VGS approximation.*

The error, or distortion, committed by the n-term approximation can be calculated using different methods, one is the Mean Square Error (MSE) defined as follows

$$\text{MSE}_n = \int_\Omega [X(w) - X_n(w)]^2\ dP(w).$$

In the image compression field the standard measure of error used is the Peak Noise to Signal Ratio (PSNR). It is not the best method due to the fact that to images with a low PSNR could be close with respect to the human vision system. Although several techniques were developed in this area, in general there is no perfect method to measure the distortion of an approximating image.

2.2 Bit Encoding of the VGS Approximation

We will, given a certain error level $\epsilon > 0$, define the associated approximation $X_n = \sum_{k=0}^{n-1} [X, \mu_{h(k)}]\ \mu_{h(k)}$, where $n = n(\epsilon)$ is the smallest integer such that $||X - X_n|| \leq \epsilon$. The corresponding n nodes A_k which satisfy $\psi_{A_k}^{(0)} = \mu_{h(k)}$ are called the *active* nodes for the given ϵ.

Notice that retaining only the active nodes from the full VGS tree (this process is called *pruning the tree*) results into a data structure which is not a tree. Therefore, in order to reconstruct X_n, while storing a minimal amount of bits, we need to store the information about which nodes are active and at each of these nodes A_k we need to keep $[X, \psi_{A_k}^{(0)}]$ and the corresponding b'. We also need to keep enough of the tree

information in order to evaluate $\psi^{(0)}_{A_k}(w)$ at different points $w \in \Omega$. In particular, this information will contain the relevant children-parent relationship. This information will be called the *significance map* and denoted with $\mathcal{M}_S$. Its actual encoding is technically challenging as our approach only deals with the actives nodes (i.e. we do not complete with the missing nodes in order to obtain a tree). Besides of the information contained in the significance map, we also need to know how active atoms are made up of Ω points. This information will be called the *partition map*, and denoted $\mathcal{M}_\Pi$, and consists on encoding the partition associated to the active nodes only.

In this Chapter, when reporting bit values of the significance map we will be actually reporting the bit cost of encoding quantized values $[X, \psi^{(0)}_{A_k}]$ and the corresponding quantized values for b'. When we report bit values of the partition map we will be reporting the bit cost of encoding a lossless compressed version of the corresponding data structure (we have found that the partition map is very sensitive to quantization).

In a more detailed analysis, which is performed in Chapter 5, a third map, the *quantization map* $\mathcal{M}_Q$, will be introduced. Therefore, in effect, in the present Chapter the bit cost of $\mathcal{M}_S$ includes the bit cost of $\mathcal{M}_Q$.

We need some notation that indicates that we have run VGS on d inputs, so $C_{\mathcal{M}_S}[i](d)$ will mean that we have run VGS for d inputs and the component i has a significance cost of $C_{\mathcal{M}_S}[i](d)$ bits. Whenever $d = 1$ we will write $C_{\mathcal{M}_S}[1](1)$ as $C_{\mathcal{M}_S}(1)$. In short, $C_{\mathcal{M}_S}(1)$ represents the (quantized) significance map cost of encoding the output of VGS (*excluding* the partition cost) and VGS was executed on a single image. We use similar notation for the partition map cost but we will assume the partition cost is independent of i. Therefore the notation $C_{\mathcal{M}_\Pi}(d)$ denotes the number of bits needed to store the partition map when VGS was executed on d images.

We expect $C_{\mathcal{M}_S}[i](d)$ to deteriorate as d increases (for any i), and we also expect $C_{\mathcal{M}_S}(1)$ to be of best quality, i.e. $C_{\mathcal{M}_S}(1) << C_{\mathcal{M}_S}[i](d)$ for all i and d. We also note that $C_{\mathcal{M}_\Pi}(d)$ has a uniform upper bound (i.e. the upper bound is independent of d) which depends solely on the size of Ω.

Let us use $C_{FixedBasis}$ to denote the cost of encoding a given image by a certain method with fixed basis (in particular it could be JPEG, JPEG2000, Haar basis, etc.). If there are d images we will consider that $C_{FixedBasis}[i]$ denotes the cost, of the method, for image i. We expect that $C_{\mathcal{M}_S}(1) << C_{FixedBasis}[1]$.

We introduce next a useful quantity to quantify the quality of VGS's approxima-

tion

$$\gamma(d) \equiv \frac{C_{\mathcal{M}_\Pi}(d)}{d} + \frac{\sum_{i=1}^{d} C_{\mathcal{M}_S}[i](d)}{d}. \tag{2.2.1}$$

Clearly, the optimal $d^\star$ is the one that minimizes $\gamma(d)$. It is clear that there is a tension between how large d has to be so $\frac{C_{\mathcal{M}_\Pi}(d)}{d}$ is small enough and at the same time we want $\frac{\sum_{i=1}^{d} C_{\mathcal{M}_S}[i](d)}{d}$ to remain small but we know that $C_{\mathcal{M}_S}[i](d)$ deteriorates as d growths.

Notice that VGS will outperform the cost of the fixed basis method, namely $C_{FixedBasis}$ if

$$\gamma(d) < \frac{\sum_{i=1}^{d} C_{FixedBasis}[i]}{d}.$$

As an illustration, Figure 2.2 shows a set of images; technical characteristics on this set of images are discussed in Chapter 7. We have run VGS on increasingly larger subsets of this collection of images by adding one image at a time to the previous subset. In this way we were able to compute $\gamma(d)$ for $d = 1, \ldots, 9$. The results are plotted in Figure 2.3, the term $\frac{C_{\mathcal{M}_\Pi}(d)}{d}$ in (2.2.1) (average cost for Partition Map) is denoted PM (Partition Map) in the Figure, and the term $\frac{\sum_{i=1}^{d} C_{\mathcal{M}_S}[i](d)}{d}$ in (2.2.1) (average total cost for Quantization Map and Significance Map) is denoted by $QM + SM$.

As another illustration, Figure 2.2 shows a set of images taken from a video sequence; technical characteristics on this set of images are discussed in Chapter 7. We have run VGS on increasingly larger subsets of this collection of images by adding one image at a time to the previous subset. In this way we were able to compute $\gamma(d)$ for $d = 1, \ldots, 20$. The results are plotted in Figure 2.5, the same explanations supplied for the previous Figure also apply to the present Figure.

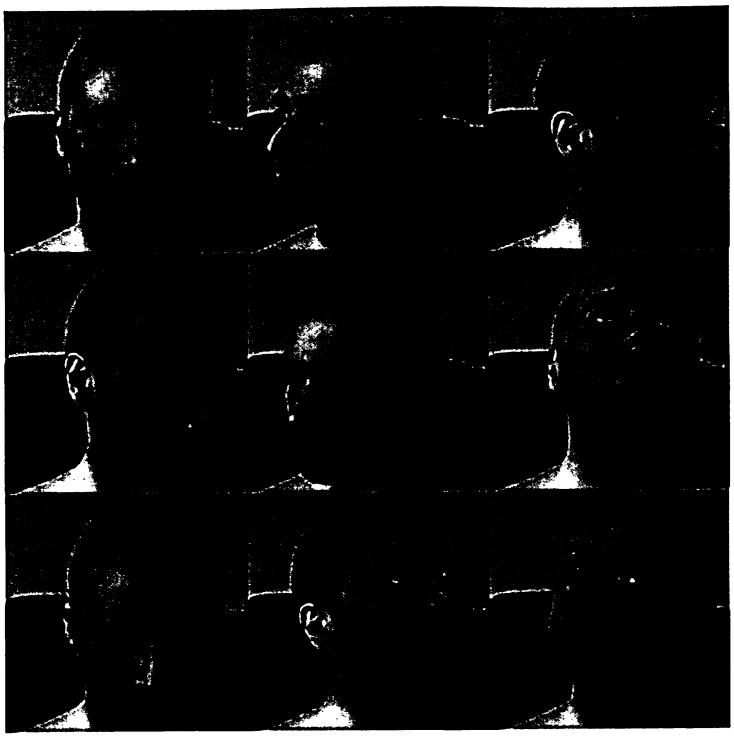


Figure 2.2: Faces set

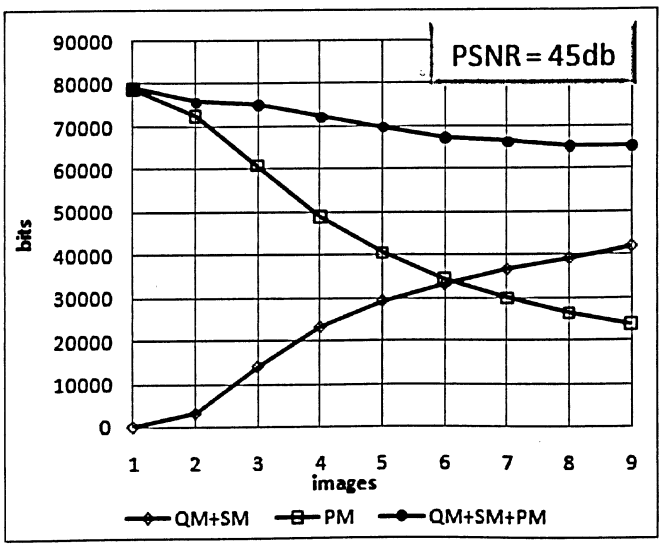


Figure 2.3: Average $C_{\mathcal{M}_\Pi}$ and average $C_{\mathcal{M}_Q} + C_{\mathcal{M}_S}$ plotted against d.

Figure 2.4: Video sequence set

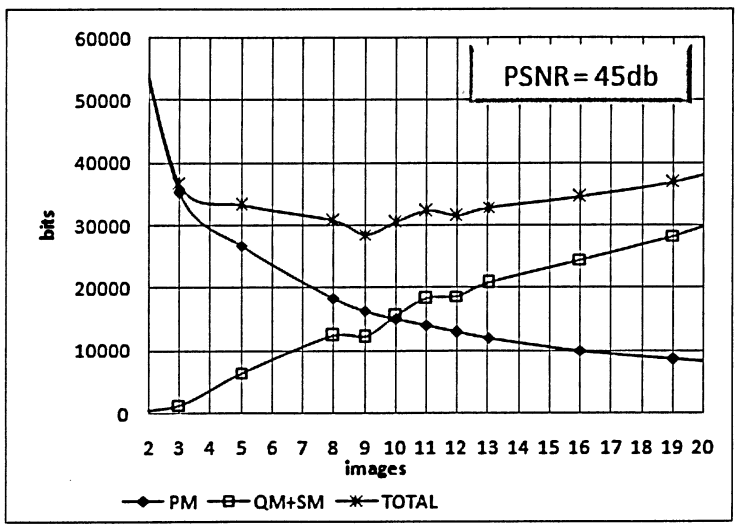


Figure 2.5: Average $C_{\mathcal{M}_\Pi}$ and average $C_{\mathcal{M}_Q} + C_{\mathcal{M}_S}$ plotted against d.

Chapter 3

Mathematical Framework

This chapter contains the fundamental framework needed to understand the following chapters. It introduces the general notation and the description of the algorithm from a mathematical point of view.

Motivated by the Matching Pursuit algorithm [18] the VGS algorithm is described in terms of maximizing an inner product under the constraint of constructing a tree. Using an interesting dictionary (given by the VGS functions) and under general hypothesis, the problem can be solved using the Bathtub principle, which plays an important role in the development of the algorithm. Different setup variations applied to the general case provide the special cases considered in the thesis (many more cases are also possible but not developed in the thesis). We label these cases as follows: Full Bathtub case, Haar case and Martingale Differences case. These cases are analyzed in this Chapter after we complete the general description of the VGS algorithm.

3.1 General Notation and Definitions

Given a set of inputs signals, we consider each such a set as a vector valued random variable in a Hilbert space $L^2(\Omega, \mathbb{R}^d)$ associated to the probability space $(\Omega, \mathcal{A}, P)$. $\mathcal{A}$ is a given σ-algebra and in the case when Ω is finite, we may take $\mathcal{A}$ to be the collection of all subsets of Ω namely $\mathcal{A} = \mathcal{P}(\Omega)$ where $\mathcal{P}$ is the power set of Ω. The only requirement is that $\sigma(X) \subseteq \mathcal{A}$. Elements from $L^2(\Omega, \mathbb{R}^d)$ are vector valued random variables $X : \Omega \to \mathbb{R}^d$, $X(w) = (X_1(w), \ldots, X_d(w))$, the components X_i will be the given input signals. From now on, in order to avoid confusions with the use of subscripts, instead of using X_i to describe the i-th scalar component of the vector X, we will use $X[i]$.

The inner product in $L^2(\Omega, \mathbb{R}^d)$, for two vector valued random variables X and Y, is given by

$$[X, Y] \equiv \int_\Omega \langle X(w), Y(w) \rangle \, dP(w),$$

where $\langle \ , \ \rangle$ is the Euclidean inner product in $\mathbb{R}^d$, defined by,

$$\langle X(w), Y(w) \rangle = \sum_{i=1}^{d} X[i](w) \; Y[i](w).$$

Notice that for any constant vector $b \in \mathbb{R}^d$, any $A \subset \Omega$ and any vector valued (i.e. any $\mathbb{R}^d$-valued function) random variable X, the following equality holds

$$\langle b, \int_\Omega X(w) \, dP(w) \rangle = \int_\Omega \langle b, X(w) \rangle \, dP(w),$$

where the left hand side is computationally more efficient.

Also we will use the following notation for the characteristic functions:

$$\mathbf{1}_A(w) = \begin{cases} 1 & if \ w \in A \\ 0 & otherwise. \end{cases}$$

This definition leads immediately to two main properties of characteristic functions:

$$\mathbf{1}_A \cdot \mathbf{1}_A = \mathbf{1}_A,$$

$$\mathbf{1}_A \cdot \mathbf{1}_B = 0, \quad \text{if} \quad A \cap B = \emptyset.$$

Definition 1. *A set A is called an "event" if $A \in \mathcal{A}$. In special cases, introduced later, events will be called atoms.*

Definition 2. *A collection of events $\Pi(A) = \{A_1, \ldots, A_n\}$ where $A_i \in \mathcal{A}$ and $i = 1, \ldots, n$ is called a finite "Partition" of A if satisfies*

$$\bigcup_{i=1}^{n} A_i = A \quad where \quad n \in \mathbb{N}$$

and

$$A_i \cap A_j = \emptyset \quad for \ all \quad i \neq j.$$

Definition 3. *Given $A \in \mathcal{A}$, $P(A) > 0$, a function ψ is called a (vector valued) "Haar function" on A if there exist $A_0, A_1 \in \mathcal{A}$, $A_0, A_1 \subseteq A$, $A_0 \cap A_1 = \emptyset$, $A = A_0 \cup A_1$, $\psi = a \ \mathbf{1}_{A_0} + b \ \mathbf{1}_{A_1}$, where $a, b \in \mathbb{R}^d$ and*

$$\mathbf{E}(\psi) = \int_\Omega \psi(w) \, dP(w) = 0, \tag{3.1.1}$$

and

$$\|\psi\|^2 \equiv [\psi, \psi]^2 = \int_\Omega \langle \psi(w), \psi(w) \rangle \, dP(w) = 1. \tag{3.1.2}$$

Using the last definition we can find the values for a and b, actually (3.1.1) gives

$$a = \frac{-b\ P(A_1)}{P(A_0)} \tag{3.1.3}$$

and replacing (3.1.3) in (3.1.2) gives

$$\|b\| = \sqrt{\frac{P(A_0)}{P(A)\ P(A_1)}} \tag{3.1.4}$$

also, if $b' \equiv \dfrac{b}{\|b\|}$ we obtain

$$b = b' \sqrt{\frac{P(A_0)}{P(A)\ P(A_1)}} \tag{3.1.5}$$

where $b' \in \mathbb{R}^d$ and $\|b'\| = 1$ then b' belongs to the d-dimensional sphere S^d defined by

$$S^d = \left\{ x = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d : \|x\|^2 = \sum_{i=1}^{d} x_i^2 = 1 \right\}. \tag{3.1.6}$$

A more general class of functions than the Haar functions can be defined as follows

Definition 4. *A function $\psi_A : \Omega \mapsto \mathbb{R}^d$ is called a "VGS function" if for a given $A \in \mathcal{A}$ the following conditions are satisfied*

$$\psi_A(w) = a\ \varphi_1(w) + b\ \varphi_2(w) \quad \forall w \in \Omega, \tag{3.1.7}$$

where $a, b \in \mathbb{R}^d$ and $\varphi_i : \Omega \mapsto \mathbb{R}$ defined by

$$0 \leq \varphi_i(w) \leq 1 \quad \forall\ w \in A, \quad i = 1, 2 \tag{3.1.8}$$

and

$$\varphi_i(w) = 0 \quad \forall\ w \notin A, \quad i = 1, 2 \tag{3.1.9}$$

also, we require

$$\int_\Omega \psi_A(w)\ dP(w) = 0 \tag{3.1.10}$$

$$\int_\Omega \|\psi_A(w)\|^2\ dP(w) = 1. \tag{3.1.11}$$

Notice that the two equations (3.1.10) and (3.1.11) are incompatible if $\varphi_1 = \varphi_2$ as this equality gives $\psi = 0$. Equations (3.1.10) and (3.1.11) will implicitly rule out $\varphi_1 = \varphi_2$.

Remark 3. *For the special case $d = 1$, ψ_A will be called a scalar VGS function.*

Remark 4. *Notice that as φ_1 and φ_2 are zero outside A then $\psi_A(w) = 0$ if $w \notin A$. Also whenever A is understood we will avoid the use of the subscript by writing ψ instead ψ_A.*

Definition 5. *Let $\mathcal{C}_A \subset L^2$ be the collection of all possible VGS functions on $A \in \mathcal{A}$ then*

$$\mathcal{C}_A = \{\psi : \psi \text{ satisfies (3.1.7) to (3.1.11)}\}.$$

Remark 5. *We assume that there exist a routine called* `bestSplit` *such that for a given $A \in \mathcal{A}$ provides a finite number of best functions $\psi_A^{(i)} \in \mathcal{C}_A$, $i = 0, \ldots, I_A - 1$, and a partition of A into a finite number of best children $A_k \in \mathcal{A}$, $k = 0, \ldots, I_A$. We will also require*

$$[\psi_A^{(i)}, \psi_A^{(j)}] = 0 \text{ whenever } i \neq j.$$

Actually, $\psi_A^{(0)}$ will be the function that optimizes the inner product and the remaining functions $\psi_A^{(i)}$ are constructed to be orthonormal to $\psi_A^{(0)}$. Also $\psi_A^{(i)}(w) = 0$ if $w \notin A$

We refer the reader to the end of Appendix C for a complete listing of all the properties satisfied by `bestSplit` and an indication how `bestSplit` is actually constructed in this thesis. The key step is given by the construction of $\psi_A^{(0)}$, the rest of the needed constructions follow from it. $\psi_A^{(0)}$ is characterized in (3.3.13) and the different cases leading towards $\psi_A^{(0)}$ are detailed in the remainder of the present Chapter. Further details are also provided in Chapter 4.

Remark 6. *We will always have $I_\emptyset = 1$ and will use the notation $\psi_\emptyset = \psi_\emptyset^{(0)}$.*

Remark 7. *We also assume that the functions $\psi_A^{(i)}$ take a finite number of values. For the three cases considered in the thesis the functions $\psi_A^{(i)}$ take exactly $I_A + 1$ different non-zero values. Set*

$$\mathcal{R}_A(\psi^{(0)}) = \{r_0, r_1, \ldots, r_{I_A}\}$$

to be the range of values taken by $\psi_A^{(0)}(w)$ when evaluated at points $w \in A$. Then the best children are defined by

$$A_i \equiv \{w \in A : \psi^{(0)}(w) = r_i\}. \tag{3.1.12}$$

In short, best children are given as pre-images of constant values of $\psi^{(0)}$. We will also require that the same children are obtained as pre-images of constant values of $\psi^{(i)}$ for $i > 0$; details are provided in (C.3.3)

Given the above notions, we will say that A *splits* into its best children A_i.

3.2 Formal Description of the VGS Algorithm

The VGS algorithm, builds a sequence of partitions Π_n on Ω indexed by $n = 1, 2, ...$; this index will be referred as the n-th iteration of the VGS algorithm. The partitions are defined recursively:

- Let $\Pi_0 = \{\Omega, \emptyset\}$.
- Assuming that Π_n has been created, then Π_{n+1} is generated as follows: Consider $A^* \in \Pi_n$ such that it satisfies

$$||[X, \psi_{A^*}^{(0)}]|| \geq ||[X, \psi_A^{(0)}]|| \quad \text{for all} \quad A \in \Pi_n. \tag{3.2.1}$$

Now, if

$$[X, \psi_{A^*}^{(0)}] = 0,$$

the algorithm VGS terminates and $\Pi_p \equiv \Pi_n$ for all $p \geq n$. Otherwise, i.e. $[X, \psi_{A^*}] \neq 0$, we set

$$\Pi_{n+1} = \Pi_n \backslash \{A^*\} \bigcup_{i=0}^{I_A} \{A_i^*\}$$

where, as indicated previously (3.1.12), the sets A_i^* are the best children of A^*.

Remark 8. *Events, i.e. elements from $\mathcal{A}$, that belong to any of the partitions Π_n constructed by the VGS algorithm, will be called atoms.*

The VGS algorithm builds a tree $\mathcal{T}$ where its nodes are atoms from the partitions Π_n. The formal definition is given by:

$$\mathcal{T} = \bigcup_{n=0}^{\infty} \Pi_n, \tag{3.2.2}$$

also we can define the n-term tree as

$$\mathcal{T}_n = \bigcup_{i=0}^{n} \Pi_i.$$

The parent-children relationship is given by the *split* relationship mentioned in the previous Section.

We will define an increasing sequence of orthonormal systems $\mathcal{H}_n$, for $n \geq 0$ corresponding to the n-th. iteration of the VGS algorithm, as follows: $\mathcal{H}_0 \equiv \{\mu_0 \equiv \psi_\emptyset\}$ also, assume, recursively that $\mathcal{H}_n = \{\mu_0, \ldots, \mu_{k_n}\}$ has been constructed. We then let,

$$\mathcal{H}_{n+1} \equiv \mathcal{H}_n \bigcup_{i=0}^{I_A-1} \{\psi_{A^*}^{(i)}\}$$

where A^* is the set in (3.2.1), also set $\mu_{k_n+i+1} \equiv \psi_{A^*}^{(i)}$ for $i = 0, \ldots, I_A - 1$. We also set

$$\mathcal{H} \equiv \bigcup_{n \geq 0} \mathcal{H}_n. \tag{3.2.3}$$

The following basic Proposition proves that $\mathcal{H}$ is an orthonormal system.

Proposition 2. *The set of VGS functions $\mathcal{H}$ built by the VGS algorithm is an orthonormal system.*

Proof. In order to prove the theorem the following equation should be satisfied

$$[\psi_{A_i}^{(k)}, \psi_{A_j}^{(l)}] = \begin{cases} 0, & \text{if } i \neq j \text{ or } k \neq l; \\ 1, & \text{if } i = j \text{ and } k = l;. \end{cases}$$

It is possible to observe that by definition we require that $\|\psi_{A_i}^{(k)}\|^2 = [\psi_{A_i}^{(k)}, \psi_{A_i}^{(k)}] = 1$. Also by the constraint imposed to $\psi_{A_i}^{(k)}$ with $k > 0$, $[\psi_{A_i}^{(k)}, \psi_{A_i}^{(l)}] = 0$ for all $k \neq l \in \mathbb{N}$.

The next step is to check the orthogonality condition.

It remains only to check the orthogonality condition for the case $i \neq j$.

1. $\psi_{A_i}^{(k)}$ is not a descendant or an ancestor of $\psi_{A_j}^{(l)}$, this means that they are defined in two disjoint atoms, and as $\psi_{A_i}^{(k)}(w) = 0$ if $w \notin A_i$ then the inner product

$$\begin{aligned} [\psi_{A_i}^{(k)}, \psi_{A_j}^{(l)}] &= \int_\Omega \langle \psi_{A_i}^{(k)}(w), \psi_{A_j}^{(l)}(w) \rangle \, dP(w) \\ &= \int_{A_i} \langle \psi_i^{(k)}(w), \psi_{A_j}^{(l)}(w) \rangle \, dP(w) + \int_{A_j} \langle \psi_{A_i}^{(k)}(w), \psi_{A_j}^{(l)}(w) \rangle \, dP(w) \\ &= \int_{A_i} \langle \psi_{A_i}^{(k)}(w), 0 \rangle \, dP(w) + \int_{A_j} \langle 0, \psi_{A_j}^{(l)}(w) \rangle \, dP(w) \\ &= 0 \end{aligned}$$

2. if $\psi_{A_i}^{(k)}$ is an ancestor of $\psi_{A_j}^{(l)}$, as in the first case it is assumed that $\psi_{A_i}^{(k)}(w) = 0$ if $w \notin A_i$. Also $A_j \subset A_i$, but the atom A_i is partitioned where $\psi_{A_i}^{(0)}$ takes constant

values then also $\psi_{A_i}^{(k)}$ takes constant values on A_i, therefore $\psi_{A_i}^{(k)}(w) = c$ for all $w \in A_j$, then

$$\begin{aligned}
[\psi_{A_i}^{(k)}, \psi_{A_j}^{(l)}] &= \int_\Omega \langle \psi_{A_i}^{(k)}(w), \psi_{A_j}^{(l)}(w) \rangle \, dP(w) \\
&= \int_{A_i} \langle \psi_{A_i}^{(k)}(w), \psi_{A_j}^{(l)}(w) \rangle \, dP(w) + \int_{A_j} \langle \psi_{A_i}^{(k)}(w), \psi_{A_j}^{(l)}(w) \rangle \, dP(w) \\
&= \int_{A_i} \langle \psi_{A_i}^{(k)}(w), 0 \rangle \, dP(w) + \int_{A_j} \langle c, \psi_{A_j}^{(l)}(w) \rangle \, dP(w) \\
&= c \int_{A_j} \psi_{A_j}^{(l)}(w) \, dP(w) \\
&= 0
\end{aligned}$$

because by definition of the VGS functions we require $\mathbf{E}(\psi_{A_j}^{(l)}) = 0$.

□

The VGS algorithm stops partitioning an atom A when $X(w)$ is constant in A and if $X(w)$ is constant then the inner product $[X, \psi] = 0$ for all $\psi \in \mathcal{C}_A$. Recall that $X(w)$ is a vector valued random variable, then $X(w)$ is constant if the i-th. components $X[i](w)$ is constant for all $i = 1, \ldots, d$.

Although there are no constraints on the set Ω, the input set $X(w)$ should take a finite number of values in order for the algorithm to stop in a finite number of steps. Intuitively if $X(w)$ takes a finite number of distinct (vector) values N, the total number of elements in the final partition should be less or equal than N. This is proven in Appendix C.

Given a tree $\mathcal{T}_n$ with $n \geq 0$ then the associated *VGS approximation* is defined by the following equation

$$X_{\mathcal{T}_n} \equiv \sum_{A \in \mathcal{T}_n} \sum_{i=0}^{I_A - 1} [X, \psi_A^{(i)}] \, \psi_A^{(i)}. \tag{3.2.4}$$

Clearly, the outer summation in (3.2.4) can be rewritten recursively as follows, starting with the first iteration,

$$X_{\mathcal{T}_0} = [X, \psi_\emptyset] \, \psi_\emptyset + \sum_{i=0}^{I_\Omega - 1} [X, \psi_\Omega^{(i)}] \, \psi_\Omega^{(i)} \tag{3.2.5}$$

where $\psi_\emptyset \equiv c \, \mathbf{1}_\Omega$ and

$$c[i] = \frac{\int_\Omega X[i] \, dP}{\sqrt{\sum_{k=1}^{d} \left(\int_\Omega X[k] \, dP \right)^2}}.$$

Then

$$X_{\mathcal{T}_1} = X_{\mathcal{T}_0} + \sum_{i=0}^{I_{A^*}-1} [X, \psi_{A^*}^{(i)}]\ \psi_{A^*}^{(i)}.$$

In general

$$X_{\mathcal{T}_{n+1}} = X_{\mathcal{T}_n} + \sum_{i=0}^{I_{A^*}-1} [X, \psi_{A^*}^{(i)}]\ \psi_{A^*}^{(i)}.$$

Under appropriate conditions Appendix C shows that

$$\lim_{n\to\infty} X_{\mathcal{T}_n}(w) = X(w) \quad \text{for almost all } w \in \Omega.$$

In fact if X takes only a finite number of distinct values the above limit will actually be finite, therefore, there exists N such that $X_{\mathcal{T}_N}(w) = X(w)$ for almost all $w \in \Omega$.

3.3 Inner Product Maximization Using the Bath-tub Theorem

The goal of this section is to setup for computation the quantity $[X, \psi]$ for the case when $\psi \in \mathcal{C}_A$ ($\mathcal{C}_A$ as defined in Definition 5). To this end we introduce the following notation

$$u_1 = \mathbf{E}(\varphi_1),\quad u_2 = \mathbf{E}(\varphi_2),\quad u_3 = \mathbf{E}(\varphi_1^2),\quad u_4 = \mathbf{E}(\varphi_2^2),\quad u_5 = \mathbf{E}(\varphi_1\varphi_2), \tag{3.3.1}$$

clearly (3.1.10) gives

$$a = \frac{-b\ u_2}{u_1} \tag{3.3.2}$$

and (3.1.11) implies

$$1 = \|a\|^2\ u_3 + \|b\|^2\ u_4 + \langle a, b\rangle\ u_5. \tag{3.3.3}$$

Using (3.3.2) in (3.3.3) gives

$$\|a\|^2 = \frac{u_2^2\ \|b\|^2}{u_1^2} \tag{3.3.4}$$

and also

$$\langle a, b\rangle = \frac{-u_2\ \|b\|^2}{u_1}, \tag{3.3.5}$$

finally,

$$\|b\|^2 = \frac{u_1^2}{u_2^2\ u_3 + u_4\ u_1^2 - 2\ u_5\ u_2\ u_1}. \tag{3.3.6}$$

It is important to point that from equations (3.3.1) and (3.1.8) , $u_i \in [0, P(A)]\quad i = 1\ldots 5$ and $u_3 \leq u_1$ and $u_4 \leq u_2$.

For a given set of input signals X and a given $A \in \mathcal{A}$ we would like to compute $\sup_{\psi \in \mathcal{C}_A} [X, \psi]$, where, we recall, the inner product is defined as follows:

$$[X, \psi] \equiv \int_\Omega \langle X(w), \psi(w) \rangle \, dP(w), \tag{3.3.7}$$

then replacing the definition of ψ in equation (3.3.7) we obtain

$$[X, \psi] = \|b\| \, u_2 \left(\frac{1}{u_2} \int_A \langle X(w), b' \rangle \, \varphi_2(w) \, dP(w) - \frac{1}{u_1} \int_A \langle X(w), b' \rangle \, \varphi_1(w) \, dP(w) \right), \tag{3.3.8}$$

where

$$\|b\| = \frac{u_1}{\sqrt{u_2^2 u_3 + u_4 u_1^2 - 2 u_5 u_2 u_1}} \tag{3.3.9}$$

and

$$b' = \frac{b}{\|b\|}$$

moreover, $b' \in S^d$ is an independent variable. We can interpret b' as the weights of all input components and $\langle X(w), b' \rangle$ the weighted average signal.

Equation (3.3.8) implies that the inner product depends only on the variables b', u_1, u_2, u_3, u_4, u_5, φ_1, and φ_2. Then the supremum depends only on the same list of variables and can be written as iterated suprema as follows

$$\sup_{\psi \in \hat{\mathcal{C}}_A} [X, \psi] = \sup_{b'} \left[\sup_{u_1, u_2, u_3, u_4, u_5} \left(\sup_{\varphi_1, \varphi_2} [X, \psi] \right) \right]. \tag{3.3.10}$$

Recall that we simplify the notation avoiding to write the sets where the different variables are defined, then $b' \in S^d$, $0 \leq u_i \leq P(A)$ $i = 1, 2$ and φ_i $i = 1, 2$ satisfy (3.1.8), (3.1.9) and (3.3.1).

In order to be able to simplify the above optimizations, we will need to restrict the general class of functions in $\mathcal{C}_A$ as follows: we will assume

$$||b|| = ||b||(u_1, u_2), \tag{3.3.11}$$

i.e. the norm of b depends only on u_1 and u_2. From the above formulas this is equivalent to assuming that:

$$u_k = u_k(u_1, u_2) \text{ for all } k = 3, 4, 5.$$

In short, we are restricting to functions φ_i such that their second moments and correlation depend on only both of their means.

Remark 9. *Such restricted class will be denoted $\hat{\mathcal{C}}_A$ from now on.*

Under this constraint the iterated suprema depends only on b', u_1, u_2, φ_1, and φ_2 and can be written as

$$\sup_{\psi \in \mathcal{C}_A} [X, \psi] = \sup_{b'} \left[\sup_{u_1, u_2} \left(\sup_{\varphi_1, \varphi_2} [X, \psi] \right) \right]. \tag{3.3.12}$$

In the form (3.3.12) we can already solve the inner supremum via the Bathtub Theorem (see Apendix A).

Remark 10. *A comment is needed at this point, notice that (3.3.12) provides a potentially larger supremum than the one obtained by imposing (3.3.11). It turns out that restricting (3.3.12) with (3.3.11) does not change the the value given in (3.3.12). The reason for this is that the solutions will be given by φ_1 and φ_2 from Appendix A which satisfy (3.3.12).*

We will denote with $\psi_A^{(0)}$ the function in $\hat{\mathcal{C}}_A$ which satisfies,

$$[X, \psi_A^{(0)}] = \sup_{\psi \in \hat{\mathcal{C}}_A} [X, \psi]. \tag{3.3.13}$$

As we have indicated before, $\psi_A^{(0)}$, the solution to (3.3.13), is the key step to define `bestSplit` which in turn is at the heart of the VGS algorithm. The rest of the present Chapter details how to construct $\psi_A^{(0)}$. Appendix C explains how to completely define `bestSplit` using $\psi_A^{(0)}$.

As it will be seen in the following sections, we will impose even further constraints to (3.3.12) so as to have faster optimizations.

3.3.1 Iterative Optimization for b'

At this point in the developments it is reasonable to describe a practical way to perform a fast local optimization on b'. It will be used throughout the thesis and it applies to all the different cases (this can be seen easily given the generality of the argument presented below). The expression of b' that we will write is also crucial to attain convergence of the VGS approximation, this can be seen from he developments in Appendix C.

A necessary condition at a global maxima for $\sup_{\psi \in \mathcal{C}_A} [X, \psi]$ can be obtained by conditioning in given functions φ_1 and φ_2. A simple inspection then indicates that we obtain a linear functional on b' constrained to $b' \in S^d$. This can be solved by

Lagrange multipliers, if we denote with $\hat{b}'$ the resulting optimal value we then have:

$$\hat{b}'_i = \frac{\frac{1}{u_2}\int_A X[i]\ \varphi_2\ dP - \frac{1}{u_1}\int_A X[i]\varphi_1\ dP}{\sqrt{\sum_{k=1}^{d}\left(\frac{1}{u_2}\int_A X[k]\ \varphi_2 dP - \frac{1}{u_1}\int_A X[k]\ \varphi_1 dP\right)^2}}, \tag{3.3.14}$$

where, as it was defined previously, $u_i = \int_A \varphi_i dP$.

In practice the functions φ_i will not be known independently of b' and hence the above formula can not be used directly. In fact, the Bathtub Theorem provides explicit solutions (up to a one-dimensional optimization in the variables u_i) for the functions φ_i *conditional* on knowing b'. This remarks indicate that one can iteratively compute the optimal b' and then, conditional on this b', compute the optimal pair φ_i etc. This iterative optimization is a local optimization as it requires a starting pair of functions φ_i or it does require an starting value for b'.

To address this problem, we need to supply a global optimization algorithm to run jointly with the above described local optimization. For a discussion of the global optimization techniques used in this thesis, we refer the reader to Section 4.4.

3.4 Full Bathtub Case: $\varphi_2 = 1_A - \varphi_1$

In order to simplify the optimization in (3.3.12) we will first consider the special case $\varphi_2 = 1_A - \varphi_1$. Under this constraints, Definition 4 is satisfied and equation (3.1.7) can be written as

$$\psi(w) = a\ \varphi_1(w) + b\ (1_A - \varphi_1(w)) \quad \forall w \in \Omega. \tag{3.4.1}$$

Then equation (3.3.8) becomes

$$[X, \psi] = \|b\| \left(\int_A \langle X(w), b' \rangle\ (1_A - \varphi_1(w))\ dP(w) - \frac{u_2}{u_1} \int_A \langle X(w), b' \rangle\ \varphi_1(w)\ dP(w) \right),$$

and

$$[X, \psi] = \|b\| \left(\int_A \langle X(w), b' \rangle\ dP(w) - \frac{u_1 + u_2}{u_1} \int_A \langle X(w), b' \rangle\ \varphi_1(w)\ dP(w) \right), \tag{3.4.2}$$

Then as we have defined in (3.3.1)

$$u_1 = \mathbf{E}(\varphi_1) = \int_\Omega \varphi_1(w)\ dP(w) = \int_A \varphi_1(w)\ dP(w)$$

in the same way

$$u_2 = \mathbf{E}(\varphi_2) = \int_\Omega 1_A - \varphi_1(w)\ dP(w) = P(A) - u_1.$$

Also, as indicated previously, we have assumed that u_3 depends only on u_1.

$$u_3 = \mathbf{E}(\varphi_1^2) = \int_\Omega \varphi_1^2(w)\ dP(w) = u_3(u_1),$$

and

$$u_4 = \mathbf{E}(\varphi_2^2) = \int_\Omega (1_A - \varphi_1(w))^2(w)\ dP(w) = P(A) + u_3(u_1) - 2\ u_1,$$

and

$$u_5 = \mathbf{E}(\varphi_1\ \varphi_2) = \int_\Omega \varphi_1(w)\ (1_A - \varphi_1(w)))\ dP(w) = u_1 - u_3(u_1).$$

Then equation (3.3.9) becomes

$$\|b\| = \frac{u_1}{\sqrt{P(A)^2\ u_3(u_1) - P(A)\ u_1^2}}.$$

Focusing into the inner product (3.4.2), the equation can be rewritten as follows:

$$[X,\psi] = \sqrt{\frac{u_1^2\ P(A)}{P(A)\ u_3(u_1) - u_1^2}} \left[\frac{1}{P(A)} \int_A \langle X(w), b' \rangle\ dP(w) - \frac{1}{u_1} \int_A \langle X(w), b' \rangle\ \varphi_1\ dP(w)\right]. \tag{3.4.3}$$

It is important to remark that the inner product $[X,\psi]$ depends only on b', u_1 and φ_1, (the value of u_3 depends only on u_1. Now let us define

$$\Gamma(b', u_1, \varphi_1) = [X,\psi],$$

then we compute the supremum as

$$\sup_{\psi \in \mathcal{C}_A} [X,\psi] = \sup_{b' \in S^d} \left[\sup_{0<u_1<P(A)} \left(\sup_{0\le\varphi_1(w)\le 1} \Gamma(b', u_1, \varphi_1)\right)\right].$$

In order to compute the inner supremum, we can see in (3.4.3) that there is only one term affected because the first factor and the first term in the second factor are fixed, therefore to find the supremum of (3.4.3) we have to compute the infimum of the second term of the second factor.

$$\sup_{\varphi_1} \Gamma(b', u_1, \varphi_1) = \frac{u_1}{\sqrt{P(A)\ u_3(u_1) - u_1}} \times \left[\frac{1}{P(A)} \int_A \langle X, b' \rangle\ dP(w) - \frac{1}{u_1} \inf_{\varphi_1} \int_{A_0} \langle X, b' \rangle \varphi_1(w)\ dP(w)\right]. \tag{3.4.4}$$

To find this infimum we use the Bathtub principle (Appendix A) that gives us the best φ_1 for a given u_1 and b'. The Bathtub solution for φ_1 is the following

$$\varphi_1 = 1_{\{w \in A : X[b'](w) < y_1\}} + c\ 1_{\{w \in A : X[b'](w) = y_1\}},$$

then

$$u_1 = \int_\Omega \varphi_1(w) dP(w) = \int_\Omega \left(1_{\{w \in A : X[b'](w) < y_1\}} + c\ 1_{\{w \in A : X[b'](w) = y_1\}}\right) dP(w),$$

and

$$u_1 = P(\{X[b'] < y_1\} \cap A) + c\ P(\{X[b'] = y_1\} \cap A),$$

solving for c gives

$$c = \frac{u_1 - P(\{X[b'] < y_1\} \cap A)}{P(\{X[b'] = y_1\} \cap A)}, \tag{3.4.5}$$

now we can see that

$$u_3 = u_3(u_1) = \int_\Omega \varphi_1^2(w) dP(w) = P(\{X[b'] < y_1\} \cap A) + c^2\ P(\{X[b'] = y_1\} \cap A).$$

From equation (3.4.5) it is clear that there are two different cases, the case $c = 0$ and the case $c \neq 0$.

Case I: $c = 0$

$$\varphi_1 = \mathbf{1}_{\{w \in A : X[b'](w) < y_1\}}, \tag{3.4.6}$$

$$\varphi_2 = \mathbf{1}_{\{w \in A : X[b'](w) \geq y_1\}}, \tag{3.4.7}$$

$$u_1 = P(\{X[b'] < y_1\} \cap A),$$

$$u_2 = P(A) - u_1,$$

$$u_3 = u_1,$$

$$u_4 = P(A) - u_1,$$

$$u_5 = 0.$$

Case II: $c \neq 0$

$$\varphi_1 = \mathbf{1}_{\{w \in A : X[b'](w) < y_1\}} + c\ \mathbf{1}_{\{w \in A : X[b'](w) = y_1\}},$$

$$\varphi_2 = \mathbf{1}_{\{w \in A : X[b'](w) > y_1\}} + (1 - c)\ \mathbf{1}_{\{w \in A : X[b'](w) = y_1\}},$$

$$u_1 = P(\{X[b'] < y_1\} \cap A) + c\ P(\{X[b'] = y_1\} \cap A) \tag{3.4.8}$$

$$u_2 = P(A) - u_1$$

$$u_3 = P(\{X[b'] < y_1\} \cap A) + c^2\ P(\{X[b'] = y_1\} \cap A)$$

$$u_4 = P(A) + u_3 - u_1$$

$$u_5 = u_1 - u_3.$$

Then the inner product $[X, \psi]$ can be calculated using these two cases.

Case I: $c = 0$

$$[X, \psi] = \sqrt{\frac{u_1\ P(A)}{P(A) - u_1}} \left[\frac{1}{P(A)} \int_A \langle X(w), b' \rangle\ dP(w) - \frac{1}{u_1} \int_A \langle X(w), b' \rangle\ \varphi_1\ dP(w) \right].$$

Case II: $c \neq 0$

$$[X, \psi] = \sqrt{\frac{u_1^2\ P(A)}{P(A)\ u_3 - u_1^2}} \left[\frac{1}{P(A)} \int_A \langle X(w), b' \rangle\ dP(w) - \frac{1}{u_1} \int_A \langle X(w), b' \rangle\ \varphi_1\ dP(w) \right].$$

3.4.1 Haar Case

Under the assumption that

$$P(\{X[b'] = y_1\}) = 0 \quad \text{for all } y_1 \in \mathbb{R} \text{ and all } b' \in S^d \tag{3.4.9}$$

it follows that

$$\varphi_1 = 1_{\{w \in A : X[b'](w) < y_1\}} \tag{3.4.10}$$

and

$$\varphi_2 = 1_A - 1_{\{w \in A : X[b'](w) < y_1\}} = 1_{\{w \in A : X[b'](w) \geq y_1\}} \tag{3.4.11}$$

also

$$u_1 = \int_\Omega \varphi_1 \, dP = P(\{X[b'] < y_1\} \cap A),$$

$$u_2 = P(A) - u_1,$$

$$u_3 = u_1,$$

$$u_4 = P(A) - u_1,$$

$$u_5 = 0$$

Remark 11. *We note that it can be proven that (3.4.9) follows if we assume that the cumulative distribution function of X is continuous. This explains why we mentioned the Haar case in connection with this last condition during Chapter 2.*

Remark 12. *Notice that the Full Bathtub case contains the Haar case as a special case.*

The norm of b, expressed in (3.3.9), under the conditions assumed above, becomes

$$\|b\| = \sqrt{\frac{u_1}{P(A)\ (P(A) - u_1)}}, \tag{3.4.12}$$

also equation (3.4.2) can be written as

$$[X, \psi] = \sqrt{\frac{u_1\ P(A)}{P(A) - u_1}} \left[\frac{1}{P(A)} \int_A \langle X(w), b' \rangle \, dP(w) - \frac{1}{u_1} \int_A \langle X(w), b' \rangle \ \varphi_1 \, dP(w) \right]. \tag{3.4.13}$$

Again for this case it is important to remark that the inner product $[X, \psi]$ depends only on b', u_1 and φ_1. Now let us define

$$\Gamma(b', u_1, \varphi_1) = [X, \psi],$$

then the supremum is calculated as

$$\sup_{\psi \in \hat{\mathcal{C}}_A} [X, \psi] = \sup_{b' \in S^d} \left[\sup_{0 < u_1 < P(A)} \left(\sup_{0 \le \varphi_1(w) \le 1} \Gamma(b', u_1, \varphi_1) \right) \right],$$

where

$$\sup_{\varphi_1} \Gamma(b', u_1, \varphi_1) = \sqrt{\frac{u_1 \; P(A)}{P(A) - u_1}} \times \left[\frac{1}{P(A)} \int_A \langle X(w), b' \rangle \; dP(w) - \frac{1}{u_1} \inf_{\varphi_1} \int_A \langle X(w), b' \rangle \; \varphi_1 \; dP(w) \right]. \quad (3.4.14)$$

3.5 Martingale Differences Case

This case is a generalization of the Haar case (and so it contains it as a special case) and there is no constraint for the form of φ_1 and φ_2. As in the Haar case we will require that

$$P(\{X[b'] = y_1\}) = 0 \quad \text{for all} \quad y_1 \in \mathbb{R} \quad \text{and all} \quad b' \in S^d. \tag{3.5.1}$$

Recall that, in general, we have assumed

$$u_k = u_k(u_1, u_2), \ k = 3, 4, 5, \tag{3.5.2}$$

and we will keep u_1 and u_2 fixed, therefore (3.5.2) implies that u_3, u_4 and u_5 are also fixed. As indicated, we will actually solve an optimization problem over the φ_i's under *less* stringent constraints, namely only their first moments will be fixed. So the inner supremum is actually computed over a larger set, therefore, we need to check a posteriori if the obtained solution satisfies (3.5.2); if it does, we are then proving that the solution obtained over the larger set will be equal to the one obtained with more constraints.

Analyzing equation (3.3.8) it is possible to appreciate that the supremum of the inner product can be computed finding a supremum of the first term and an infimum of the second term. Defining

$$[X, \psi] = \|b\| \ u_2 \ \Gamma(b', u_1, u_2, \varphi_1, \varphi_2),$$

then for a fixed b', u_1 and u_2 we can write

$$\sup_{\varphi_1, \varphi_2} \Gamma(\varphi_1, \varphi_1) = \frac{1}{u_2} \sup_{\varphi_2} \int_\Omega \langle X(w), b' \rangle \ \varphi_2(w) \, dP(w) - \frac{1}{u_1} \inf_{\varphi_1} \int_\Omega \langle X(w), b' \rangle \ \varphi_1(w) \, dP(w), \tag{3.5.3}$$

To calculate the supremum of the first term of equation (3.5.3) the dual version of Theorem 1 in Appendix A is needed. This dual version is described in Corollary 1 in Appendix A and shows that the supremum is realized at

$$\varphi_1 = \mathbf{1}_{\{w \in A : X[b'](w) \leq y_1\}} \tag{3.5.4}$$

and

$$\varphi_2 = \mathbf{1}_{\{w \in A : X[b'](w) \geq y_2\}} \tag{3.5.5}$$

now

$$u_3 = E(\varphi_1^2) = u_1$$
$$u_4 = E(\varphi_2^2) = u_2$$

and

$$u_5 = E(\varphi_1\ \varphi_2) = P(\{w \in A : X[b'](w) \leq y_1\}\ \cap\ \{w \in A : X[b'](w) \geq y_2\})$$

then

$$u_5 = \begin{cases} 0, & \text{if } y_1 \leq y_2, \\ u_1 + u_2, & \text{if } y_1 > y_2. \end{cases}$$

Then the condition (3.5.2) is satisfied.

Recall that taking $y_1 = y_2$ implies that $u_5 = 0$ and we get the usual Haar functions that we have been using before.

3.6 Children of Atoms

In previous sections we have assumed a function called `bestSplit` which for a given atom $A \in \mathcal{A}$, provides a finite number of best functions and a partition of A into a finite number of best children called $A_i \in \mathcal{A}$. Now we are going to explain how to obtain such children depending on the case selected.

3.6.1 Haar Case

Let us start with the simplest case, the Haar Case:

Equations (3.4.10) and (3.4.11) show that for a given atom $A \in \mathcal{A}$

$$\psi_A^{(0)} = a\ \mathbf{1}_{\{w \in A : X[b'](w) < y_1\}} + b\ \mathbf{1}_{\{w \in A : X[b'](w) \geq y_1\}} \tag{3.6.1}$$

where the first and second terms are disjoint, therefore $\psi_A^{(0)}$ takes only two possible values, one in each partition, the constant value a when $X[b'](w) < y_1$ and the constant value b when $X[b'](w) \geq y_1$. In the Haar case there is only one VGS function then

$$\psi_A = \psi_A^{(0)},$$

and so the best children are given by:

$$A_0 \equiv \{w \in A : X[b'](w) < y_1\} \text{ with } \psi_A(w) = a \text{ for all } w \in A_0$$

and

$$A_1 \equiv \{w \in A : X[b'](w) \geq y_1\} \text{ with } \psi_A(w) = b \text{ for all } w \in A_1$$

Equation (3.6.1) can be written as follows

$$\psi_A = a\ \mathbf{1}_{A_0} + b\ \mathbf{1}_{A_1}.$$

And the moments can be obtained easily

$$\varphi_1 = \mathbf{1}_{A_0}, \quad u_1 = P(A_0)$$
$$\varphi_2 = \mathbf{1}_{A_1}, \quad u_2 = P(A_1).$$

3.6.2 Full Bathtub Case

From equations (3.4.6) and (3.4.7)

$$\psi_A^{(0)} = a\ (\mathbf{1}_{\{w \in A : X[b'](w) < y_1\}} + c\ \mathbf{1}_{\{w \in A : X[b'](w) = y_1\}}) + \\ + b\ (\mathbf{1}_{\{w \in A : X[b'](w) > y_1\}} + (1-c)\ \mathbf{1}_{\{w \in A : X[b'](w) = y_1\}}) \tag{3.6.2}$$

then

$$\psi_A^{(0)} = a\ 1_{\{w\in A: X[b'](w)<y_1\}} + b\ 1_{\{w\in A: X[b'](w)>y_1\}} + (b + c(a-b))\ 1_{\{w\in A: X[b'](w)=y_1\}}) \quad (3.6.3)$$

and is possible to observe two different cases

Case I: $c = 0$

$$\begin{aligned}\psi_A^{(0)} &= a\ 1_{\{w\in A: X[b'](w)<y_1\}} + b\ 1_{\{w\in A: X[b'](w)>y_1\}} + b\ 1_{\{w\in A: X[b'](w)=y_1\}} \\ &= a\ 1_{\{w\in A: X[b'](w)<y_1\}} + b\ 1_{\{w\in A: X[b'](w)\geq y_1\}}\end{aligned}$$

then

$$A_0 \equiv \{w \in A : X[b'](w) < y_1\} \text{ with } \psi_A^{(0)}(w) = a \text{ for all } w \in A_0$$

$$A_1 \equiv \{w \in A : X[b'](w) \geq y_1\} \text{ with } \psi_A^{(0)}(w) = b \text{ for all } w \in A_1$$

Case II: $c \neq 0$

$$A_0 \equiv \{w \in A : X[b'](w) < y_1\} \text{ with } \psi_A^{(0)}(w) = a \text{ for all } w \in A_0$$

$$A_1 \equiv \{w \in A : X[b'](w) > y_1\} \text{ with } \psi_A^{(0)}(w) = b \text{ for all } w \in A_1$$

$$A_2 \equiv \{w \in A : X[b'](w) = y_1\} \text{ with } \psi_A^{(0)}(w) = b + c(a-b) \text{ for all } w \in A_2$$

Remark 13. *Whenever $c = 0$ we obtain back the Haar case and hence only two children are obtained. In the case $c \neq 0$ an special optimization for the Bathtub algorithm is needed. Also another important result is the fact that $\psi^{(0)}$ takes three different values in Case II, then (as indicated in Appendix C) an extra function $\psi^{(1)}$ is required.*

Equation (3.6.3) can be written as follows

$$\psi_A^{(0)} = \begin{cases} a\ 1_{A_0} + b\ 1_{A_1}, & \text{if } c = 0 \\ a\ 1_{A_0} + b\ 1_{A_1} + b + c(a-b)\ 1_{A_2}, & \text{if } c \neq 0. \end{cases}$$

3.6.3 Martingale Differences Case

From equations (3.5.4) and (3.5.5)

$$\psi_A^{(0)} = a\ 1_{\{w\in A: X[b'](w)\leq y_1\}} + b\ 1_{\{w\in A: X[b'](w)\geq y_2\}} \quad (3.6.4)$$

and is possible to observe two different cases

Case I: $y_1 \leq y_2$

$$A_0 \equiv \{w \in A : X[b'](w) < y_1\} \text{ with } \psi_A^{(0)}(w) = a \text{ for all } w \in A_0$$

$$A_1 \equiv \{w \in A : y_2 < X[b'](w)\} \text{ with } \psi_A^{(0)}(w) = b \text{ for all } w \in A_1$$
$$A_2 \equiv \{w \in A : y_1 \leq X[b'](w) \leq y_2\} \text{ with } \psi_A^{(0)}(w) = 0 \text{ for all } w \in A_2.$$

Case II: $y_1 > y_2$

$$A_0 \equiv \{w \in A : X[b'](w) \leq y_2\} \text{ with } \psi_A^{(0)}(w) = a \text{ for all } w \in A_0$$
$$A_1 \equiv \{w \in A : y_1 \leq X[b'](w)\} \text{ with } \psi_A^{(0)}(w) = b \text{ for all } w \in A_1$$
$$A_2 \equiv \{w \in A : y_2 < X[b'](w) < y_1\} \text{ with } \psi_A^{(0)}(w) = a + b \text{ for all } w \in A_2.$$

Remark 14. *The atom A_2 satisfies $P(A_2) = 0$, and hence it can be ignored, only when $y_1 = y_2$, this condition provides the Haar case as a special instance of the present case. In the discreet case, considered in Chapter 4, the Haar case becomes a special case of the Martingale Differences case when $y_1 = r_k$ and $y_2 = r_{k-1}$ because then $A_2 = \{w \in A : r_{k-1} < X[b'](w) < r_k\} = \emptyset$.*

Equation (3.6.4) can be written as follows

$$\psi_A^{(0)} = \begin{cases} a\ \mathbf{1}_{A_0} + b\ \mathbf{1}_{A_1}, & \text{if } y_1 \leq y_2 \\ a\ \mathbf{1}_{A_0} + b\ \mathbf{1}_{A_1} + (a+b)\ \mathbf{1}_{A_2}, & \text{if } y_1 > y_2. \end{cases}$$

Remark 15. *In both cases $\psi^{(0)}$ takes three different values (unless $y_1 = y_2$) and whenever $y_1 > y_2$ the third value on $A^{(2)}$ is equal to $a + b$ then again an extra function $\psi^{(1)}$ is required in order to obtain the mean values in the leaves (and hence the approximation will converge to the input vector signal).*

3.7 Adding More VGS Functions at a Node

So far we have described how to construct $\psi_A^{(0)}$ at a given atom A. Notice that depending on the case (Haar, Full Bathtub or Martingale Differences) we will have $I_A = 1$ (Haar case) or $I_A = 2$ (the remaining two cases). It follows from the proof presented in Appendix C (more specifically from the condition in (C.2.34)) that we need to have I_A VGS functions at each node A in order to achieve convergence. This implies that, for the Full Bathtub and Martingale Differences cases, we need to specify a new function which we have been denoting $\psi_A^{(1)}$. According to our VGS algorithm, this function needs to satisfy some basic properties listed in the `bestSplit` routine which is detailed at the end of Appendix C.

The functions $\psi^{(1)}$ could be constructed in several ways. Moreover, notice that the decision, taken by VGS, on how to split a given atom, is based solely in $\psi_A^{(0)}$. One may also decide how to split an atom based on both functions $\psi_A^{(i)}$, $i = 0, 1$, this is suggested in (C.4.1).

Below, we describe a canonical construction of $\psi_A^{(1)}$ which depends on knowing $\psi_A^{(0)}$ beforehand. It relies on introducing *scalar* VGS functions which are important in its own right, for this reason we dedicate the following Section to this topic. As noted, we only need to provide the construction of $\psi_A^{(1)}$ for the cases Full Bathtub and Martingale Differences. Therefore, we may assume without loss of generality that A splits into three children.

3.7.1 Scalar VGS Functions

Starting with a vector valued VGS function $\psi_A^{(0)} = a\varphi_1 + b\varphi_2$ at node A we will show how to construct $\psi_A^{(1)}$. To describe this construction we need to introduce scalar VGS functions at a given node A. As we have done before, we will write ψ instead of ψ_A once the atom A is understood.

If $\psi_A = \psi = a\varphi_1 + b\varphi_2$ is a vector valued VGS function, we will use the following notation for the associated scalar function

$$\psi_{A,s} = \psi_s = d_1\varphi_1 + d_2\varphi_2,$$

which should satisfy,

$$\int_\Omega \psi_s(w)\, dP(w) = 0, \tag{3.7.1}$$

$$\int_\Omega \psi_s^2(w)\, dP(w) = 1. \tag{3.7.2}$$

This allows us to write the scalar basis function as follows

$$\psi_s = d_1\ \varphi_1 + d_2\ \varphi_2 = d_2\ u_2\ \left(\frac{\varphi_2}{u_2} - \frac{\varphi_1}{u_1}\right) =$$

$$|d_2|\ u_2\ d_2'\ \left(\frac{\varphi_2}{u_2} - \frac{\varphi_1}{u_1}\right).$$

Where $d_2' \equiv d_2/|d_2| \in \{-1, 1\}$. Notice also that

$$|d_2| = ||b||. \tag{3.7.3}$$

Therefore

$$\psi_s^{(0)} = ||b||\ u_2\ d_2'\ \left(\frac{\varphi_2}{u_2} - \frac{\varphi_1}{u_1}\right).$$

In short $\psi^{(0)}$ specifies $\psi_s^{(0)}$ uniquely.

As it was indicated above, we only need to describe the construction of $\psi^{(1)}$ for the case when A splits into three children A_0, A_1, A_2 (so $A_i \cap A_j = \emptyset$ whenever $i \neq j$ and $\cup_{i=0}^2 A_j = A$). This means we may assume we are given

$$\psi_{A,s}^{(0)} = a_0 \mathbf{1}_{A_0} + a_1 \mathbf{1}_{A_1} + a_2 \mathbf{1}_{A_2}. \tag{3.7.4}$$

We will now describe how to construct a function $\psi_{A,s}^{(1)} = \psi_s^{(1)}$ which will be the one used to construct $\psi_A^{(1)}$.

We set

$$\psi_s^{(1)} = e_0 \mathbf{1}_{A_0} + e_1 \mathbf{1}_{A_1} + e_2 \mathbf{1}_{A_2},$$

where the $e_i \in \mathbb{R}$. We will impose the following constraints :

$$1 = ||\psi_s^{(1)}||^2 = [\psi_s^{(1)}, \psi_s^{(1)}]_1 = e_0^2\ P(A_0) + e_1^2\ P(A_1) + e_2^2\ P(A_2),$$

$$0 = \int_A \psi_s^{(1)}\ dP(w) = e_0\ P(A_0) + e_1\ P(A_1) + e_2\ P(A_2),$$

and

$$0 = [\psi_s^{(0)}, \psi_s^{(1)}]_1 = a_0\ e_0\ P(A_0) + a_1\ e_1\ P(A_1) + a_2\ e_2\ P(A_2).$$

It is easy to find the solution to the above system of equations:

$$e_2 = \frac{-\ e_0\ P(A_0)\ (a_1 - a_0)}{P(A_2)\ (a_1 - a_2)},$$

$$e_1 = \frac{-\ e_0\ P(A_0)\ (a_2 - a_0)}{P(A_1)\ (a_2 - a_1)},$$

and

$$e_0^2 = \frac{P(A_1)\ P(A_2)\ (a_2 - a_1)^2}{P(A_0)\ [P(A_1)P(A_2)(a_2 - a_1)^2 + P(A_0)P(A_2)(a_2 - a_0)^2 + P(A_0)P(A_1)(a_1 - a_0)^2]}.$$

Finally we define $\psi_A^{(1)}$ as follows:

$$\psi^{(1)}[i](w) \equiv \frac{[X[i], \psi_s^{(1)}]_1\ \psi_s^{(1)}(w)}{\sqrt{\sum_{j=1}^{d} [X[j], \psi_s^{(1)}]_1^2}}. \tag{3.7.5}$$

This function satisfies all the required properties required by `bestSplit`, these results are presented in Appendix C.

3.8 Vector and Scalar Approximation

At this point in this Chapter we have completed the description of an orthonormal system $\mathcal{H} = \{\mu_k\}$ (see formal definition in (3.2.3). Elements from $\mathcal{H}$ are vector valued VGS functions of the type $\psi_A^{(i)}$, $i = 0, 1$. As we have done before elements in $\mathcal{H}$ will be labelled μ_k. Note that the construction in the previous Section provides an scalar function $\psi_{A,s}^{(i)}$ associated to each function $\psi_A^{(i)}$. Clearly, this defines an orthonormal system of scalar valued VGS functions, we will label this system with $\mathcal{G}$. Elements from $\mathcal{G} = \{u_k\}$ are labelled u_k (see Appendix C). From the construction in the previous Section, there is a natural association between elements from $\mathcal{H}$ and elements from $\mathcal{G}$ and we will assume u_k is the element in $\mathcal{G}$ naturally associated with μ_k.

The results in Appendix C show that for any finite index set $I \subseteq \mathbb{N}$ we have the fundamental identity

$$\sum_{k \in I} [X, \mu_k]\mu_k[i] = \sum_{k \in I} [X[i], u_k]_1\ u_k \quad \text{for all} \quad i = 1, \ldots, d.$$

This is a basic result and shows, along with the convergence result in Appendix C, that one could use the vector valued orthonormal system $\mathcal{H}$ to approximate X or one could use the scalar valued orthonormal system $\mathcal{G}$ to approximate each $X[i]$, $i = 1, \ldots, d$.

The two systems, $\mathcal{H}$ and $\mathcal{G}$, are not equivalent when one considers the optimized expansions as we explain next.

Let $h : \mathbb{N} \longrightarrow \mathbb{N}$ be a re-ordering function for $\mathcal{H}$ in such a way that

$$|[X, \mu_{h(0)}]| \geq |[X, \mu_{h(1)}]| \geq \cdots.$$

We then have the n-term VGS optimized approximation given by

$$X_n = \sum_{k=0}^{n-1} [X, \mu_{h(k)}]\ \mu_{h(k)}. \tag{3.8.1}$$

In practice, the integer n is chosen to satisfy some error criteria, say an vector error level ϵ_v is given so we can find $n = n(\epsilon_v)$ so that

$$||X - X_n|| \leq \epsilon_v.$$

As a side remark, we mention that the software implementation works with the set of inner products

$$\Lambda \equiv \{\lambda_k \equiv [X, \mu_{h(k)}]\},$$

and inserts them in a priority queue $\mathcal{Q}_\Lambda$ (ordered by the sizes of the inner products $|\lambda_k|$.)

One can define the same notions for the orthonormal system $\mathcal{G}$, let $g_i : \mathbb{N} \to \mathbb{N}$ one such re-ordering function for each $i = 1, \ldots, d$, so that

$$|[X[i], u_{g_i(0)}]_1| \geq\ |[X[i], u_{g_i(1)}]_1| \geq \ldots.$$

We then define the n-term VGS optimized approximation by

$$X[i]_n = \sum_{k=0}^{n-1} [X[i], u_{h(k)}]_1\ u_{h(k)}. \tag{3.8.2}$$

(a caution to the reader: please do not confuse the i-th. component of the n-term vector approximation $X_n[i]$ with the n-term approximation of the scalar i-th. component $X[i]_n$).

Then, given an scalar error level ϵ_s we will find integers n_i such that

$$||X[i] - X[i]_{n_i}|| \equiv \sqrt{[X[i] - X[i]_{n_i}, X[i] - X[i]_{n_i}]_1} \leq \epsilon_s \ \text{ for all } \ i = 1, \ldots, d.$$

Therefore, we have two optimized approximations, the optimized VGS approximation given by (3.8.1) (which we call the *vector approximation*) and the d optimized scalar VGS approximations given by (3.8.2) (which we call the *scalar approximations*). Given these two types of approximations, we will prune the tree by keeping only the active nodes for further processing. The pruning in each case will give rise to two different set of active nodes. Given a vector error level ϵ_v, after pruning the tree we will need to store information related to $n = n(\epsilon_v)$ active nodes in the tree. In the scalar case, given an scalar error level ϵ_s, each component $X[i]$ requires $n_i = n_i(\epsilon_s)$ nodes. Of course, many of these nodes are common to several signals. In any case, the final collection of active nodes for the scalar case can be quite different than for the vector case.

3.8.1 Types of Data Needed to be Stored at Active nodes

We have called active nodes to those nodes that remain after pruning. Depending if we are performing a scalar or a vector approximation we will need to store different data types so that the reconstruction (by the decoder) of the approximation can be performed.

In general, the type of data to be stored for each type of approximation, i.e. scalar or vector, is quite different. In the vector case one needs to store the following information at the active nodes: numbers of the form $[X, \psi_A^{(0)}]$ and/or $[X, \psi_A^{(1)}]$ and a corresponding vector b'_A. In the scalar case one needs to store some (or all) of the following numbers: $[X[i], \psi_{A,s}^{(0)}]_1$ and/or $[X[i], \psi_{A,s}^{(1)}]$, $i = 1, \ldots, d$. These matters are considered with some detail in Chapter 5.

Chapter 4

Formulation for Software Implementation

In this chapter we will describe the software implementation of the different variants of the VGS algorithm that we have described in Chapter 3. A description of the algorithm in terms of a discrete setup is also developed. This discrete setup is motivated by the analysis of discrete signals that have been discretized and quantized.

This chapter is subdivided in four sections, the first three sections describe the three different methods, the first one describes the Haar case, the second one shows the software implementation of the Martingale Differences case and in the third Section the Full Bathtub software implementation is provided. A final section describes global optimization techniques.

The main problem is to solve the iterated supremum introduced by equation (3.3.12). For each case we provide the solution in three steps: computation of the inner supremum, the computation of the outer supremum and a general description of the algorithm. The solution of the outer supremum makes use of a common optimization technique over the values of b', that is described in Section 4.4.

While describing the different computational steps we also provide the computational costs of the associated algorithms.

4.1 Haar Case

4.1.1 Discrete Restricted Bathtub

This case was introduced in Section 3.4.1 and the Bathtub principle (Appendix A) is used to find the infimum of the following expression, notice that now $Y : \Omega \rightarrow \mathbb{R}$ is a one dimensional random variable (in the actual applications we will specialize to $X[b']$ where X is the input vector).

$$I = \inf_{\varphi \in \mathcal{D}_u} \int_{\Omega} Y(w)\ \varphi(w)\ dP(w), \tag{4.1.1}$$

where

$$\varphi(w) = 1_{\{w \in A : Y(w) < y\}} + c\ 1_{\{w \in A : Y(w) = y\}} \tag{4.1.2}$$

and

$$\int_{\Omega} \varphi(w) dP(w) = u, \tag{4.1.3}$$

but for the Haar case we are assuming that $P(\{Y = y\}) = 0$ then

$$\varphi(w) = 1_{\{w \in A : Y(w) < y\}}, \tag{4.1.4}$$

and so

$$I = \int_{\{w \in A : Y(w) < y\}} Y(w)\ dP(w). \tag{4.1.5}$$

At this point it is interesting to describe the Bathtub algorithm for the discrete setup. Recall that in this special case there is only one parameter to be optimized, namely u, and is related to A_0, actually $u = P(\{Y < y\})$, where y is a value in the range of the function Y.

Next we introduce some useful notation, let $\mathcal{R}_A(Y) = \{r_0, ..., r_{n-1}\}$ be a complete ordering $r_0 < ... < r_{n-1}$ of all values $Y(w)$, $w \in A$ and $P(\{Y = r_k\}) \neq 0$ for all such r_k. Recall that $\mathcal{R}_A(Y)$ is the range of $Y(w)$. Then

$$y_k \in \mathcal{R}_A(Y), \quad \text{and} \quad y_k \equiv r_{k+1}, \quad \text{with} \quad k = 0, \ldots, n-2 \tag{4.1.6}$$

As it is possible to observe in equation (4.1.6), there exists boundary constraints that should be satisfied in order to avoid some undesirable particular cases. Figure. 4.1 shows these constraints and the associated partition.

Note that as we have previously introduced in Section 3.6.1, the children (for a given y_k) can be obtained as follows:

$$A_0 = \{w \in A : Y(w) < y_k\} \tag{4.1.7}$$

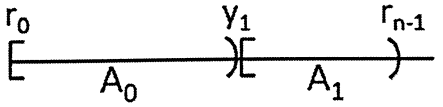


Figure 4.1: Boundaries constraints Haar Bathtub

$$A_1 = \{w \in A : Y(w) \geq y_k\}. \tag{4.1.8}$$

Let us call u_k the value of u associated with y_k. Then

$$u_k = P(\{Y < y_k\}) = P(\{Y < r_{k+1}\}) = P(\{Y \leq r_k\}) = F_Y(r_k) \tag{4.1.9}$$

then for a given u it is possible to find the corresponding r_k as follows

$$F_Y(r_k) \leq u < F_Y(r_{k+1}) \tag{4.1.10}$$

but as we will check in next section, for the Haar case the best way to iterate over the values of u is to iterate over r_k.

Remark 16. *In order to speed up the computations we pre-calculate the probabilities, the cumulative distribution function and the expected values. The cumulative function is defined as follows*

$$F_Y(r_k) = \sum_{i=0}^{k} p_i \tag{4.1.11}$$

where $p_i = P(\{Y = r_i\})$ *then*

$$F_Y(r_{k+1}) = F_Y(r_k) + p_{k+1} \quad \textit{and} \quad F_Y(r_0) = p_0. \tag{4.1.12}$$

and the expectations $\boldsymbol{E}(Y|Y \leq r_k)$

$$\boldsymbol{E}(Y|Y \leq r_k) = \sum_{i=0}^{k} p_i \; y_i \tag{4.1.13}$$

then

$$\boldsymbol{E}(Y|Y \leq r_{k+1}) = \boldsymbol{E}(Y|Y \leq r_k) + p_{k+1} \; r_{k+1} \quad \textit{and} \quad \boldsymbol{E}(Y|Y \leq r_0) = p_0 \; r_0. \tag{4.1.14}$$

Notice that $\boldsymbol{E}(Y|Y \leq r_0)$ *can be different from zero if the function does not take the zero value.*

4.1.2 Inner Supremum: Best Split Algorithm

In practical terms the `bestSplit` calculates the best partition of a vector random variable $X[b'] \equiv \langle X, b' \rangle$ in an atom $A \in \mathcal{A}$ for a given b', by maximizing the inner

product $[X, \psi_A]$. In order to find the supremum of the inner product calculated in (3.4.14), the Bathtub principle should be applied once for each value on the range. Now let

$$\hat{\lambda}(b') = \sup_{u_1, \varphi_1} \Gamma(b', u_1, \varphi_1) \tag{4.1.15}$$

then the `bestSplit` should return the best inner product and the best partition, but in order to calculate the children, the value of the range r_k is needed.

Remark 17. *Recall that $u_1 \in (0, P(A))$, but it is faster to evaluate $u_1 = F_{X[b']}(r_k)$ and apply the Bathtub theorem for each r_k with $k = 1, \ldots, n-2$, a finite set, rather than inspect the real interval $(0, P(A))$.*

The following picture shows the flow of this function. and the pseudo-code for the

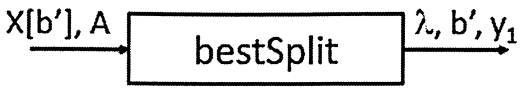


Figure 4.2: Flow chart Bathtub algorithm

`bestSplit` algorithm in the Haar case

```
Function bestSplit(X[b'])
  Calc_CDF(X[b'])            //Calculate the cumulative
  Calc_Exp(X[b'])            //Calculate the expectation
  For each ri in R(X[b'])    //Covering the range of x[b']
    ui=CDF(ri)               //Calculate ui
    I1=E(ri)                 //Calculate the expectation for the last term
    Calc_Lambda(ui,I1)       //Calculate the value of lambda
    If(lambda>lambda_max)    //Find the maximum value of lambda
      lambda_max=lambda      //Store the maximum value of lambda and ri
      r_max=ri
    End If
  End For
  Return(r_max, lambda_max)  //Return ri and lambda maxima
End Function
```

We have to precalculate $X[b']$ before it is processed by the `bestSplit` function. From the pseudo-code it is possible to see that the algorithm's order is $\mathcal{O}(n)$ where n is the number of different values that $X[b']$ takes on A. Assuming that the input images $X[i]$ take integer values in the following interval $[0, v]$, then as $b' \in S^d$ where d is the number of input signals. Then

$$-v\sqrt{d} \leq X[b'](w) = \sum_{i=1}^{d} X[i](w)\, b[i] \leq v\sqrt{d} \tag{4.1.16}$$

then

$$n \leq 2\, v\, \sqrt{d} \tag{4.1.17}$$

in case of 256 gray-levels images and 16 images n could be at most 1024.

4.1.3 Outer Supremum: b' Optimization Algorithm

Once we have found the best ψ_A for a given b' the idea is to propose different values of b' such that the inner product $[X, \psi_A]$ achieves a maximum value. As $\|b'\| = 1$ then $b' \in S^d$, the d-unit sphere, there are different techniques that can be applied in order to solve this optimization problem. These techniques are described in Section 4.4.

Outer Supremum: b' Optimization Algorithm

We construct a function called `bestVectorSplit` that finds the best partition of a vector valued random variable X for given atom $A \in \mathcal{A}$. As we have mentioned before

$$b' = \frac{b}{\|b\|} \quad \Rightarrow \quad \|b'\| = 1, \tag{4.1.18}$$

and then $b' \in S^d$. Figure 4.3 shows the flow chart of the optimization algorithm. At first the algorithm fixes a value for b' and calculates $X[b'] = \langle X, b' \rangle$ then $X[b']$ is passed to the `bestSplit` function that returns the best partition and the maximum inner product for a given b'. This is iterated until a stopping criteria is reached; this iteration is performed by a global optimization procedure. Different procedures for global optimization are described in Section 4.4.

```
Function bestVectorSplit(X, A)// X: input vector, A: current atom
  lambda_max = 0              // max inner product set to 0
  While(condition)            // loop on b'
    b'= Create()              // propose a b'
    X[b'] = CreateLC(X,b')    // construct the linear combination
    lambda = bestSplit(X[b']) // find the bestSplit on A of X[b']
    If(lambda>lambda_max)     // evaluate the maximum
      lambda_max = lambda     // actualize the maximum
    End If
    Evaluate(condition)       // evaluate the stop criteria
  End While
  Return(r_max, lambda_max, bp_max)  //Return r, lambda and b' maxima
End Function
```

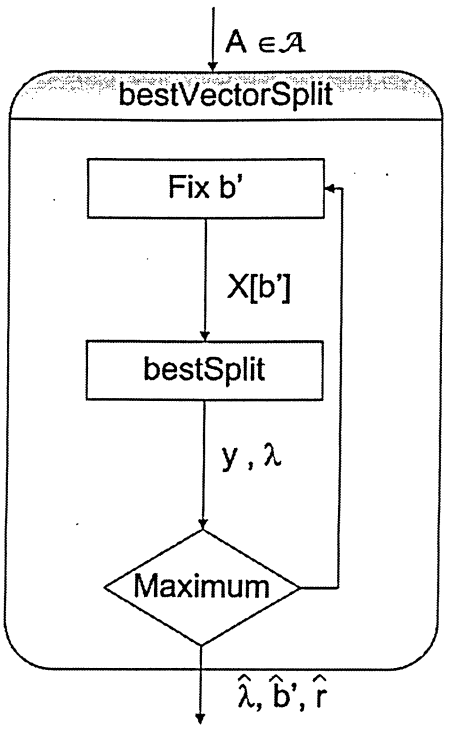


Figure 4.3: Optimization Flow chart

The algorithm returns the best $\hat{r}$, the best $\hat{\lambda}$ and the best $\hat{b}'$.

From the pseudo-code it is possible to see that the computational order of the algorithm to calculate $\sup_{\psi_A \in \hat{C}_A} [X, \psi_A]$ is $\mathcal{O}(k\ n)$ where k is the total number of iterations of the while loop, and n was explained at (4.1.17). Depending on the method selected the value of k is not always possible to be determined a priori.

4.1.4 General Description of the Algorithm

The previous VGS algorithm description gives us the background for the next step. Once the algorithm is applied to a given atom the `bestVectorSplit` returns the best partition for all inputs at the same time in the given atom. This partition defines the atom's children and the following step is to decide which atom should be split next. It seems to be a good method to select the largest inner product. The only way to know the inner product at an specific child is to apply the `bestVectorSplit` to each child.

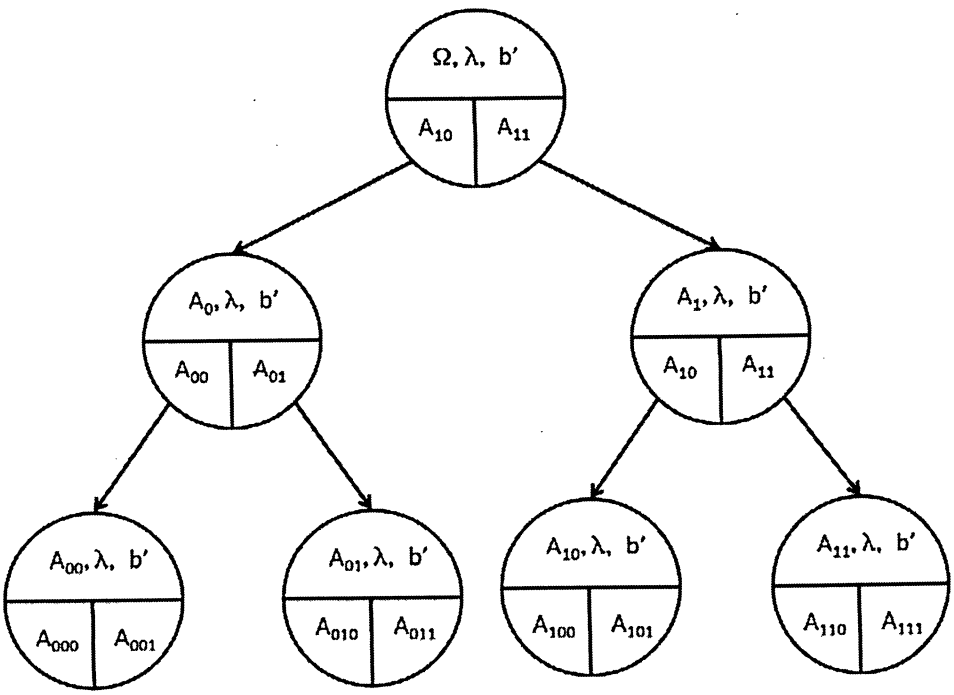


Figure 4.4: Haar Tree Description

Figure 4.4 shows the tree generated by the algorithm in three steps, and it shows the node structure composed by the inner product λ, the partition A_0 and A_1, the b' and associated r . Now, let us consider the tree $\mathcal{T}$ introduced in (3.2.2), but in

this case from the computational point of view. These nodes ν_k are composed by the inner product λ, the vector b' and the corresponding r which are used to obtain the children A_0 and A_1.

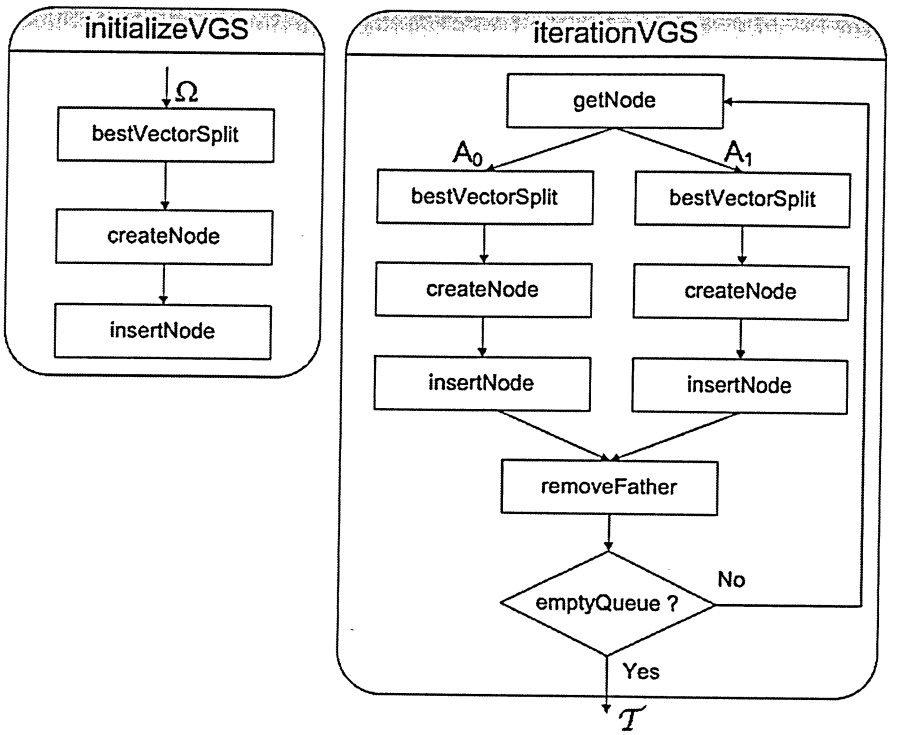


Figure 4.5: a) VGS Initialization - b) VGS Running iteration

The first step of the algorithm consists in applying `bestVectorSplit` to the vector X on the entire domain Ω and storing the best parameters in a node called root in both $\mathcal{Q}_\Lambda$ (please refer to Section 3.8 for a formal presentation of $\mathcal{Q}_\Lambda$) and the tree $\mathcal{T}$, this procedure is called `initializeVGS`, Figure.4.5a shows the flowchart of this procedure. Then, as there is only one node corresponding to the entire partition Ω in $\mathcal{Q}_\Lambda$, this node is selected and the `bestVectorSplit` is applied over the children A_0 and A_1, and the first node is removed from $\mathcal{Q}_\Lambda$, the two new nodes are created and stored in the queue and in the tree. The algorithm now selects the node in $\mathcal{Q}_\Lambda$ with greatest inner product and again applies `bestVectorSplit` over its children, then it removes this node from the queue and inserts its children; it continues recursively iterating until some criteria is reached or until there are no atoms left. This iterative process is called `iterateVGS` and Figure 4.5b graphically shows this process. When the `iterateVGS` finishes the full tree $\mathcal{T}$ is obtained.

Figure 4.5 shows secondary procedures that are important to mention.

- The procedure `getNode`, looks into the queue for the next atom to split.
- The procedure `createNode` based on the parameters obtained from the `bestVectorSplit`, creates a new node.
- The procedure `insertNode` inserts the new node in both the queue and the tree.
- The procedure `removeFather` removes the former atom from the queue.
- The function `emptyQueue` checks if there is a node left in the queue.

The pseudo code for the `initializeVGS` procedure is defined as:

```
Function initializeVGS(X)       // X: input vector
   // the best split of X in Omega
   lambda_max,r_max, bp_max=bestVectorSplit(X,Omega)
   v = createNode(lambda_max,r_max, bp_max)
   insertNode(v)                //insert node v into the queue
End Function
```

and the pseudo code for the `iterateVGS` procedure is described next

```
Function iterateVGS(X)   // X: input vector
   While(!emptyQueue){
     // the best split of X in the children  A0 of A
     lambda_max,r_max, bp_max = bestVectorSplit(X,v->A0)
     v0 = createNode(lambda_max,r_max, bp_max)
     insertNode(v0)       //insert node v0 into the queue and the tree
     // the best split of X in the children  A1 of A
     lambda_max,r_max, bp_max = bestVectorSplit(X,v->A1)
     v1 = createNode(lambda_max,r_max, bp_max)
     insertNode(v1)       //insert node v1 into the queue and the tree
     removeFather(A)      //remove father A only from the queue
  End While
End Function
```

Remark 18. *Notice that $\mathcal{T}$ in this case is a finite binary tree and we insert a new node in the tree only if the function is not not constant in that node. Therefore when considering the full tree, X is constant in the leaves' children (note the children are not included in the tree). The number of leaves' children is not larger than the number of samples in the input vector, then if N_s is the number of samples, the maximum number of nodes in tree is $N_s - 1$.*

Remark 19. *In general the total number of different vector values taken by the input vector is given by*

$$N_v = |\{v \in \{0, \ldots, 255\}^d : \exists w \in \Omega \text{ and } X(w) = v\}|. \tag{4.1.19}$$

The number of samples depends on the resolution selected and could be expressed as $N_s = W \times H$, *where* W *and* H *are the corresponding image's width and height . Note that* $N_v \leq N_s$.

Analyzing the pseudo-code of the procedure `iterateVGS`, it is possible to see that the order of the algorithm to obtain the full tree is given by

$$\mathcal{O}(N_v \; k \; n) \tag{4.1.20}$$

where k is the total number of iterations done by the procedure `bestVectorSplit`, and n was explained at (4.1.17).

At this point the reader would be interested to know the order of the algorithm in a real example. Assuming we are working with nine 256-gray level images of $128 \times 128 = 16384$ samples each one, then $N_s = 16384$, the total number of different values $N_v \approx 13000$ ($\approx 80\%$ of N_s), $n \approx 256$ and $k \approx 100$ then the total number of operations is $\approx 300E6$.

4.2 Martingale Differences

4.2.1 Discrete Bottom-up Bathtub

Introduced in Section 3.5 the Martingale Differences (MD) algorithm assumes that and there is no constraint for the form of φ_1 and φ_2 but it requires $P(\{X[b'] = c\}) = 0$ and also

$$u_k = u_k(u_1, u_2), \ k = 3, 4, 5, \tag{4.2.1}$$

This case considers the possibility to solve the Bathtub principle and its Dual version at once. Assume that the range of an scalar input X is $\mathcal{R}_A(X) = \{r_0, ..., r_{n-1}\}$. Then we consider $y_1, y_2 \in \mathcal{R}_A(X)$, and the children defined as

Case I: $y_1 \leq y_2$

$$A_0 \equiv \{w \in A : X[b'](w) < y_1\} \tag{4.2.2}$$

$$A_1 \equiv \{w \in A : y_2 < X[b'](w)\} \tag{4.2.3}$$

$$A_2 \equiv \{w \in A : y_1 \leq X[b'](w) \leq y_2\} \tag{4.2.4}$$

Case II: $y_1 > y_2$

$$A_0 \equiv \{w \in A : X[b'](w) \leq y_2\} \tag{4.2.5}$$

$$A_1 \equiv \{w \in A : y_1 \leq X[b'](w)\} \tag{4.2.6}$$

$$A_2 \equiv \{w \in A : y_2 < X[b'](w) < y_1\} \tag{4.2.7}$$

There are several cases that are potentially confusing, it is crucial to keep the following key ideas in mind in order to resolve some special cases. Recall that the sets $\{w : X[b'] < y_1\}$ $\{w : X[b'] > y_2\}$ can not have probability equal to 0 (i.e. $u_1 > 0$ and $u_2 > 0$, in particular they are not empty.) In the present discrete setting this means that we will require

$$r_0 < y_1 \leq y_2 < r_{n-1} \quad \text{and} \quad r_0 \leq y_2 < y_1 \leq r_{n-1} \tag{4.2.8}$$

Figure. 4.9 shows these constraints and the associated partition, defining the two subsets as $\{w : X[b'] < y_1\}$ $\{w : X[b'] > y_2\}$ in case a) for a given $y_1, y_2 \in R_A(X)$ where $y_1 \leq y_2$ there is no intersection between both then clearly it is possible to define three non empty sets A_0, A_1, A_2. Case b) $y_1 > y_2$ means that there is an intersection between them and also the intersection defines three subsets, also it is interesting to stress case c) that is equivalent to the Haar case where $A_2 = \emptyset$ due to there is no values in $R_A(X)$ that could satisfy the constraints.

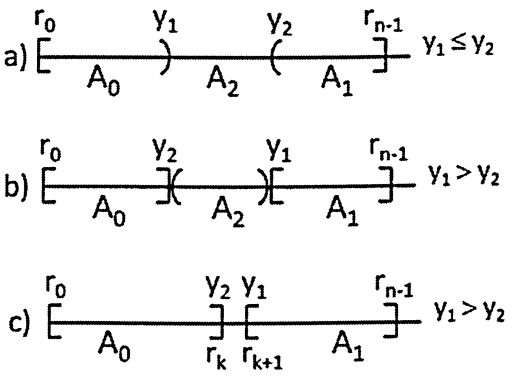


Figure 4.6: Boundaries constraints a) Case I, b) Case II, c) Haar Case

4.2.2 Inner Supremum: Best Split Algorithm

The `bestSplit` for the MD case calculates the best partition of a vector random variable X in an atom $A \in \mathcal{A}$ for a given b', by maximizing the inner product $[X, \psi_A]$, but now the number of children could be more than two, recall that we require the number of children to be greater than one, otherwise there is no split. Then the `bestSplit` should return the best inner product and the best partition, but in order to calculate the children, the values y_1 and y_2 are also needed.

The following picture shows the flow of this function. and the pseudo-code for the

Figure 4.7: Flow chart Bathtub algorithm

`bestSplit` algorithm in the MD case

```
Function bestSplit(X[b'])
  Calc_CDF(X[b'])                  //Calculate the cumulative
  Calc_Exp(X[b'])                  //Calculate the expectation
  For each ri in {r0,..,rn-2}      //Covering the range of x[b']
    u2=1-CDF(ri)                   //Calculate u1
    I2=E(rn-1)-E(ri)  //Calculate the expectation for the last term
    For each rj in {r1,..,rn-1} //Covering the range of x[b']
      u1=CDF(rj)                   //Calculate u1
      I1=E(rj)          //Calculate the expectation for the last term
      Calc_Lambda(u1, u2 , I1, I2) //Calculate the value of lambda
      If(lambda>lambda_max)        //Find the maximum value of lambda
```

```
        lambda_max=lambda   //Store the maximum value of lambda, y1, y2
        y1=ri
        y2=rj
      End If
    End For
  End For
  Return(y1,y2 , lambda_max)  //Return y1,y2 and lambda maxima
End Function
```

From the pseudo-code it is possible to see that the algorithm's order is $\mathcal{O}(n^2)$ where n is the number of different values taken by $X[b']$ explained at (4.1.17).

4.2.3 Outer Supremum: b' Optimization Algorithm

Practically, this algorithm is the same as the Haar version, the only difference remains in the values that algorithm returns, now as there exist two different values used to split the range, the best split returns both.

```
Function bestVectorSplit(X, A)// X: input vector, A: current atom
  lambda_max = 0              // max inner product set to 0
  While(condition)            // loop on b'
    b'= Create()              // propose a b'
    X[b'] = CreateLC(X,b')    // construct the linear combination
    lambda = bestSplit(X[b']) // find the bestSplit on A of X[b']
    If(lambda>lambda_max)     // evaluate the maximum
      lambda_max = lambda     // actualize the maximum
    End If
    Evaluate(condition)       // evaluate the stop criteria
  End While
  Return(y1,y2, lambda_max, bp_max)  //Return y1, y2, lambda and b' maxima
End Function
```

The algorithm returns the best $\hat{y}_1$, $\hat{y}_2$,the best $\hat{\lambda}$ and the best $\hat{b}'$.

Again from the pseudo-code is possible to see that the order of the algorithm to calculate $\sup_{\psi_A \in \hat{\mathcal{C}}_A} [X, \psi_A]$ in the MD case is $\mathcal{O}(k\ n^2)$ where k is the total number of iterations, and n was explained at (4.1.17).

4.2.4 General Algorithm Description

The difference between this case and the Haar case resides in the tree structure, in the MD case the number of children could be greater than 2. Then it is a 3-ary tree.

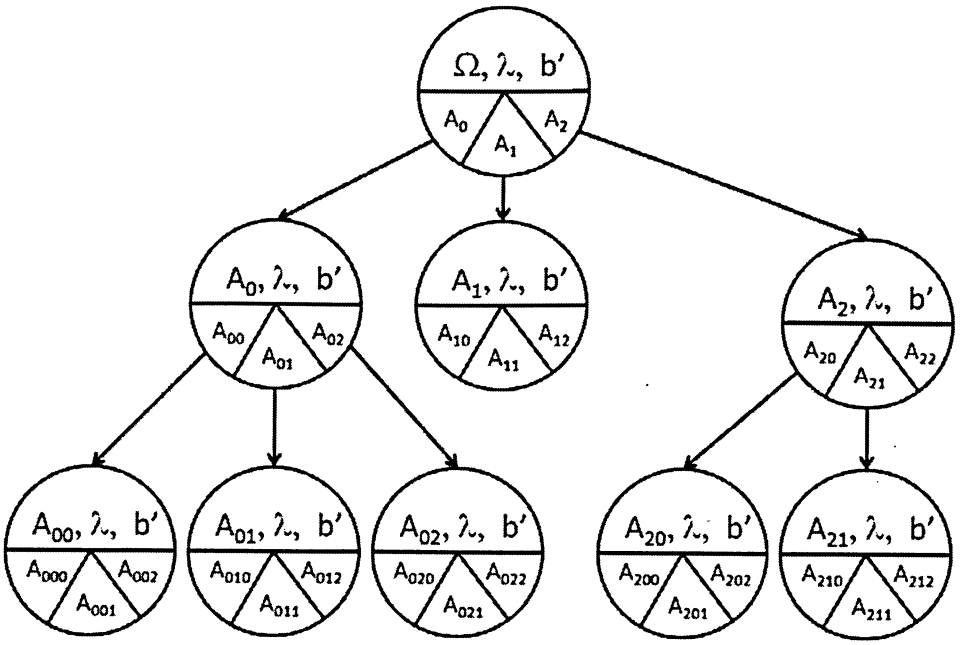


Figure 4.8: MD Tree Description

Figure 4.11 shows the tree generated by the algorithm in three steps, and it shows the node structure composed by the inner product λ, the partition A_0, A_1 and A_2, the b'.

The first step consists in applying `bestVectorSplit` to the vector X on the entire partition Ω and storing the best parameters in a node called root in both $\mathcal{Q}_\Lambda$ and the tree $\mathcal{T}$, this procedure is called `initializeVGS`, Figure.4.5a shows the flowchart of this procedure. Then as there is only one node corresponding to the entire partition Ω in $\mathcal{Q}_\Lambda$, this node is selected and the `bestVectorSplit` is applied over the children A_0, A_1 and A_2, and the first node is removed from $\mathcal{Q}_\Lambda$, then two new nodes are created and stored in the queue and in the tree, now the difference between this algorithm and the Haar version, if $A_2 \neq \emptyset$ another node is created an inserted in both the queue and the tree. The algorithm now selects the node in $\mathcal{Q}_\Lambda$ with greatest inner product and again applies `bestVectorSplit` over its children, removes this node from the queue and inserts its children; it continues recursively iterating until some criteria is reached or until there are no atoms left. This iterative process is called `iterateVGS` and Figure 4.5b graphically shows this process. When the `iterateVGS` finishes the full tree $\mathcal{T}$ is obtained.

and the pseudo code for the iterateVGS procedure in the MD case is described

```
Function iterateVGS(X)    // X: input vector
   While(!emptyQueue){
     // the best split of X in the children  A0 of A
     lambda_max,r_max, bp_max = bestVectorSplit(X,v->A0)
     v0 = createNode(lambda_max,y1,y2, bp_max)
     insertNode(v0)       //insert node v0 into the queue and the tree
     // the best split of X in the children  A1 of A
     lambda_max,r_max, bp_max = bestVectorSplit(X,v->A1)
     v1 = createNode(lambda_max,y1,y2, bp_max)
     insertNode(v1)       //insert node v1 into the queue and the tree
     If(!v->A2=0)
       v2 = createNode(lambda_max,y1,y2, bp_max)
       insertNode(v2)       //insert node v1 into the queue and the tree
     End If
     removeFather(A)      //remove father A only from the queue
  End While
End Function
```

Analyzing the pseudo-code of the procedure iterateVGS is possible to see that the order of the algorithm to obtain the full tree is given by

$$\mathcal{O}(N_v\ k\ n^2) \tag{4.2.9}$$

where N_v is obtained in (4.1.19), k is the total number of iterations done by the procedure bestVectorSplit, and n was explained at (4.1.17).

4.3 Full Bathtub Case

4.3.1 Discrete Full Bathtub

This case, introduced in Section 3.4 requires that $\varphi_2 = \mathbf{1}_A - \varphi_1$ and uses the Bathtub principle without any constraint. Assume that the range of an scalar input X is $\mathcal{R}_A(X) = \{r_0, ..., r_{n-1}\}$. Then we consider $y_1, y_2 \in \mathcal{R}_A(X)$, and the children defined as

Case I: $y_1 \leq y_2$

$$A_0 \equiv \{w \in A : X[b'](w) < y_1\} \tag{4.3.1}$$

$$A_1 \equiv \{w \in A : y_2 < X[b'](w)\} \tag{4.3.2}$$

$$A_2 \equiv \{w \in A : y_1 \leq X[b'](w) \leq y_2\} \tag{4.3.3}$$

Case II: $y_1 > y_2$

$$A_0 \equiv \{w \in A : X[b'](w) \leq y_2\} \tag{4.3.4}$$

$$A_1 \equiv \{w \in A : y_1 \leq X[b'](w)\} \tag{4.3.5}$$

$$A_2 \equiv \{w \in A : y_2 < X[b'](w) < y_1\} \tag{4.3.6}$$

There are several cases that are potentially confusing, it is crucial to keep the following key ideas in mind in order to resolve some special cases. Recall that the sets $\{w : X[b'] < y_1\}$ $\{w : X[b'] > y_2\}$ can not have probability equal to 0 (i.e. $u_1 > 0$ and $u_2 > 0$, in particular they are not empty.) In the present discrete setting this means that we will require

$$r_0 < y_1 \leq y_2 < r_{n-1} \quad \text{and} \quad r_0 \leq y_2 < y_1 \leq r_{n-1} \tag{4.3.7}$$

Figure. 4.9 shows these constraints and the associated partition, defining the two subsets as $\{w : X[b'] < y_1\}$ $\{w : X[b'] > y_2\}$ in case a) for a given $y_1, y_2 \in R_A(X)$ where $y_1 \leq y_2$ there is no intersection between both then clearly it is possible to define three non empty sets A_0, A_1, A_2. Case b) $y_1 > y_2$ means that there is an intersection between them and also the intersection defines three subsets, also it is interesting to stress case c) that is equivalent to the Haar case where $A_2 = \emptyset$ due to there is no values in $R_A(X)$ that could satisfy the constraints.

4.3.2 Inner Supremum: Best Split Algorithm

The `bestSplit` for the MD case calculates the best partition of a vector random variable X in an atom $A \in \mathcal{A}$ for a given b', by maximizing the inner product $[X, \psi_A]$,

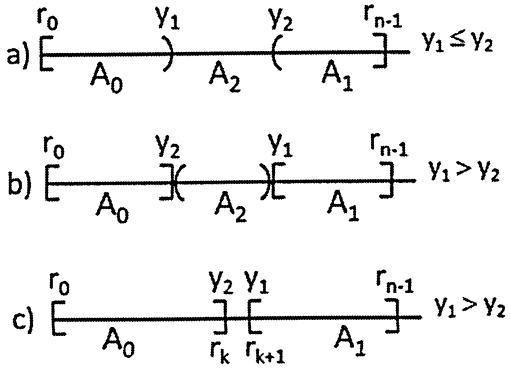


Figure 4.9: Boundaries constraints a) Case I, b) Case II, c) Haar Case

but now the number of children could be more than two, recall that we require the number of children to be greater than one, otherwise there is no split. Then the `bestSplit` should return the best inner product and the best partition, but in order to calculate the children, the values y_1 and y_2 are also needed.

The following picture shows the flow of this function. and the pseudo-code for the

Figure 4.10: Flow chart Bathtub algorithm

`bestSplit` algorithm in the MD case

```
Function bestSplit(X[b'])
  Calc_CDF(X[b'])                 //Calculate the cumulative
  Calc_Exp(X[b'])                 //Calculate the expectation
  For each ri in {r0,..,rn-2}     //Covering the range of x[b']
    u2=1-CDF(ri)                  //Calculate u1
    I2=E(rn-1)-E(ri)              //Calculate the expectation for the last term
    For each rj in {r1,..,rn-1}   //Covering the range of x[b']
      u1=CDF(rj)                  //Calculate u1
      I1=E(rj)                      //Calculate the expectation for the last term
      Calc_Lambda(u1, u2 , I1, I2) //Calculate the value of lambda
      If(lambda>lambda_max)       //Find the maximum value of lambda
        lambda_max=lambda          //Store the maximum value of lambda, y1, y2
        y1=ri
        y2=rj
      End If
    End For
```

```
  End For
  Return(y1,y2 , lambda_max)  //Return y1,y2 and lambda maxima
End Function
```

From the pseudo-code is possible to see that the algorithm's order is $\mathcal{O}(n^2)$ where n is the number of different values taken by $X[b']$ explained at (4.1.17).

4.3.3 Outer Supremum: b' Optimization Algorithm

Practically, this algorithm is the same as the Haar version, the only difference remains in the values that algorithm returns, now as there exist two different values used to split the range, the best split returns both.

```
Function bestVectorSplit(X, A)// X: input vector, A: current atom
  lambda_max = 0              // max inner product set to 0
  While(condition)            // loop on b'
    b'= Create()              // propose a b'
    X[b'] = CreateLC(X,b')    // construct the linear combination
    lambda = bestSplit(X[b']) // find the bestSplit on A of X[b']
    If(lambda>lambda_max)     // evaluate the maximum
      lambda_max = lambda     // actualize the maximum
      bp_max=b'
    End If
    Evaluate(condition)       // evaluate the stop criteria
  End While
  Return(y1,y2, lambda_max, bp_max)  //Return y1, y2, lambda and b' maxima
End Function
```

The algorithm returns the best $\hat{y}_1$, $\hat{y}_2$,the best $\hat{\lambda}$ and the best $\hat{b}'$.

From the pseudo-code is possible to see that the order of the algorithm to calculate $\sup_{\psi_A \in \hat{\mathcal{C}}_A} [X, \psi_A]$ in the MD case is $\mathcal{O}(k\ n^2)$ where k is the total number of iterations, and n was explained at (4.1.17).

4.3.4 General Algorithm Description

The difference between this case and the Haar case resides in the tree structure, in the MD case the number of children could be greater than 2. Then it is a 3-ary tree.

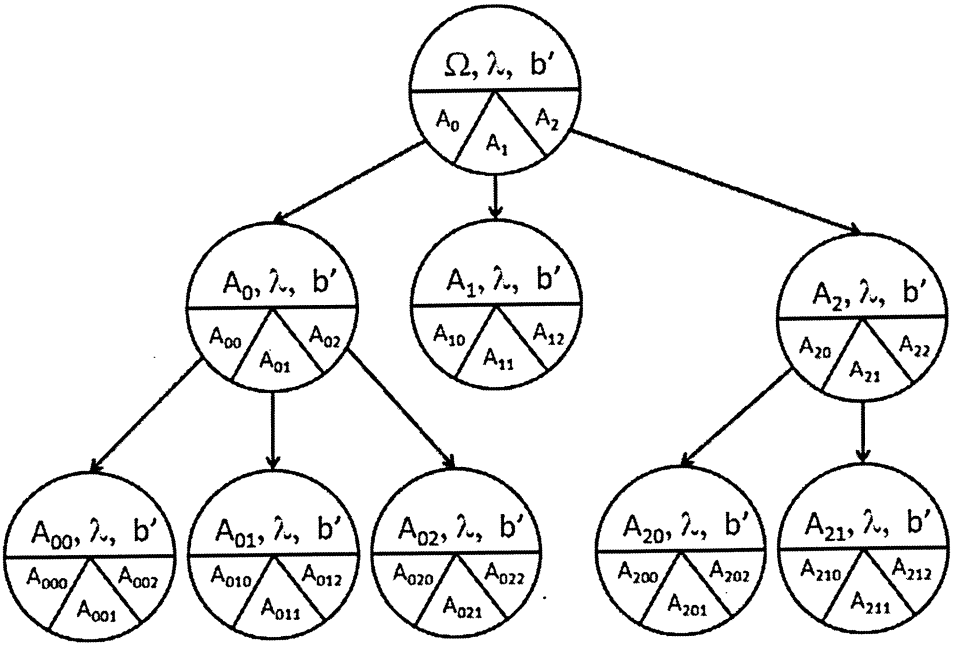


Figure 4.11: MD Tree Description

Figure 4.11 shows the tree generated by the algorithm in three steps, and it shows the node structure composed by the inner product λ, the partition A_0, A_1 and A_2, the b'.

The first step consists in applying `bestVectorSplit` to the vector X on the entire partition Ω and storing the best parameters in a node called root in both $\mathcal{Q}_\Lambda$ and the tree $\mathcal{T}$, this procedure is called `initializeVGS`, Figure.4.5a shows the flowchart of this procedure. Then as there is only one node corresponding to the entire partition Ω in $\mathcal{Q}_\Lambda$, this node is selected and the `bestVectorSplit` is applied over the children A_0, A_1 and A_2, and the first node is removed from $\mathcal{Q}_\Lambda$, then two new nodes are created and stored in the queue and in the tree, now the difference between this algorithm and the Haar version, if $A_2 \neq \emptyset$ another node is created an inserted in both the queue and the tree. The algorithm now selects the node in $\mathcal{Q}_\Lambda$ with greatest inner product and again applies `bestVectorSplit` over its children, removes this node from the queue and inserts its children; it continues recursively iterating until some criteria is reached or until there are no atoms left. This iterative process is called `iterateVGS` and Figure 4.5b graphically shows this process. When the `iterateVGS` finishes the full tree $\mathcal{T}$ is obtained.

and the pseudo code for the `iterateVGS` procedure in the MD case is described

```
Function iterateVGS(X)    // X: input vector
```

```
  While(!emptyQueue){
    // the best split of X in the children  A0 of A
    lambda_max,r_max, bp_max = bestVectorSplit(X,v->A0)
    v0 = createNode(lambda_max,y1,y2, bp_max)
    insertNode(v0)      //insert node v0 into the queue and the tree
    // the best split of X in the children  A1 of A
    lambda_max,r_max, bp_max = bestVectorSplit(X,v->A1)
    v1 = createNode(lambda_max,y1,y2, bp_max)
    insertNode(v1)      //insert node v1 into the queue and the tree
    If(!v->A2=0)
      v2 = createNode(lambda_max,y1,y2, bp_max)
      insertNode(v2)      //insert node v1 into the queue and the tree
    End If
    removeFather(A)     //remove father A only from the queue
  End While
End Function
```

Analyzing the pseudo-code of the procedure `iterateVGS` is possible to see that the order of the algorithm to obtain the full tree is given by

$$\mathcal{O}(N_v\ k\ n^2) \tag{4.3.8}$$

where N_v is obtained in (4.1.19), k is the total number of iterations done by the procedure `bestVectorSplit`, and n was explained at (4.1.17).

4.4 Optimization Techniques

As we have introduced in (3.3.13) we need to find the best b' in the d-dimensional sphere (3.1.6). In order to solve the outer supremum problem we propose four different algorithms.

1. Random Optimization, the initial reason of this method was to find the range of values taken by the cost function. But finally this method worked as well as the others.
2. Quadratic Optimization, using a fixed perturbation on each direction.
3. Standard Simulated Annealing Optimization was implemented.
4. Iterative optimization for b'.

4.4.1 Random Optimization Technique

This algorithm generates random vectors in the unit d-dimensional sphere, by using normal random variables in each direction and normalizing the vector, as we can see in equation (4.4.1) where N_i are random values with normal distribution ($\mu = 0, \sigma = 1$).

$$b' = \frac{1}{\sqrt{\sum N_i^2}} (N_1, N_2, ..., N_d) \tag{4.4.1}$$

The algorithm iterates a number of times and each time we evaluate the cost function and we take the value if it is greater than the maximum. Here is the pseudo code:

```
Function bestVectorSplit(X, A, NI)// X: input vector, A: current atom
                                  // NI: Iterations
  lambda_max = 0                  // max inner product set to 0
  For k = 0 To NI                 // loop on b'
    b' = uniform_rd()             // propose a b'
    X[b'] = CreateLC(X,b')        // construct the linear combination
    lambda = bestSplit(X[b'])     // find the bestSplit on A of X[b']
    If(lambda>lambda_max)         // evaluate the maximum
      lambda_max = lambda         // actualize the maximum
      bp_max=b'
    End If
  End For
```

```
  Return(y1,y2, lambda_max, bp_max)  //Return y1, y2, lambda and b' maxima
End Function
```

4.4.2 Quadratic Optimization Technique

This algorithm uses a quadratic approximation of the cost function in order to maximize the outer supremum of the inner product. We use $b' = (1, 0, 0, 0)$ as initial guess and fixing the all components except the first we can write the following

$$\begin{aligned} z_1 &= F(x_1) = ax_1^2 + bx_1 + c \\ z_2 &= F(x_2) = ax_2^2 + bx_2 + c \\ z_3 &= F(x_3) = ax_3^2 + bx_3 + c \end{aligned}$$

and

$$\begin{aligned} x_1 &= b'_1 - \alpha \\ x_2 &= b'_1 \\ x_3 &= b'_1 + \alpha \end{aligned}$$

where b'_1 is the first scalar component of b' and α is a fix constant perturbation. We can find a, b, c solving the equation system. Once we have calculated a, b we can compute the optimum $x^* = \dfrac{-b}{2a}$ and $z^* = F(x^*)$ then using these values we continue with the next component b'_2.

The pseudo code for this method is described below:

```
Function bestVectorSplit(X, A, NI, alpha)// X: input vector, A: current atom
                                          // NI: Iterations, alpha: perturbation
  lambda_max = 0  // max inner product set to 0
  b'=(1,0,0,0,...)
  For j = 0  To NI
    For i = 0 To d
      x1 = b'[i] - alpha            //perturbations on b'
      x2 = b'[i]
      x3 = b'[i] + alpha
      X[xi] = CreateLC(X,xi)        // construct the linear combinations
      z1 = bestSplit(X[x1])
      z2 = bestSplit(X[x2])
      z3 = bestSplit(X[x3])
      Find( A, B, C )               //solve the linear equations
```

```
      xopt = - B / (2*A)
      X[xopt] = CreateLC(X,xopt)      // construct the linear combination
      zopt = bestSplit(X[xopt])
      if zopt < z1 then zopt=z1 and xopt=x1
      if zopt < z2 then zopt=z2 and xopt=x2
      if zopt < z3 then zopt=z3 and xopt=x3
      b'[i]= xopt
      if zopt>lmax)
        lambda_max=zopt;
        bp_max=b'
      End
    End
  End
```

Where the input parameters are X the input vector, A the actual atom, NI the number of iterations and α a constant perturbation . Recall that d is the number of inputs.

4.4.3 Simulated Annealing Optimization Technique

The Simulated Annealing Technique [15] was implemented as an alternative method to find the maximum value of the outer supremum of the inner product. The pseudo code is shown below. Notice that, as in any heuristic algorithm we keep track of the best solution obtained by the algorithm so far. Although it is computationally intensive for some parameters, the results are better than the results obtained by the previous methods, also with less number of iterations.

```
Function bestVectorSplit(X, A, NI)// X: input vector, A: current atom
                                  // NI: Iterations
  lambda_max = 0  // max inner product set to 0
  b'=(1,0,0,0,...)
  X[b'] = CreateLC(X,b')        // construct the linear combination
  lambda = bestSplit(X[b'])
  e = 1 / ( 1+ lambda  )
  k = 0
  While( k < NI and e > emax )
     b'0 = neighbor( b' )       //find a neighbor
     X[b'0] = CreateLC(X,b'0)  // construct the linear combination
     lambda = bestSplit(X[b'0])
     en = 1 / ( 1 + lambda )
```

```
    If(lambda > lambda_max)  //tracking of the maximum value
       lambda_max = lambda
       bpmax = b'0
    End
    T=temp(k/NI)       //compute the temperature for a given time ratio
    If(random() < exp((e - en)/T)) or en < e) //aceptance probability
       b' = b'0
       e = en;
    End
    k=k+1;
End
```

The function `Neighbor(b')` finds a neighbor for a given b' and `temp(k/NI)` computes the temperature for a given fraction of the time that has been expended.

4.4.4 Iterative optimization for b'

Introduced in (3.3.1) the optimum value of $\hat{b}'$ that maximizes $[X, \psi]$ for a given functions φ_i is described in equation (3.3.14). The key idea is to iteratively compute the best b' and then, based on the result, to calculate the optimal φ_i until some criteria is reached. Figure 4.12 shows a flowchart of this procedure.

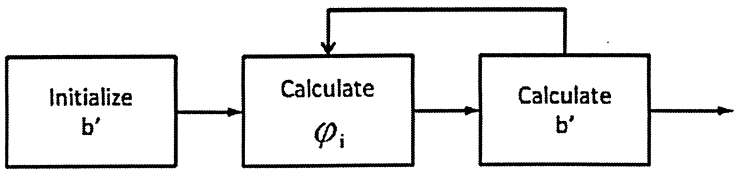


Figure 4.12: Local maximum iterative optimization

We have experimentally verified that this optimization gives rise to a local maximum, and the inner product is always increasing while moving towards this local maximum. Also we have checked that the average number of iterations needed for the inner products to stabilize is three iterations and the maximum number of iterations is less than 20.

Taking advantage of this local optimization, it is possible to address the global optimization problem by using it jointly with any of the previously defined techniques. Actually we are using the random technique presented in Section 4.4.1 but we understand that using Simulated Annealing could be a great improvement for the quality of the maximum. The pseudo code is shown below:

```
Function bestVectorSplit(X, A, NI)// X: input vector, A: current atom
                                  // NI: Iterations
  lambda_max = 0                  // max inner product set to 0
  lambda_local=0                  // local maximum set to 0
  For k = 0 To NI                 // loop on b'
    b' = uniform_rd()             // propose a b'
    X[b'] = CreateLC(X,b')        // construct the linear combination
    lambda = bestSplit(X[b'])     // find the bestSplit on A of X[b']
    If(lambda>lambda_max)         // evaluate the maximum
      lambda_max = lambda         // keep track of the maximum
      bp_max=b'
    End If
    If(lambda>lambda_local)       // evaluate the local maximum
      lambda_local=lambda
      bestBp(Ai)                  //Ai obtained from bestSplit
    Else
      lambda_loca=0
      b' = uniform_rd()           // propose a new b'
    End
  End For
  Return(y1,y2, lambda_max, bp_max)  //Return y1, y2, lambda and b' maxima
End Function
```

This method is the best, comparing quality and speed, with respect to the others. For a review of the performance of the algorithm see Section 7.1.5.

Chapter 5

Application to Image Compression

Up to the present Chapter we have analyzed the VGS algorithm from different points of view: mathematical, discrete representation and software implementation. We will now describe the VGS algorithm for a specific application, the VGS algorithm will be used to perform image compression.

An important property of most images is that there exist a *spatial correlation* among nearby pixels. From the Human Visual System (HSV) there is some *irrelevant information* related to the high frequencies or to the color response that can be removed or reduced such that a person may not notice the difference.

There is also another important correlation, the *temporal correlation* that occurs between two consecutive frames in a video sequence, in order to be a continuous smooth video the number of frames per second should be high enough to capture most of the events in typical recording situations. Therefore the difference between two consecutive frames will be practically negligible.

A typical encoder is composed of an encoding transform, a quantization block and an entropy encoding block as is shown in Figure 5.1

The redundancy correlation corresponding to the spatial and temporal correlations, in some algorithms, can be performed by the encoder transform, therefore it could appear before the quantization block. The role of the encoder function is to transform the information into a space where the information could be better classified from the relevant point of view. The quantizer reduces the number of bits needed to encode the transformed coefficients, it can be applied to each single coefficient (scalar quantization) or on a group (vector quantization). Finally the entropy encoder block encodes the quantized coefficients using Huffman [10] or arithmetic code. The decoder performs the same operations as the encoder but in reverse order.

Assuming that we have applied the VGS algorithm using a specific input vector,

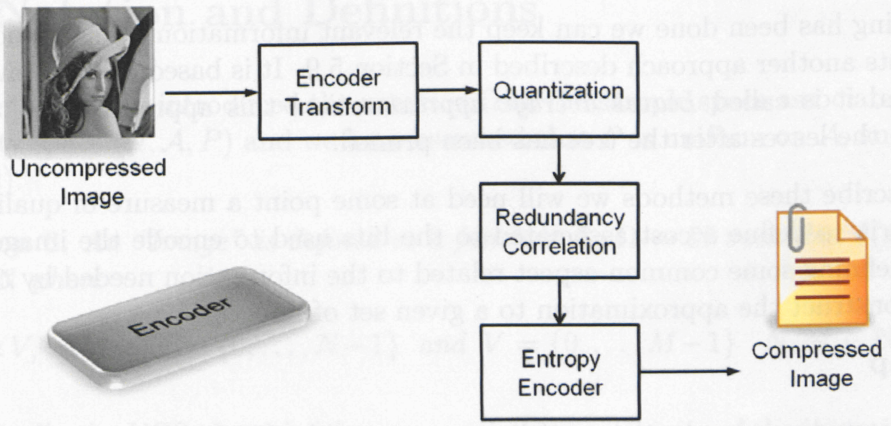


Figure 5.1: Typical lossy encoder

a full tree $\mathcal{T}$ and the associated partition $\Pi_{\mathcal{T}}(\Omega)$ are obtained. Denoting with $\hat{X}$ the vector approximation to X given by this deep VGS tree, we have $||X - \hat{X}|| \equiv \sqrt{[X - \hat{X}, X - \hat{X}]} \approx 0$. The next step is to perform a transform compression, this involves pruning the tree nodes until some stopping criteria is reached. To perform this task several different approaches could be used.

There are two different approaches to pruning used in this thesis. On the one hand, for a fixed vector error level ϵ_V, we can approximate X up to the error ϵ_V. This approximation is a vector approximation as it uses the norm in the space $L^2(\Omega, \mathbb{R}^d)$. If X_n denotes the approximation obtained pruning the full tree, we then will have $||X - X_n|| \leq \epsilon_V$, this vector approximation will provide a certain error level for the components i.e. $||X[i] - X_n[i]||$ (notice that, in this second instance $||\ ||$ denotes the norm in $L^2(\Omega, \mathbb{R})$) . On the other hand, for a fixed scalar error level ϵ_S, it is possible to generate scalar approximations $X_n[i]$, where now $n = n(i, \epsilon_S)$, for each component $X[i]$ in such a way that $||X[i] - X_n[i]|| \leq \epsilon_S$ for each $i = 1, \ldots, d$.

These two different points of view will be called the *Vector approximation* and the *Scalar approximation*, they are explained with some detail in Section 3.8. To summarize:

Vector approximation: this approach uses the largest inner products $[X, \mu_k]$. It does include as many of them $k = 0, \ldots, n-1$ as is necessary to obtain the desired error level $||X - \sum_{k=0}^{n-1} [X, \mu_k]\ \mu_k|| \leq \epsilon_V$.

Scalar approximation: this approach uses the largest scalar inner products $[X[i], \psi_{A,s}]_1$, where $i = 1, \ldots, d$, A ranges over all the nodes in the tree and the subscript S indicates that we are dealing with a scalar VGS function. All these numbers $[X[i], \psi_{A,s}]_1$ are sorted and the largest are kept until the criteria $||X[i] - \sum_{k=0}^{n(i)-1} [X[i], \psi_{A_k,s}]_1\ \psi_{A_k,s}|| \leq \epsilon_S$ are satisfied for all $i = 1, \ldots, d$.

Once the pruning has been done we can keep the relevant information at each node. There also exists another approach described in Section 5.9. It is based on the leaves information and it is called *Leaves average approximation*, this approach uses the information on the leaves after the tree has been pruned.

In order to describe these methods we will need at some point a measure of quality and some criteria to define a cost associated to the bits used to encode the images. We will start defining some common aspect related to the information needed by the decoder to reconstruct the approximation to a given set of images.

Partition Map

As we have indicated previously, the partition constructed by VGS is data dependent and it needs to be encoded entirely. This information is called the "Partition Map" (PM). The partition map is described in Section 5.2.

Significance Map

The tree structure required to reconstruct the input vector is called the "Significance Map" (SM), it contains the information on the number of children and depending on the method selected also contains the vector inner product, the scalar inner products or the average values. We will analyze this map in Section 5.3.

Quantization Map

The inner products stored or the average values in each tree node should be quantized in order to reduce the number of bits needed to store them. This information is called the "Quantization Map" and is described in Section 5.4.

5.1 Notation and Definitions

In chapter 3 we have introduced the notation Ω as the sample space associated to the probability space $(\Omega, \mathcal{A}, P)$ and we have assumed that P is uniform on $\mathcal{A}$, and we set $\mathcal{A} = \mathcal{P}(\Omega)$.

Definition 6. *An "Image" is defined as a function* $I : \Omega \to \mathbb{N}$ *such that* $I(x,y) = v$ *and* $v \in \mathbb{N}$ *where*

$$\Omega = U \times V, \text{ where } U = \{0,\ldots,N-1\} \text{ and } V = \{0,\ldots,M-1\} \quad N, M \in \mathbb{N} \quad (5.1.1)$$

Practically the VGS algorithm has no restriction on the size of the images, but for simplicity we will consider $N = M$. Also, we will just consider with 256-gray scale images with 8 bits/pixel.

Let us consider a vector valued random variable $X : \Omega \to \mathbb{N}^d$ where $X[i]$ (the i-th scalar component of X) is an image. Notice that all input images share the same domain Ω, in particular all images have the same size.

Recall that the VGS algorithm has no restriction about the geometry of the input vector, in particular the algorithm can be applied to images by using the following transformation

$$H(u) = I(x,y) \quad \text{where} \quad u = x + N\left[\frac{y}{N}\right] \quad (5.1.2)$$

where [ ] is the integer part function, and $H : W \to \mathbb{N}$ and $W = \{0,\ldots,M \times N - 1\}$ The inverse is also possible

$$I(x,y) = H(u) \quad \text{where} \quad y = \left[\frac{u}{N}\right] - 1 \quad \text{and} \quad x = u \mod (N+1) \quad (5.1.3)$$

The Greek letter λ will be used throughout this Chapter to denote inner products. Depending on the context, these inner products will be of the form $[X, \psi_A^{(0)}]$or $[X, \psi_A^{(1)}]$ or $[X[i], \psi_{A,s}^{(0)}]_1$ or $[X[i], \psi_{A,s}^{(1)}]_1$. Variations on this type of notation, mainly used in the diagrams, should be self explanatory.

5.2 Partition Map ($\mathcal{M}_\Pi$)

Definition 7. *A function $\mathcal{M}_\Pi : \Omega \to \mathbb{N}$ is called a "Partition Map" if satisfies*

$$\mathcal{M}_\Pi(w) = v_k \quad \forall\, w \in A_k,\ A_k \in \Pi(\Omega) \text{ and } k = 1, \ldots, n \tag{5.2.1}$$

and

$$\text{if } k \neq j \ \Rightarrow\ v_k \neq v_j \tag{5.2.2}$$

where $v_k \in \mathbb{N}$, $\Pi = \Pi(\Omega)$ is a finite partition of Ω and $n = |\Pi(\Omega)|$ is the number of elements in the partition. Also $\Omega = \bigcup_{i=1}^{n} A_k$

The Partition Map is created using the tree $\mathcal{T}$ after `compressTree` is applied, as follows: if a node is included then the atom associated with this node should be split in two or more children, otherwise it remains without change. To illustrate see Figure 5.2, it is a full tree obtained after three iterations, the partition associated to the full tree is shown in Figure 5.3a), but if we only select a few nodes, in this case $\{1, 3, 6\}$

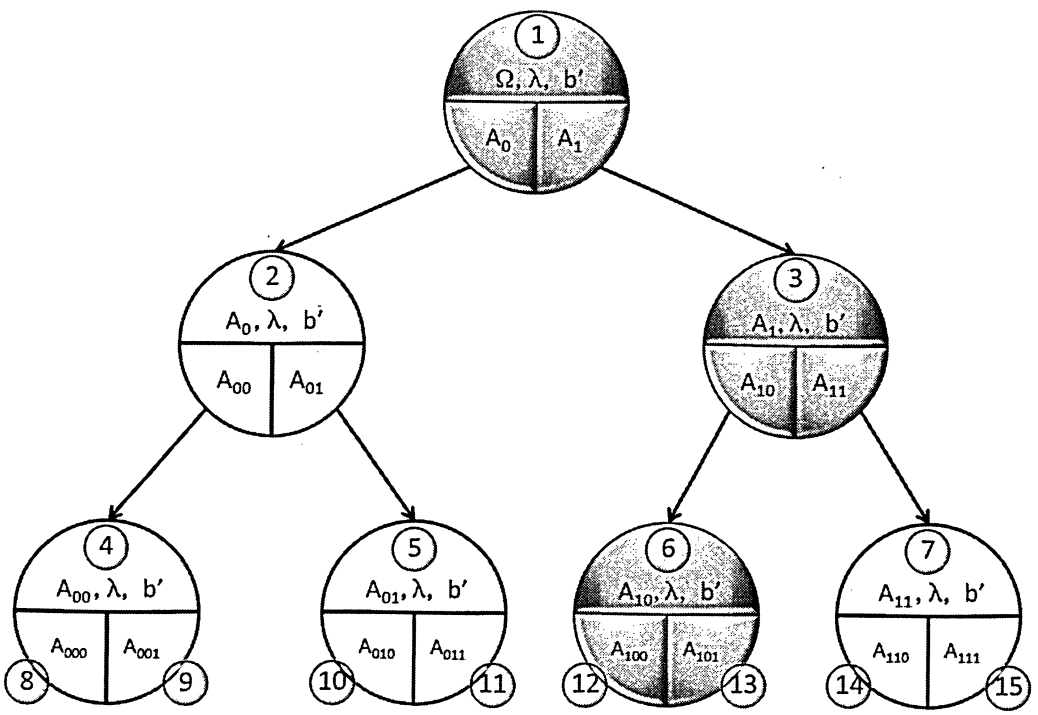


Figure 5.2: Full tree with selected nodes

the resulting partition is shown in Figure 5.3b).

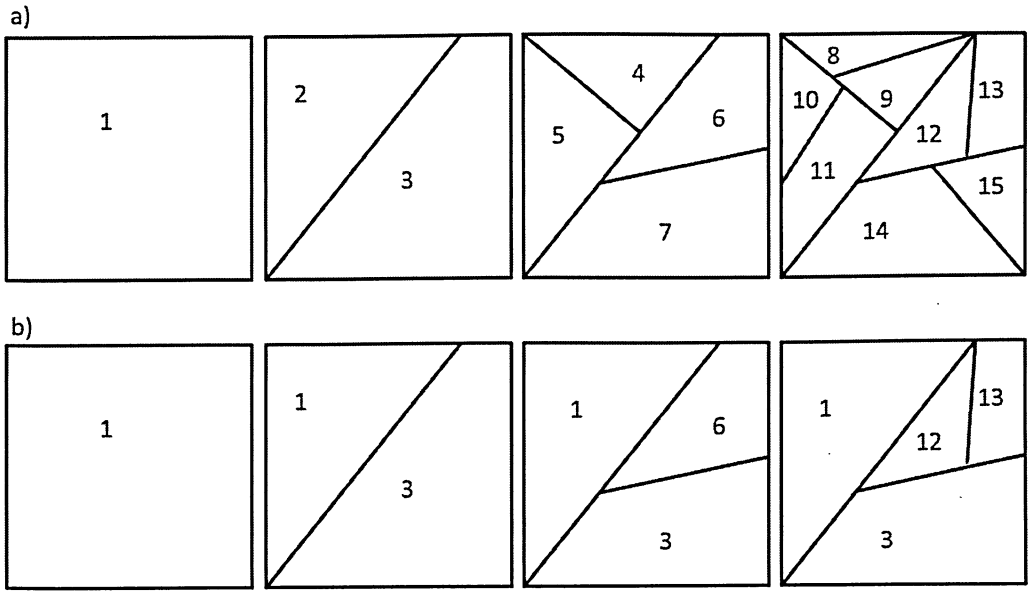


Figure 5.3: a) Partition using the full tree, b) Partition using the compressed tree

Notice that although nodes 12 and 13 are not included in the tree, they are the split of atom A_{10} in A_{100} and A_{101} where the input signals are constant, this is the reason of why we take those atoms in consideration when we construct the common partition. Also notice that node 2 is not included in the selection then node 1 is not split in this region, unless a descendant of node 2 were included.

Remark 20. *The partition map share the same domain as the input images, and the maximum number of atoms is equal to the number of pixels. Then if images has* $256 \times 256 = 65536$ *pixels the maximum size of the partition map is* 65536 *values, but now the range of this values is not anymore* $[0, 255]$*, it is* $[0, 65535]$ *therefore the maximum number of bits (without compression) is 16 bits twice the number of bits needed to encode* 256 *values. Therefore, the maximum number of bits to store the partition map, (for a given set of images) is bounded from above by the bit cost of encoding two images.*

Next we will describe the pseudo code for the `partitionMap` procedure, which is a recursive function that visits nodes using the preorder traversal method. It starts with $\mathcal{M}_\Pi(w) = 0 \ \ \forall\ w \in \Omega$, `TreeNode` is initialized to the root node of $\mathcal{T}$ and `iNode=1`.

```
Procedure partitionMap(PM, TreeNode, iNode)
   IF(TreeNode->Selected)
     For w In TreeNode->A
        PM(w) = iNode
```

```
        End For
      End If
        partitionMap(PM, TreeNode->A0, iNode+1)
        partitionMap(PM, TreeNode->A1, iNode+2)
End Procedure
```

5.2.1 Reordering Partition Values

The partition map could be interpreted as an image even though the values assigned to the atoms are not related. Technically, it is not an image because the values in the range of the partition map could be greater than 256. Although it seems to be difficult to reduce the number of values without losing information, there exists the possibility to reorder the values of the atoms in the partition map in such a way that if the distance between two different atoms is small then its corresponding values should be near too; then we need to find a way to measure a distance between atoms. There are different methods to carry out this task. The following is one such method:

For a given atom $A_k \in \Pi(\Omega)$ we compute the average of the input set in this atom

$$V_k = \frac{1}{d\ |A_k|} \sum_{i=0}^{d} \sum_{w \in A_k} X[i](w), \tag{5.2.3}$$

now using V_k is possible to reorder the values associated to each A_k. Defining the sorted set

$$\{V_{h_1}, V_{h_2}, \ldots, V_{h_n}\} \quad \text{such that} \quad V_{h_1} \leq V_{h_2} \leq \cdots \leq V_{h_n} \tag{5.2.4}$$

where n the number of elements in $\Pi(\Omega)$, then

$$\mathcal{M}_\Pi(w) = k \quad \forall\ w \in A_{h_k},\ k = 1, \ldots, n \tag{5.2.5}$$

Figure 5.4a) shows a set of two images of 128x128 pixels, Barbara and Lena, where the VGS algorithm was applied, and the reorder of the resulting partition is shown in Figure 5.4b), of course the number of atoms in the partition was equal to 256.

5.2.2 Entropy encoding

A first approach to compress the partition map is using lossless methods like Huffman or arithmetic code directly without taking into account any geometric relationship.

We will call "symbol" to the value assigned to each atom in $\Pi(\Omega)$. Then the average number of bits needed to encode each symbol is given by

$$H_{\mathcal{M}_\Pi} = -\sum_{i=0}^{n} p_i\ \log_2\ p_i \quad \text{where} \quad p_i = \frac{|A_i|}{|\Omega|} \tag{5.2.6}$$

a) b)

Figure 5.4: a) Barbara & Lena, b) Reordered Partition Map

and $N_s = |\Omega|$ is the number of pixels in Ω and then p_i becomes the relative frequency of each symbol.

The theoretical cost associated to the partition is given by

$$C_{\mathcal{M}_\Pi} = H_{\mathcal{M}_\Pi} \times N_s \tag{5.2.7}$$

Using the previous example shown in Figure 5.4 and an approximation with a PSNR = 38.38 db and using the reorder of the partition, the relative frequency of the symbols is shown in Figure 5.5, where $H_{\mathcal{M}_\Pi} = 7.83$ bpp and $C_{\mathcal{M}_\Pi} = 16384 \times 7.83 = 128439.67$ bits, approximately 97.99% of the maximum when $H_{\mathcal{M}_\Pi} = \log_2 256 = 8$.

The previous result shows that there is no compression at all, and it becomes obvious when we check in Figure 5.5 that the relative frequency of the symbols is quasi uniform.

5.2.3 Spatial correlation

There are several approaches to take advantage of any spatial correlation; we have found that one of the best techniques is the following:

We suppose that there is a spatial correlation between pixels by assuming that one column is similar to the next, if $w = (x, y)$ then

$$\mathcal{M}_\Pi(x+1, y) - \mathcal{M}_\Pi(x, y) \sim 0, \tag{5.2.8}$$

the relation between lines is also true. Then, it is advantageous to store the difference of the columns instead of the original values.

Going back to the same example as before, shown in Figure 5.4, but now considering

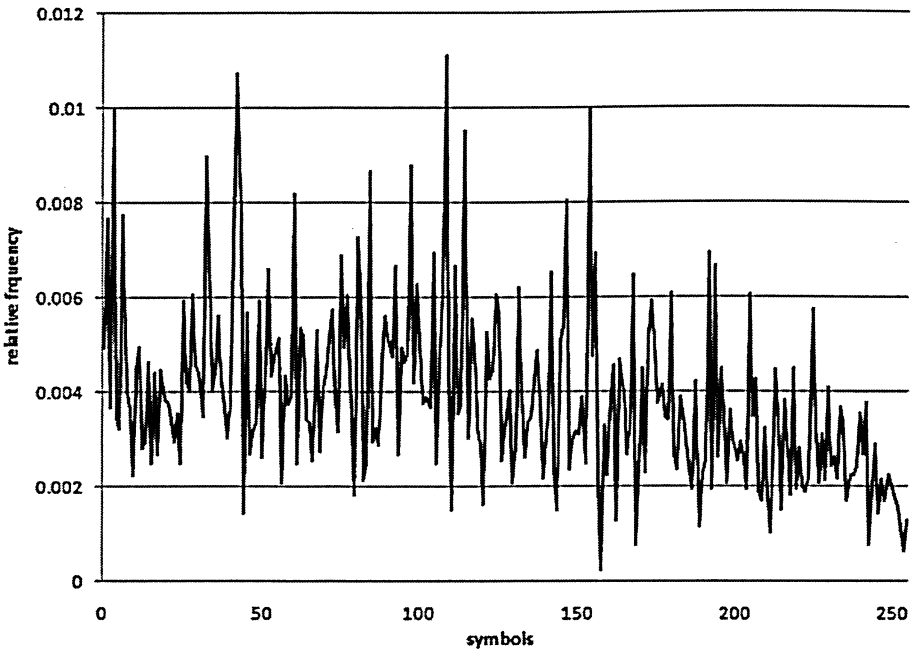


Figure 5.5: Relative frequency

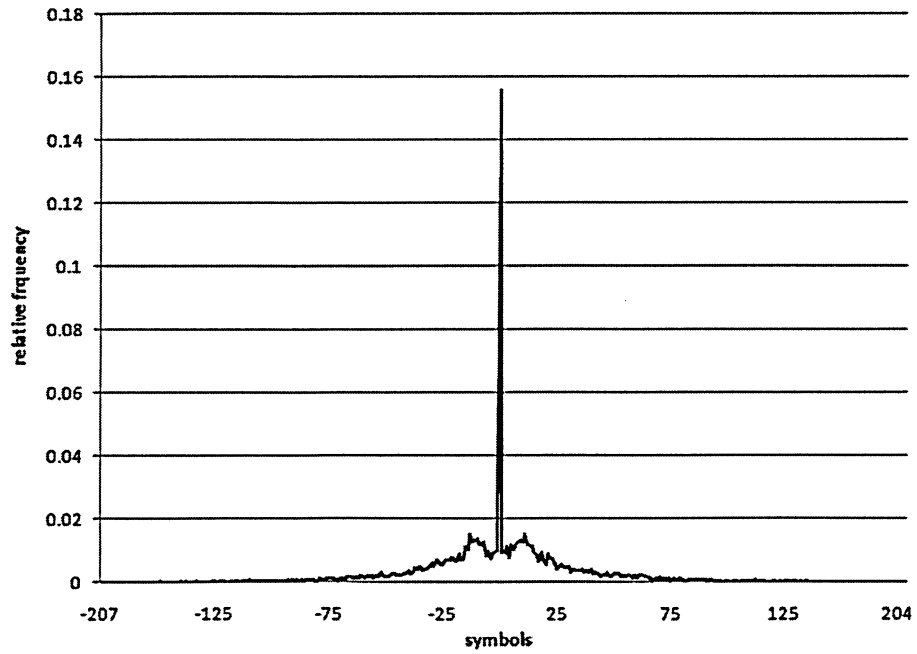


Figure 5.6: Relative frequency using spatial correlation

the differences. The relative frequency of the symbols is shown in Figure 5.6, where it is possible to appreciate that most of the symbols are equal to zero.

The results now are better than before, $H_{\mathcal{M}_\Pi} = 6.8$ bpp and $C_{\mathcal{M}_\Pi} = 16384 \times 6.8 = 111423.84$ bits, approximately 85.03% of the maximum when $H_{\mathcal{M}_\Pi} = \log_2 256 = 8$, it means a 15% of compression.

Although it seems to be not a good compression rate, we have to take in consideration that it is lossless compression, and also if you apply the JPG2000 to the partition image, with a PSNR = 45 db, the size will be approximately equal to the 60% of the image. Therefore the lossless compression using differences seems to work well.

There exists also the possibility to apply a lossy compression algorithm to the partition, but the results showed that the algorithm is very sensible to a change in the partition values. Therefore because the distortion has to be very small, the compression rate for a lossy method is practically the same as the lossless method.

5.3 Significance Map ($\mathcal{M}_S$)

For a given tree, as in Figure 5.7, we assume in general that we have a function called `compressTree` that selects a number of nodes based on some criteria, a typical result is shown in Figure 5.8.

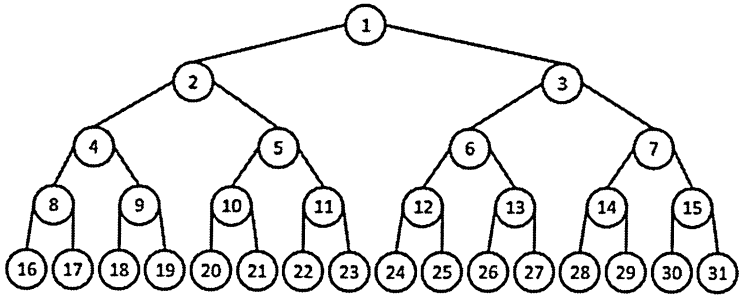


Figure 5.7: Full tree

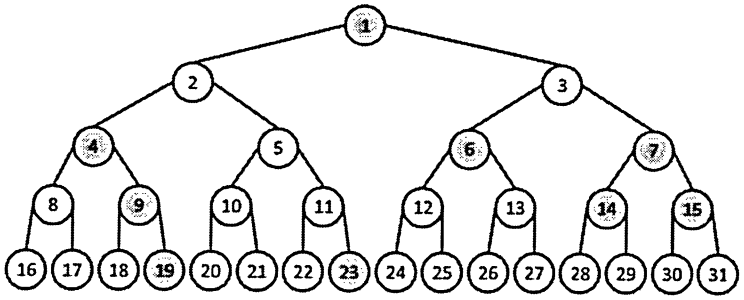


Figure 5.8: Compressed tree

The scalar approximation and the vector approximation need a tree for the reconstruction.

The significance map stores the tree information and each node contains the information associated to the approximation on each atom, e.g. inner products, b'.

Notice that the significance map also needs to include links from nodes to the partition encoded by the partition map. This is the main difference with other methods that use trees to encode inner products. This case is much more difficult due to the fact that the encoder should encode the inner products and the links to the atoms at the same time.

As we can see in Figure 5.8, if a node is selected we *do not require the ancestors to be included.* Most approaches at this point [1], [19] , assume that in sparse isotropic

wavelet representation, with high probability a significance node does not have any significance children nodes, using the zero-trees proposed by [22]. Of course we agree with the previous statement, but in our vector case the extension of that proposition is not clear and also the problem to include a node and not its ancestors can be solved without including much more extra information or introducing any extra computational cost. Due to the complexity of the algorithm, we will describe it by means of an example.

We propose three different types of symbols to encode the tree, and they will be used to create a string of symbols (this string will be called the *significant string* and denoted with $\mathcal{S}$). These symbols are:

- Q : Active node
- V : Link to the partition
- D : Dummy node

We start visiting tree nodes using a preorder traversal method.

1. Node 1 is visited an as it is selected we label it with a "Q" and 2 because of its 2 children.
2. Node 2 is visited, it is not selected and it can not cover the right branch, leaded by node 5, then we label it with a "V", also with 2 children.
3. Node 4 is visited and labeled with a "Q", because it is a selected node with 2 children.
4. Node 8 is visited, neither it is selected nor its descendant, the we introduce a "V" and we stop descending.
5. Node 9 is visited and and labeled with a "Q", because it is a selected node with 2 children.
6. Node 18 is visited and labeled with a "V".
7. Node 19 is visited and labeled with a "Q", because it is a selected node with 2 children, and as there is no children two extra symbols "V" are included. This node could be see as a terminator symbol.

The algorithm continues until the following string is constructed

Figure 5.9 shows the encoded string and Figure 5.10 shows the decoded tree that is equivalent for reconstruction to the original tree. The number of symbols proposed

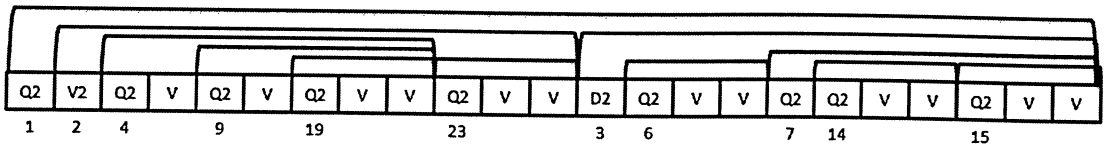


Figure 5.9: Encoded string

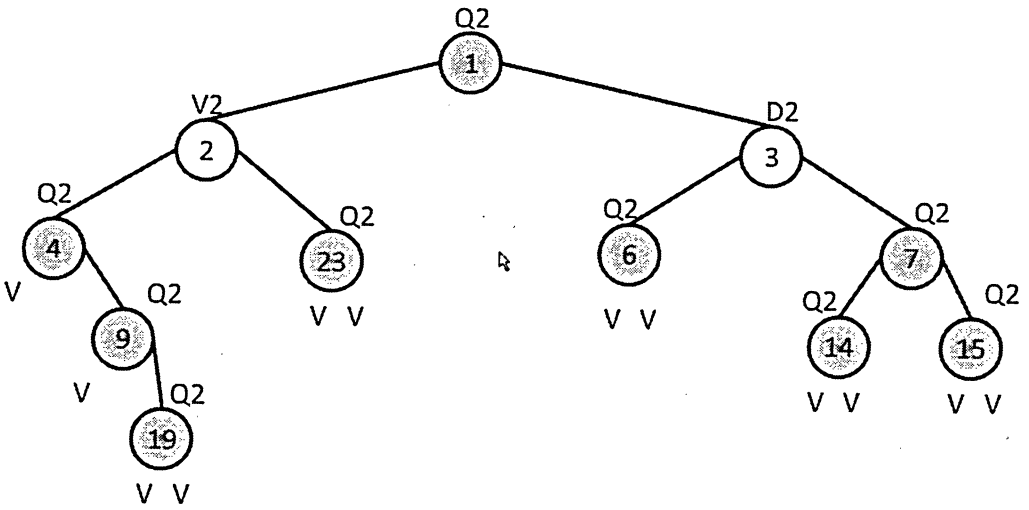


Figure 5.10: Equivalent decoded tree

is three, but if we associate the number of children to the symbol we can check that the sequence of symbols "{Q2, V, V}" has a high probability then we can introduce another new symbol called Q2VV, this is the analogous of the zero tree symbol introduced in [22]. Also we can go one step further and check the histogram of the symbols to find the best number of symbols needed to encode the string.

5.3.1 Entropy encoding

Recall that the total number of different symbols (for our present model) that could be included in the encoded string are given in the following list: $\{Q2, Q3, V, V2, V3, D2, D3\}$ and also we suggest the possibility to add the $Q2VV$ and $Q3VVV$ as symbols with high probability.

Definition 8. *A function $\mathcal{M}_S : \mathcal{S} \rightarrow \mathbb{Z}$ is called a "Significance Map"*

and

$$\mathcal{S}_k = \{s \in \mathcal{S} : \mathcal{M}_S(s) = k \text{ and } k \in \mathbb{Z}\} \tag{5.3.1}$$

also we define the symbol set $\mathcal{J}_S$

$$\mathcal{J}_S = \{\mathcal{S}_k \subset \mathcal{S} : \mathcal{S}_k \neq \emptyset\} \tag{5.3.2}$$

Using entropy encoding we find that

$$H_{\mathcal{M}_S} = -\sum_{\mathcal{S}_k \in \mathcal{J}_S} p_k \log_2 p_k \quad \text{where} \quad p_k = \frac{|\mathcal{S}_k|}{|\mathcal{S}|} \tag{5.3.3}$$

where p_k is the relative frequency of each symbol.

The theoretical cost associated to the significance map is given by

$$C_{\mathcal{M}_S} = H_{\mathcal{M}_S} \times |\mathcal{S}| \tag{5.3.4}$$

For the example above using the encoded string in Figure 5.9,

$$\mathcal{S} = \{Q2, V2, Q2, V, Q2, V, Q2VV, Q2VV, D2, Q2VV, Q2, Q2VV, Q2VV\} \tag{5.3.5}$$

The average number of bits $H_{\mathcal{M}_S} = 2.038$ bits per symbol and the theoretical total cost is equal to $C_{\mathcal{M}_S} = 26.49$ bits, using the standard coding without taking in consideration the entropy, is equal to 30.18 bits it means that we have saved 12.3%. In a real application the number of nodes in a compressed tree are near to 1000 then the number of bits needed to store such a tree is close to 2000.

We remark that the cost associated to the significance map is, relatively speaking, the lowest cost when compared with cost to encode the partition map or the quantization map. In some instances the cost of the significance map does increase (as we elaborate latter).

Remark 21. *For some cases like the scalar Haar case approximation, extra information is needed, the symbol Q represents a link between the node and the information needed for the reconstruction. We will describe this problem separately in Section (5.5).*

5.4 Quantization Map ($\mathcal{M}_Q$)

The quantization map stores the information (quantized) needed for the reconstruction in each node. Different cases need different information, the idea is to quantize this information and then to use an entropy encoding algorithm to store it with minimum number of bits.

We will describe the special Haar case using a scalar approximation to exemplify, but each case has its own quantization map and will be described in the corresponding section.

Haar Case: scalar approximation

The information needed in this specific case at each node was introduced in Section 3.8.1 and it consists of the scalar inner products $[X, \psi_s]_1$ and the partition that is encoded by the partition map. Therefore, the information to be encoded is composed by d real numbers, where d is the number of input images. It could be encoded using 4 bytes for each component but the idea is to reduce the number of bits used to encode this information.

We propose the following scheme in order to minimize the number of bits to be encoded. As we are working with real numbers, it is not possible to use entropy

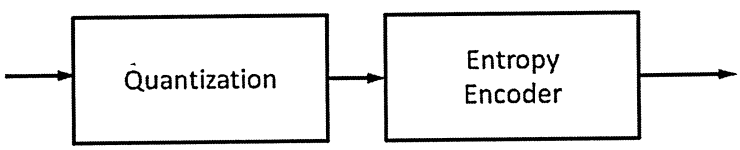


Figure 5.11: Equivalent decoded tree

encoding without previously using a quantization method. The two techniques can be combined and performed simultaneously as in the case of the arithmetic coding, see [22], [21].

Let us call $\lambda_k = [X[i], \psi_{A,s}]_1$ to the largest inner products kept after pruning a full tree by means of the scalar approximation.

5.4.1 Quantization

We have verified that the best quantization technique for our algorithm is the uniform quantization defined as follows

$$\mathcal{V}(\lambda_k) = \left\lfloor \frac{\lambda_k}{c} \right\rfloor \times c \quad \text{and} \quad c > 0 \tag{5.4.1}$$

In a real application (from the video image set using a full tree) $\lambda_k \in (0.0001, 200)$, Figure 5.12 shows the values of λ_k sorted by $|\lambda_k|$

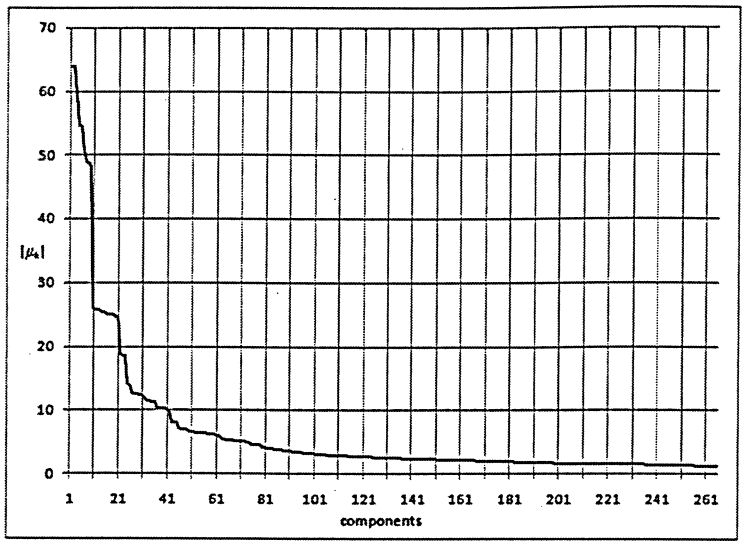


Figure 5.12: Scalar inner products distribution

5.4.2 Entropy Encoding

As we have mentioned before in order to apply the entropy encoding we have to use the quantization function defined in (5.4.1). Now defining

$$Q = \{\mathcal{V}(\lambda_i) : \lambda_i \in \mathcal{Q}_n\}, \tag{5.4.2}$$

we can define the quantization map as follows

Definition 9. *A function $\mathcal{M}_Q : Q \to \mathbb{Z}$ is called a "Quantization Map"*

Also we can define the set of al values of Q equal to k as

$$Q_k = \{q \in Q : \mathcal{M}_Q(q) = k \text{ and } k \in \mathbb{Z}\} \tag{5.4.3}$$

and then defining the symbol set as

$$\mathcal{J}_Q = \{Q_k \subset Q : Q_k \neq \emptyset\}, \tag{5.4.4}$$

we can use entropy encoding to find the average bit per symbol

$$H_{\mathcal{M}_Q} = -\sum_{Q_k \in \mathcal{J}_Q} p_k \log_2 p_k \quad \text{where } p_k = \frac{|Q_k|}{|Q|} \tag{5.4.5}$$

where p_k is the relative frequency of each symbol in $\mathcal{J}_Q$. Then the theoretical total cost associated with the quantization map can be computed as follows

$$C_{\mathcal{M}_Q} = H_{\mathcal{M}_Q} \times |Q| \tag{5.4.6}$$

In order to illustrate the performance of our algorithm using a uniform quantization, we compare the distortion produced using different values of c and a full tree taken from the video image set. Figure 5.13 shows and the relation PSNR and c, on the video test set. The PSNR is calculated using the maximum number of components different from zero.

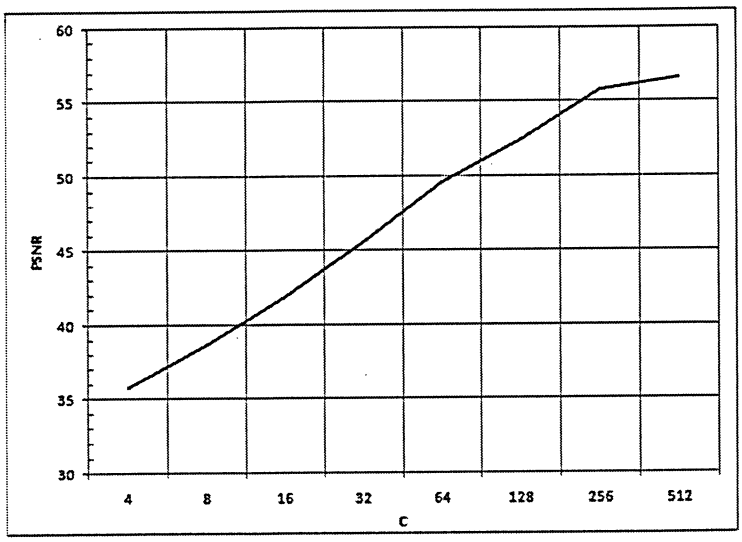


Figure 5.13: PSNR vs. c, Minimum distortion

5.5 Haar Case: Scalar Approximation

In this section we will describe the cost, in term of bits, associated with the scalar approximation for the Haar case. As we have seen before Section 3.8.1 the information needed for the reconstruction in this special case is: the scalar inner products, the partition and the tree.

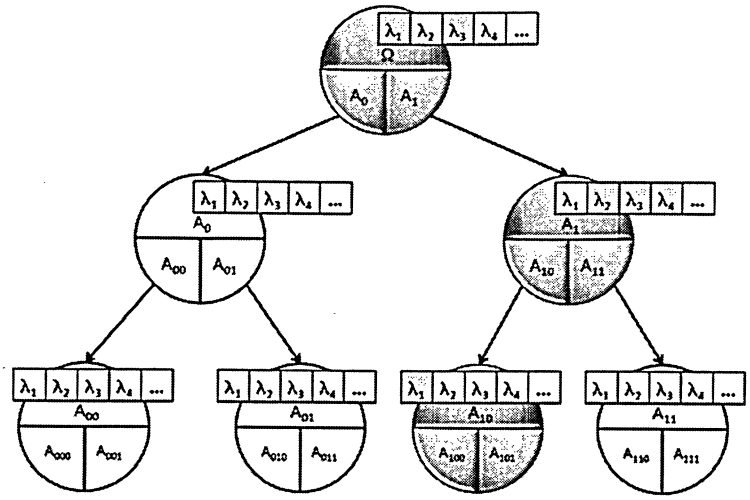


Figure 5.14: Haar case tree for the scalar approximation

Figure 5.14 shows the information of each node, the `compressTree` algorithm selects a node if at least one scalar product λ_i is needed in the node. Such information is needed for reconstruction. Therefore, it is crucial to store this information using an efficient algorithm.

5.5.1 Indices information

The indexing information can be encoded using three different approaches. The first approach uses d bits to encode whether a inner product is included or not. The second approach uses an index header for each inner product included and the third approach uses a special null character to identify when a scalar inner product is not included. Figure 5.15 shows examples of these three approaches for a given sequence of scalar inner products $\{\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5\}$ where only $\{\lambda_2, \lambda_4\}$ are needed. Then for a given node n the associated cost of each model is calculated as follows

- Case a)

$$C_{I_n} = d + k\ H_{\mathcal{M}_Q} \tag{5.5.1}$$

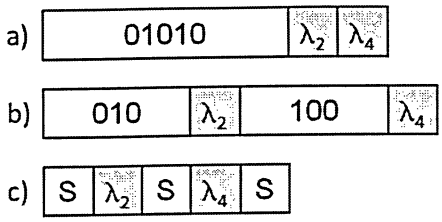


Figure 5.15: a) Binary encode, b) Indexing encode, c) Special character

- Case b)

$$C_{I_n} = k \ \log_2 d + k \ H_{\mathcal{M}_Q} \tag{5.5.2}$$

- Case c) We assume that the special null character has $H_{\mathcal{M}_Q}$ bits

$$C_{I_n} = d \ H_{\mathcal{M}_Q} \tag{5.5.3}$$

where d is, as usual, the number of inputs, $H_{\mathcal{M}_Q}$ is the average bits per scalar inner product, and k is the number inner products being used at node n. It is possible to evaluate a priori which method is the best for each node and then adding two bits to the header of the node so the decoder can use the correct method. Then

$$C_I = \sum_n C_{I_n} \tag{5.5.4}$$

This data seems to be superfluous but in the following example we can see that plays an important role, suppose $H_{\mathcal{M}_Q} \approx 4$ bits, and a given input set with 16 elements, using case b), the indices takes 4 bits that is the same as the information needed to encode the inner products.

5.5.2 Total Cost for the Scalar Haar Approximation

The total cost C_T for this case is calculated as

$$C_T = C_{\mathcal{M}_\Pi} + C_{\mathcal{M}_S} + C_I \tag{5.5.5}$$

Where $C_{\mathcal{M}_\Pi}$ is the cost associated with the partition, $C_{\mathcal{M}_S}$ is the cost associated with the tree, and C_I is the indexing cost. The cost associated with the quantized coefficients $C_{\mathcal{M}_Q}$ (see section 5.4), is included in C_I. We will see several examples in Chapter 7, related to the behavior of the algorithm and the theoretical number of bits needed to store the approximation.

5.6 Haar Case: Vector Approximation

The vector approximation for the Haar case uses the best $\hat{b}'$ (see (3.3.1) and ()), the vector inner product and the partition information to perform the approximation. The best $\hat{b}'$ is a vector defined by the following equation:

$$\hat{b}'_i = \frac{\frac{1}{u_2}\int_A X_i(w)\ \varphi_2(w)\ dP(w) - \frac{1}{u_1}\int_A X_i(w)\varphi_1(w)\ dP(w)}{\sqrt{\sum_{k=1}^{d}\left(\frac{1}{u_2}\int_A X_k(w)\ \varphi_2(w)dP(w) - \frac{1}{u_1}\int_A X_k(w)\ \varphi_1(w)dP(w)\right)^2}}. \tag{5.6.1}$$

where in this case $\varphi_1 = \mathbf{1}_{A_0}$ ($A_1 = A \setminus A_0$) and $\varphi_2 = \mathbf{1}_{A_1}$ and $u_1 = P(A_0)$ and $u_2 = P(A_1)$. Then

$$\frac{1}{u_1}\int_A X_i(w)\ \varphi_1(w)\ dP(w) = \frac{1}{P(A_0)}\int_{A_0} X_i(w)\ dP(w) = \mathbf{E}_{A_0}(X_i), \tag{5.6.2}$$

which is the expected value of X_i relative to the atom A_0. Similarly,

$$\frac{1}{u_2}\int_A X_i(w)\ \varphi_2(w)\ dP(w) = \mathbf{E}_{A_1}(X_i). \tag{5.6.3}$$

Therefore, the value of the best $\hat{b}'_i$ is given by the normalized difference of two expected values $\mathbf{E}_{A_1}(X_i) - \mathbf{E}_{A_0}(X_i)$. In order to store the values of the best $\hat{b}'_i$, we only need to store the result of such difference because the normalization can be done a posteriori. Now let us define

$$\Delta_i = \mathbf{E}_{A_1}(X_i) - \mathbf{E}_{A_0}(X_i) \tag{5.6.4}$$

then

$$\hat{b}'_i = \frac{\Delta_i}{\sqrt{\sum_{k=1}^{d}[\Delta_k]^2}} \tag{5.6.5}$$

5.6.1 Quantization Map for the Vector Haar Approximation

The quantization technique used for this special case is just the integer part of the difference of the expected values defined before,

$$\mathcal{V}(\Delta_i) = \lfloor \Delta_i + 0.5 \rfloor \tag{5.6.6}$$

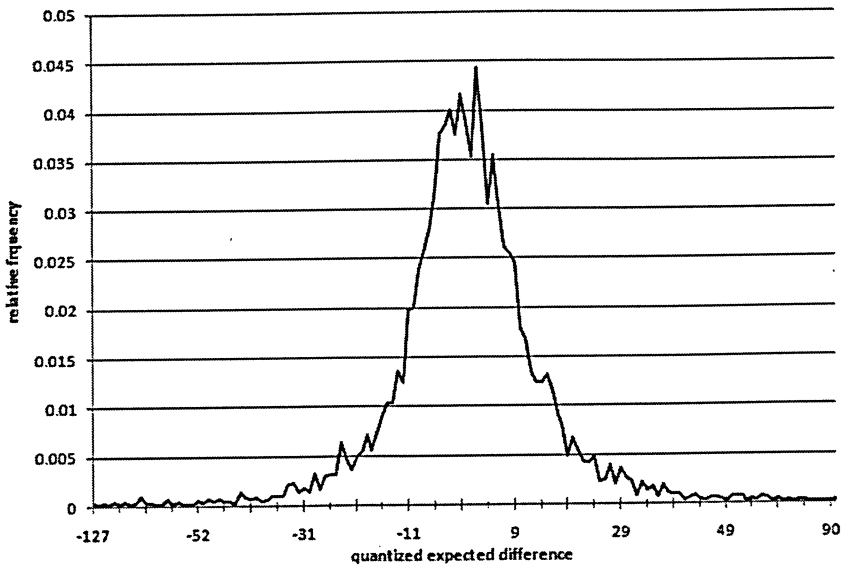


Figure 5.16: Relative frequency of the quantized difference of the expected values

Figure 5.16 shows and example of the relative frequency of the quantized differences Δ_i. The Faces set was used with a PSNR=40. As we have done previously in this Chapter (see Section 5.4.2) the average bits per symbol is given by

$$H_{\mathcal{M}_Q} = -\sum_{Q_k \in \mathcal{J}_Q} p_k \log_2 p_k \quad \text{where} \quad p_k = \frac{|Q_k|}{|Q|}, \tag{5.6.7}$$

where p_k is the relative frequency of each symbol in $\mathcal{J}_Q$. Then the theoretical total cost associated with the quantization map can be computed as follows

$$C_{\mathcal{M}_Q} = H_{\mathcal{M}_Q} \times |Q| \tag{5.6.8}$$

5.6.2 Total Cost for the Vector Haar Approximation

The total cost C_T for this case is calculated as

$$C_T = C_{\mathcal{M}_\Pi} + C_{\mathcal{M}_S} + C_{\mathcal{M}_Q}, \tag{5.6.9}$$

where $C_{\mathcal{M}_\Pi}$ is the cost associated with the partition, $C_{\mathcal{M}_S}$ is the cost associated with the tree, $C_{\mathcal{M}_Q}$ is the cost associated with the quantized coefficients. Examples of this approximation method are provided in Chapter 7.

5.7 Martingales Difference (MD): Scalar Case Approximation

The MD scalar case approximation is similar to the Haar scalar case approximation, but in this case there could exist two inner products in each node, corresponding to $\psi^{(0)}$ and $\psi^{(1)}$, also we need the information of the partition and the information of whether $y_2 > y_1$ (see Section 3.6.3). Figure 5.17 shows the tree structure for this

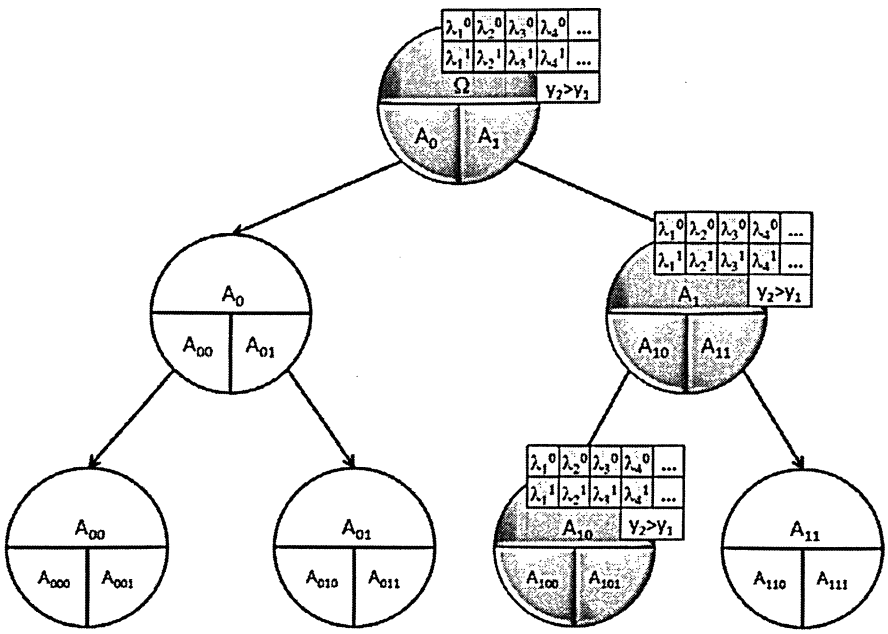


Figure 5.17: MD Tree for the scalar approximation

case, recall that there exist two sets of scalar inner products, $\{\lambda_1^0, \lambda_2^0, \lambda_3^0, \lambda_4^0, \dots\}$ and $\{\lambda_1^1, \lambda_2^1, \lambda_3^1, \lambda_4^1, \dots\}$, the encoding scheme is similar to the one we have seen in section (5.5.1), but in the present case we need extra information to decide whether an inner product belongs to the first set or to the second set.

5.7.1 Indices information

The indexing information for this case can be encoded using three different approaches. The first approach uses d bits to encode whether a inner product is included or not for each set of inner products. The second approach uses an index header for each inner product included, where the first header's bit indicates the corresponding set. And the third approach uses a special null character to identify when a scalar inner product is not included. Figure 5.18 shows the different models for the sets

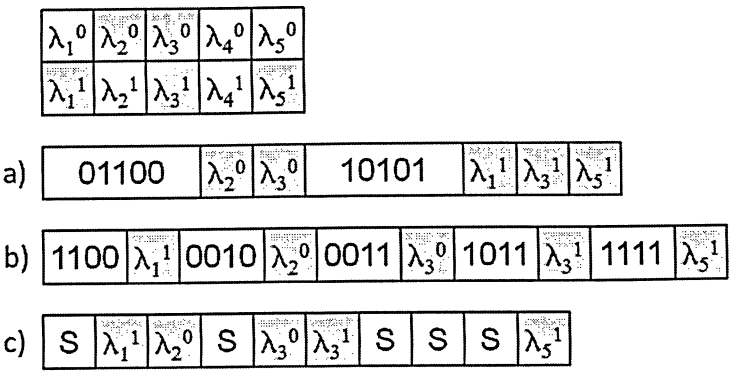


Figure 5.18: a)Binary encode b) Header encode c) Special null character

$\{\lambda_1^0, \lambda_2^0, \lambda_3^0, \lambda_4^0, \lambda_5^0\}$ and $\{\lambda_1^1, \lambda_2^1, \lambda_3^1, \lambda_4^1, \lambda_5^1\}$ where $\{\lambda_2^0, \lambda_3^0\}$ and $\{\lambda_1^1, \lambda_3^1, \lambda_5^1\}$ are needed for the approximation, it is possible to evaluate a priori which model is the best, in terms of the minimum number of bits, to store the inner products in each node.

Then for a given node n the associated cost of each model is calculated as follows

- Case a)
$$C_{I_n} = 2\ d + (k_0 + k_1)\ H_{\mathcal{M}_Q} \tag{5.7.1}$$
- Case b)
$$C_{I_n} = (k_0 + k_1)\ (\log_2 d + 1) + (k_0 + k_1)\ H_{\mathcal{M}_Q} \tag{5.7.2}$$
- Case c) We assume that the special null character has $H_{\mathcal{M}_Q}$ bits
$$C_{I_n} = 2\ d\ H_{\mathcal{M}_Q} \tag{5.7.3}$$

where d is the number of inputs, k_0 and k_1 the number of inner products of each set included in the representation, $H_{\mathcal{M}_Q}$ is the average numbers of bits needed to store the inner products. Then

$$C_I = \sum_n (C_{I_n} + 1), \tag{5.7.4}$$

recall that we add 1 bit to each node, this bit indicates whether $y_2 > y_1$.

5.7.2 Total Cost for the Scalar MD Approximation

The total cost C_T for this case is calculated as

$$C_T = C_{\mathcal{M}_\Pi} + C_{\mathcal{M}_S} + C_I \tag{5.7.5}$$

Where $C_{\mathcal{M}_\Pi}$ is the cost associated with the partition, $C_{\mathcal{M}_S}$ is the cost associated with the tree, and C_I is the indexing cost. The cost associated with the quantized coefficients $C_{\mathcal{M}_Q}$ (see section 5.4), is included in C_I. We will see several examples in Chapter 7, related to the behavior of the algorithm and the theoretical number of bits needed to store the approximation.

5.8 Full Bathtub Approximation

This particular case solved by the full bathtub algorithm also would need an extra function $\psi^{(1)}$ en each node, to be convergent. The information required for this case is the scalar inner products corresponding a each VGS functions, the partition associated and the tree structure. Also we need the value of c in order to calculate the value of b. The cost associated with this particular case is similar to the cost

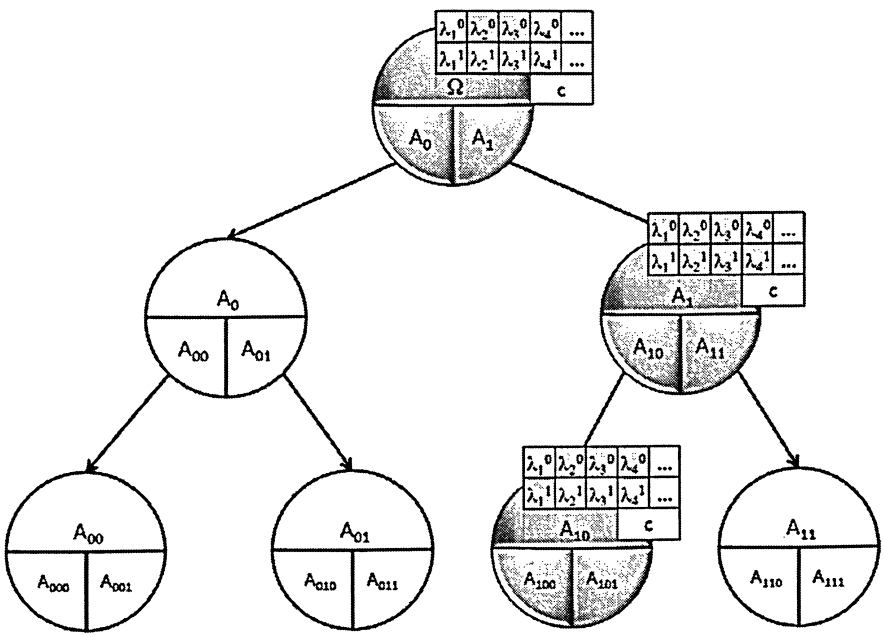


Figure 5.19: Full Bathtub scalar approximation

obtained in the previous Section 5.7.1, the only difference is in equation (5.7.4) that in this case is written as

$$C_I = \sum_n (C_{I_n} + 6), \tag{5.8.1}$$

recall that we add 6 bits to each node, this bits represent the c value that is a real number encoded quantized and encoded.

5.9 Leaves Average Approximation

For this specific approximation, we assume that the VGS algorithm was applied to an input set and at some point a partition is obtained. Then for a given partition $\Pi(\Omega)$ we compute the integer part of the average of each input image over each atom.

$$\lambda_{ij} = \left\lfloor \frac{1}{|A_j|} \sum_{w \in A_j} X[i](w) \right\rfloor \quad \text{and} \quad A_j \in \Pi(\Omega) \tag{5.9.1}$$

where $\lambda_{ij} \in \mathbb{Z}$. Now defining

$$\Lambda = \{\lambda_{ij} \text{ for all } i = 1, \ldots, d \text{ and } j = 1, \ldots, n\} \tag{5.9.2}$$

where $n = |\Pi(\Omega)|$ and d is the number of input images, then $|\Lambda| = n \times d$.

The approximation X_Π is given by

$$X_\Pi(w) = (\lambda_{1j}, \lambda_{2j}, \ldots, \lambda_{dj}) \quad \forall\, w \in A_j \tag{5.9.3}$$

then we define the total cost associated to this approximation as the cost associated to the partition, that is given, plus the cost C_Λ associated to encode Λ. Then

$$C_{Total} = C_{\mathcal{M}_\Pi} + C_\Lambda \tag{5.9.4}$$

Now in order to encode Λ with minimum amount of bits, we apply the following scheme, shown in Figure 5.20, that consist of two different parts:

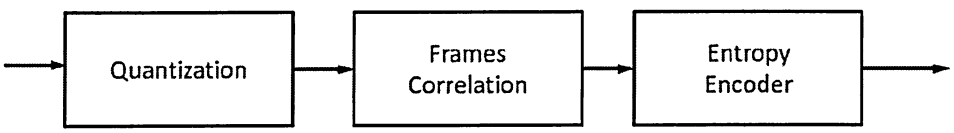


Figure 5.20: Encoding Flowchart

Any of the blocks in Figure 5.20 could included or not,with different results. We will start describing the last block because it is required in order to check the performance of the previous blocks.

5.9.1 Entropy encoding

Let us define Λ_k as the set of all values λ_{ij} equal to k

$$\Lambda_k = \{\lambda_{ij} \in \Lambda : \lambda_{ij} = k\}, \tag{5.9.5}$$

and also define the symbol set $\mathcal{J}_\Lambda$

$$\mathcal{J}_\Lambda = \{\Lambda_k \subset \Lambda : \Lambda_k \neq \emptyset\} \tag{5.9.6}$$

Using entropy encoding we find that

$$H_\Lambda = -\sum_{\Lambda_k \in \mathcal{J}_\Lambda} p_k \log_2 p_k \quad \text{where} \quad p_k = \frac{|\Lambda_k|}{|\Lambda|} \tag{5.9.7}$$

where p_k is the relative frequency of each symbol. Recall that $|\mathcal{S}_\Lambda| \leq n \times d$ where d is the number of input images.

The theoretical cost associated to the set of averages is given by

$$C_\Lambda = H_\Lambda \times n \times d \tag{5.9.8}$$

Using a set of four images formed by {Barbara, Lena, Boats, Peppers} see Chapter 7, of size 128×128 pixels, the VGS algorithm in a Haar mode was run until PSNR = 40, and the partition obtained contains 6983 atoms.

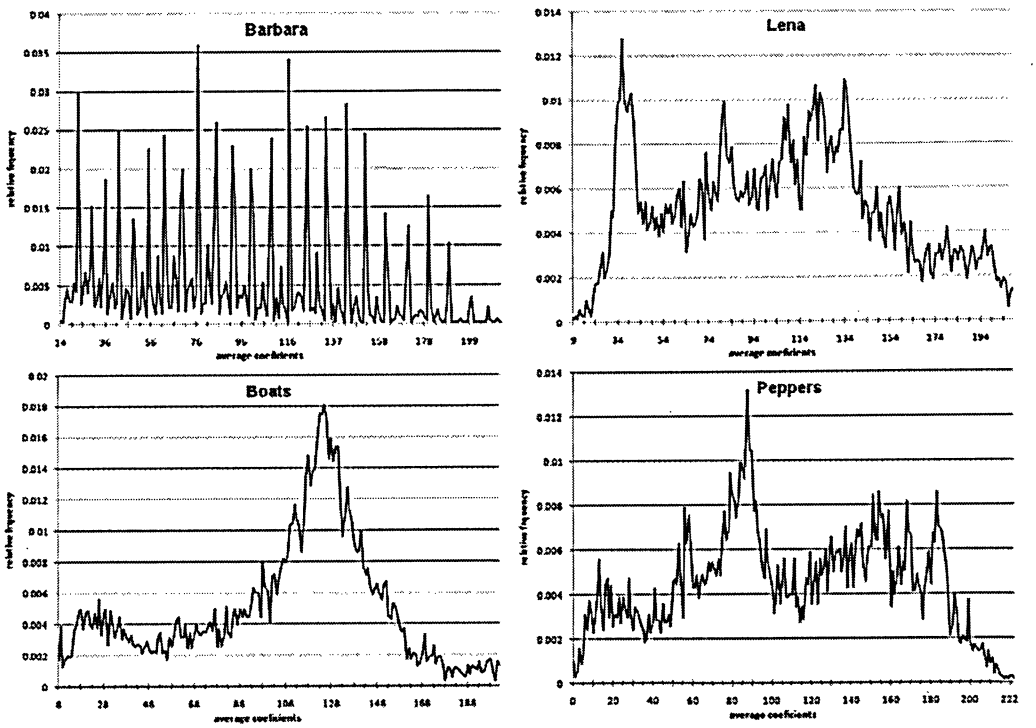


Figure 5.21: Relative frequency of the average coefficients for each input

Figure 5.21 shows the histograms of each input image, it is possible to appreciate that there exists a slight common structure, that is shown in Figure 5.24. Notice that the

range of the images is given by $[0, 255]$ therefore the worst case can be encoded with $\hat{H}_\Lambda = \log_2 256 = 8$ bpp and the maximum cost $\hat{C}_\Lambda = 8 \times 6983 \times 4 = 223456$ bits.

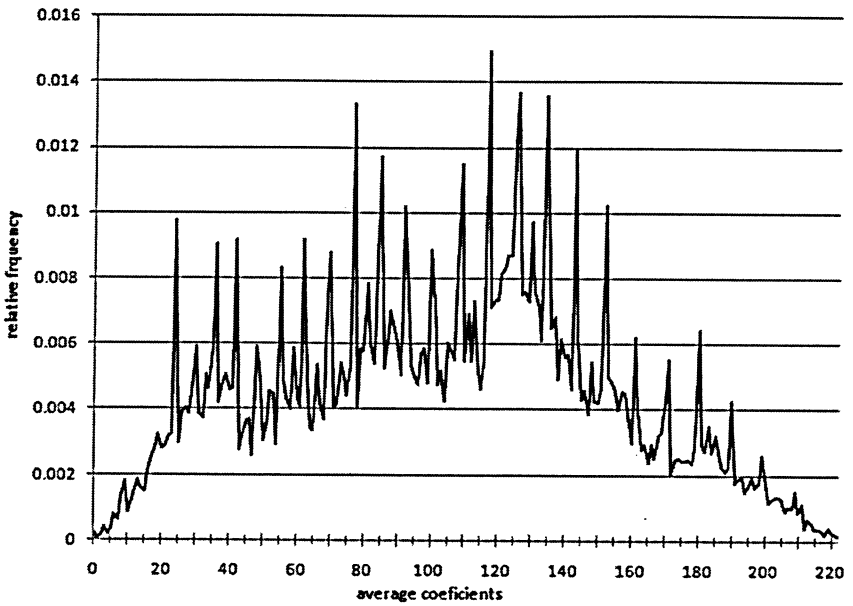


Figure 5.22: Relative frequency of the average coefficients for the input vector

The values obtained for this particular case are where $H_\Lambda = 7.499$ bpp and $C_\Lambda = 209479.22$ bits, approximately a compression of about 6.25%. Again from Figure 5.22 it is possible to observe that the values in the mid-range should be encoded with less bits than the values in the extrema.

5.9.2 Quantization

A quantization over this values means a quantization over the range of values taken by the image, i.e. a reduction of the gray levels of the image. The mean values of the images could be changed. Although the algorithm seem to be not so sensible to this quantization, in terms of the image degradation, the effects, specially on the smooth parts are not desirable.

The idea is to find a quantization function $\mathcal{V} : \mathbb{Z} \rightarrow \mathbb{Z}$ that once applied to the λ_{ij}'s the distortion remains small enough. A quantization can be done using many different approaches, but the best results were obtained using a uniform quantization algorithm described below:

Uniform quantization

The uniform quantization function affects all values alike.

$$\mathcal{V}(\lambda_{ij}) = \left\lfloor \left\lfloor \frac{\lambda_{ij}}{c} \right\rfloor \times c \right\rfloor \quad \text{and} \quad A_j \in \Pi(\Omega) \tag{5.9.9}$$

and $c > 0$.

In order to verify the performance of the uniform quantization, we used the image set previously used, with an original PSNR=40, then by applying the algorithm and the uniform quantization we can see the distortion produced by the quantization. Figure 5.23 shows two charts, the left one shows the relation between the average number of bits needed to store the each average value versus the constant c; and the right chart shows the Total PSNR, obtained once the reconstruction was done, versus the constant c. It is important to check both charts at the same time, because although for $c = 40$, the number of bits is 2.32 per average value, the PSNR=22, and this is not an acceptable result.

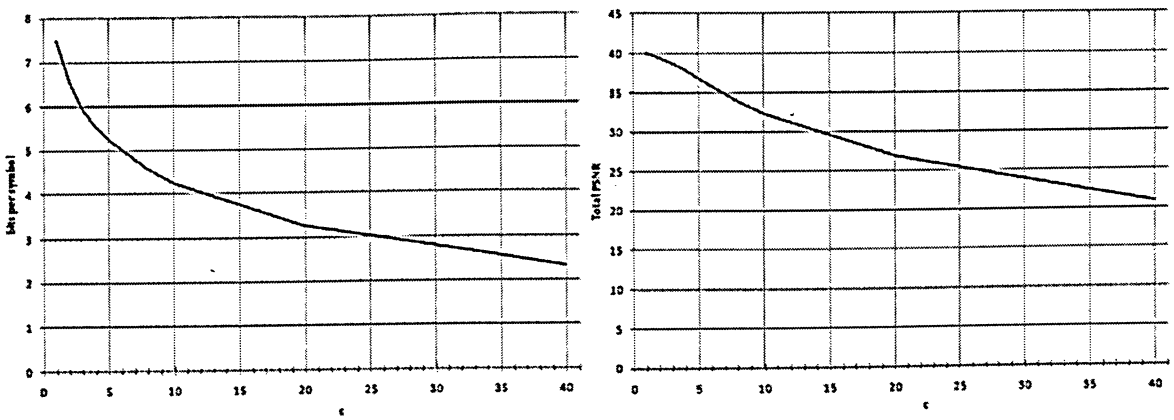


Figure 5.23: Left: number of bits per symbol vs. c. Right: Total PSNR vs. c.

There exists other types of quantization but the results were not as good as the uniform quantization model.

Square root size quantization

Intuitively atoms with less number of pixels could be more affected than pixels that belongs to large atoms. The size of an atom is given by $|A_j|$

$$\mathcal{V}(\lambda_{ij}) = \left\lfloor \left\lfloor \frac{\lambda_{ij}}{\sqrt{\frac{\max\{|A_j|\}}{|A_j|}})} \right\rfloor \times \sqrt{\frac{\max\{|A_j|\}}{|A_j|}} \right\rfloor \quad \text{and} \quad A_j \in \Pi(\Omega) \tag{5.9.10}$$

Logarithmic size quantization

$$\mathcal{V}(\lambda_{ij}) = \left\lfloor \left\lfloor \frac{\lambda_{ij}}{\ln\left(\frac{\max\{|A_j|\}}{|A_j|}\right)+1} \right\rfloor \times \left(\ln\left(\frac{\max\{|A_j|\}}{|A_j|}\right)+1 \right) \right\rfloor \quad \text{and} \quad A_j \in \Pi(\Omega) \tag{5.9.11}$$

5.9.3 Frame correlation

It is very common in video applications to use the fact that consecutive frames are similar, then some algorithms take the difference between those frames, this method reach to a cumulative error, then after a few frames most algorithms restart encoding again. We use this property too, but with different meaning. If two frames are similar the the average value in a given atom should be similar too. Then

$$\lambda_{ij} \sim \lambda_{i+1\ j} \tag{5.9.12}$$

then we define the difference average values as

$$\alpha_{i+1\ j} = \lambda_{ij} - \lambda_{i+1\ j} \quad \text{and} \quad \alpha_{0j} = \lambda_{0j} \tag{5.9.13}$$

Then the α_{ij}'s are now the new input symbols for the entropy encoding block.

The results over the image set, without using quantization, are not good; the reason of this is because those images have nothing in common, considering the set as a video sequence. But using the first 9 images of the video sequence (see Chapter 7) the results are quite impressive. Again we are using a good approximation of the video sequence with a PSNR=40, the partition has 673 atoms. Using equation (5.9.8), we find that the maximum cost associated to store the average information is equal to in this case $H_\Lambda = 8$, $n = 673$ and $d = 9$ then $\hat{C}_\Lambda = 8 \times 673 \times 9 = 48456$.

In order to show the contrast we present first the results without using frame correlation. The theoretical average number of bits to store each average value is 7.76. Figure 5.24 shows the relative frequencies histogram. The cost associated to this is equal to $C_\Lambda = 7.76 \times 673 \times 9 = 47002$ approximately a 3% of compression.

Now using the image correlation information the the average number of bits is 4.93. Figure (5.25) shows the histogram and it becomes obviously that most of the values are zeros. The cost for this case is equal to $C_\Lambda = 4.93 \times 673 \times 9 = 29861$, what is close to the 39%.

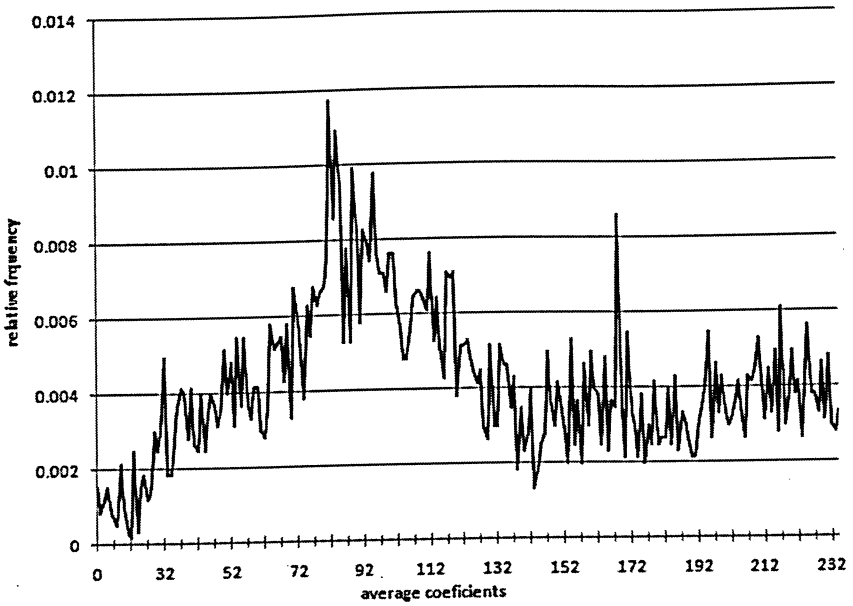


Figure 5.24: Histogram the average coefficients for the video sequence

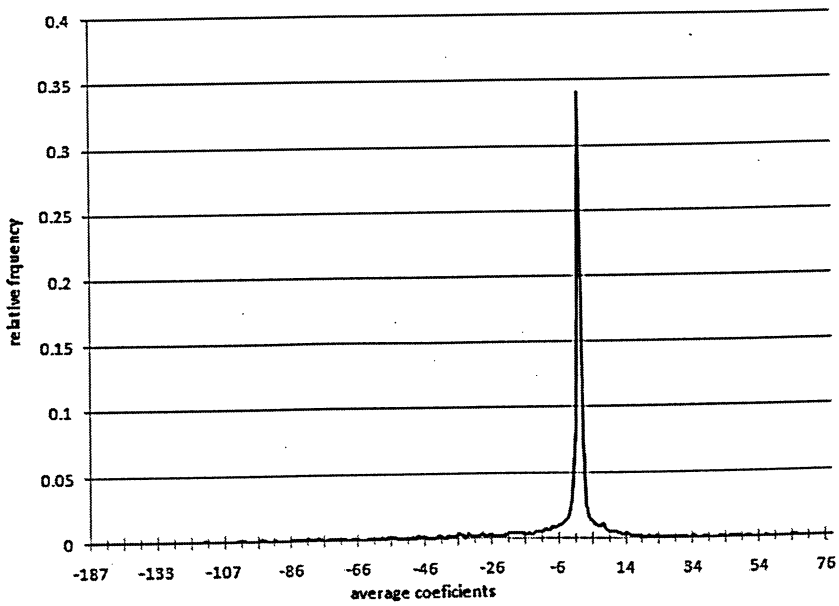


Figure 5.25: Histogram of the difference of the average coefficients for the video sequence

Chapter 6

Related Techniques

The interplay between the three components of any image coder cannot be over-emphasized since a properly designed quantizer and entropy encoder are absolutely necessary along with an optimum signal transformation to get the best possible compression.

From the point of view of the human vision system HVS, many enhancements have been made to the standard quantizers and encoders using wavelets transforms, [5] and [6]. A number of more sophisticated variations of the standard entropy encoders like the arithmetic code have also been developed. Those advances combined have resulted in a better image distortion for a specific bit rate, [23].

A variety of novel and sophisticated wavelet-based image coding schemes have been developed during the last years. These include EZW [22] , SPIHT [20], EBCOT [24], GW Geometric wavelets [1]. This list is no exhaustive and many more techniques have been developed and are being developed. We will briefly discuss only two of these interesting algorithms here because they are specially related to this thesis.

6.1 Embedded Image Coding Using Zerotrees of Wavelet Coefficients (EZW)

One of the most significant advances in the image compression field has been developed by J.M Shapiro [22]. He focused into the main two problems, the first is to obtain the best image quality for a given target rate or distortion, and the second to use and embedded code.

6.1.1 Embedded coding

The idea of an embedded code is such that all encodings of the same image at a lower bite rates are embedded at the beginning of the stream for a target bit rate.

An embedded code contains the most significant coefficients encoded first, then it is possible for the encoder to stop the encoding at a rate or distortion, by monitoring these quantities during the process. Similarly, the decoder based on a continuous measurement of the target rate or distortion, can stop the decoding.

6.1.2 Discrete wavelet transform

Over the last years, the wavelet transform has been successfully applied to many problems in signal processing, and in image compression in particular. Many applications using wavelets outperform other coding schemes like the one based on DCT. Because wavelets basis have variable length there is no need to subdivide the input image in blocks, then the blocking artifacts are avoided at higher compression rates. As the code can be embedded it is also suitable for transmissions environments.

Figure 6.2 shows and example of the subband filter applied to an image.

The wavelet transform used in [22] is based on the hierarchical subband system, where the subbands are logarithmically spaced in frequency using an octave band decomposition. Figure 6.1 shows a two-dimensional four-band filter for image encoding. In a two-scale wavelet decomposition, the image is divided into four subbands using separable filters, Figure 6.3 shows an example of this decomposition. The subbands are called LL: Low row-column subband, LH: Low rows - High columns, HL: High rows - Low columns, HH: High row-columns subbands.

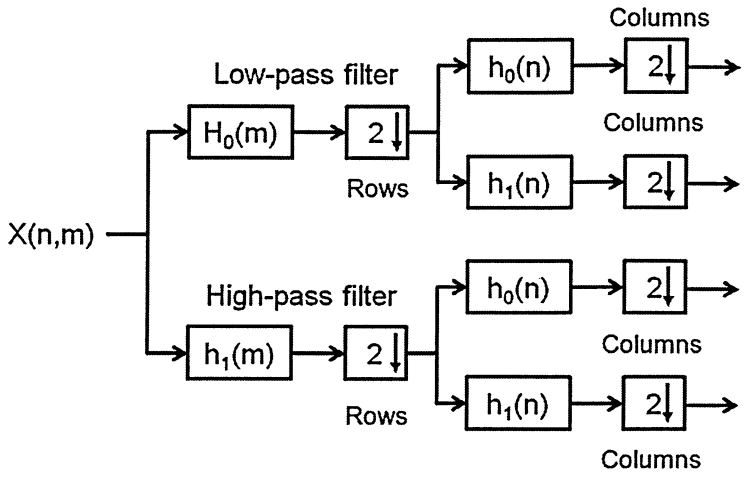


Figure 6.1: Two-dimensional, four-band filter bank

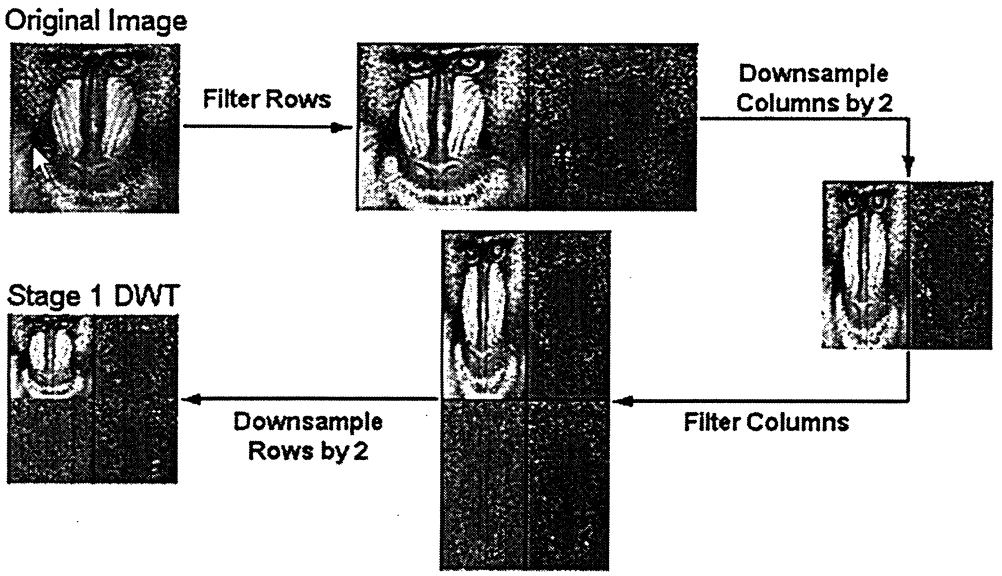


Figure 6.2: Two-dimensional image filter

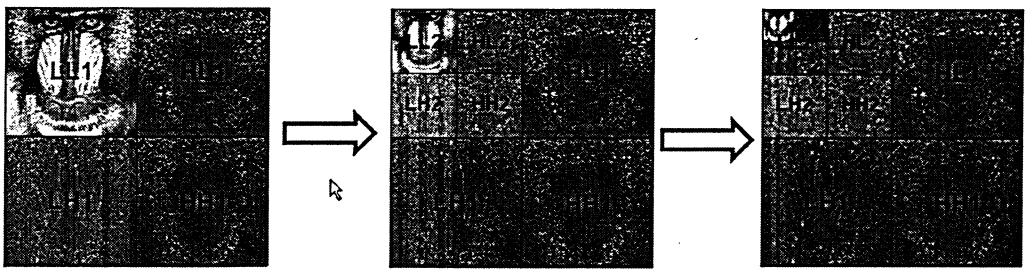


Figure 6.3: Two-scale wavelet decomposition

6.1.3 Zerotrees and wavelets coefficients

In octave-band wavelet decomposition, shown in Figure ??, each coefficient in the high-pass bands of the wavelet transform has four coefficients corresponding to its spatial position in the octave band above in frequency. Because of this very structure of the decomposition there is a tree-like data structure to represent the coefficients of the octave decomposition.

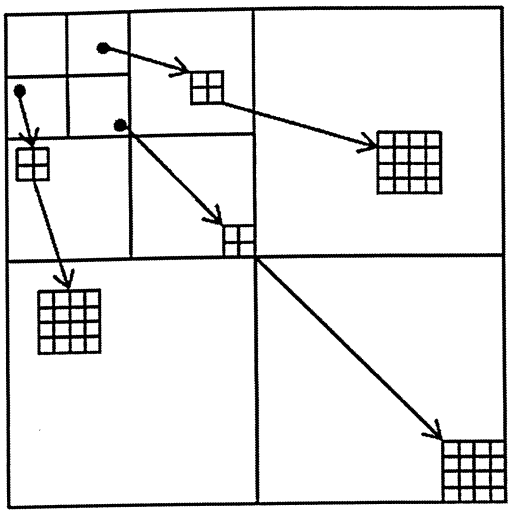


Figure 6.4: Octave-band representation

According to [22] a large fraction of the budget is spent on encoding the significant map, defined as the binary decision of whether a quantized coefficient is zero or not.

The total cost is defined by

$$\text{Total Cost} = \text{Cost of Significant Map} + \text{Cost of nonzero values}, \tag{6.1.1}$$

the cost associated to the significant map represents a large portion of the budget at low bit rates and is likely to increase as the rate decreases.

The zerotree notation introduced by Shapiro is based on the hypothesis that with a high probability, if a node in the coefficient tree is insignificant with respect to a given threshold then all nodes below in the same branch are insignificant too, i.e. if a wavelet coefficient at a coarse scale is insignificant then in the same spatial location at a finer scale with a high probability the coefficients are also insignificant. Therefore as the tree grows as powers of four, many insignificant coefficients are not considered by the encoder.

This representation uses a tree structure where the coarse coefficient are called parents and all coefficients in the same spatial location at a finer scale are called children, also the coefficients in the same branch are called descendants.

The significant map in EZW can be encoded using a 3-symbol alphabet: 1) Zerotree root, 2) Isolated zero and 3) Positive significant. But it has been checked that adding a new symbol in order to encode the sign of the significant coefficients achieved better results. Then instead of using just positive significant symbol, the following two symbols are added: 3) Positive significant and 4) Negative significant.

One of the reasons of why the zerotrees outperforms the DCT is that using EZW many coefficients are predicted at a coarse scale, specially in smooth areas where the DCT, due to the block size restriction, can not compete.

In order to visit the most significant coefficients first the following scheme is proposed: The parents must be scanned before children, also all positions in a given subband are scanned before the scan moves to the next subband, Figure 6.5 shows the scanning scheme.

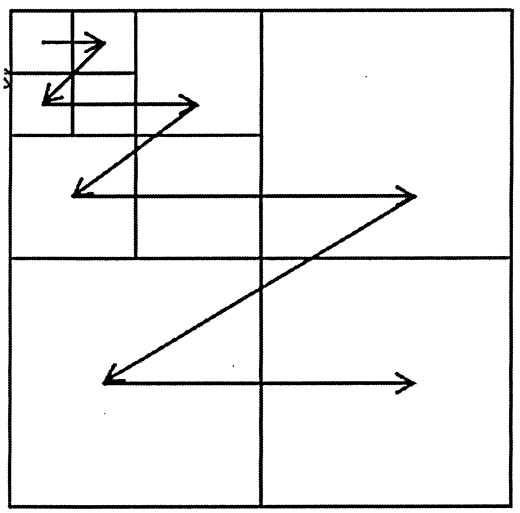


Figure 6.5: Scanning order of the subbands

To perform the embedded coding a successive approximation quantization is applied. Several passes are done over the coefficient identifying the zerotrees using different thresholds. The coefficients are quantized using a binary alphabet with a "1" indicating that the true value falls in the upper half and a "0" symbol indicating the lower half. Then using this procedure the encoder can terminate the sequence at any point. This is one of the most important aspects of the algorithm.

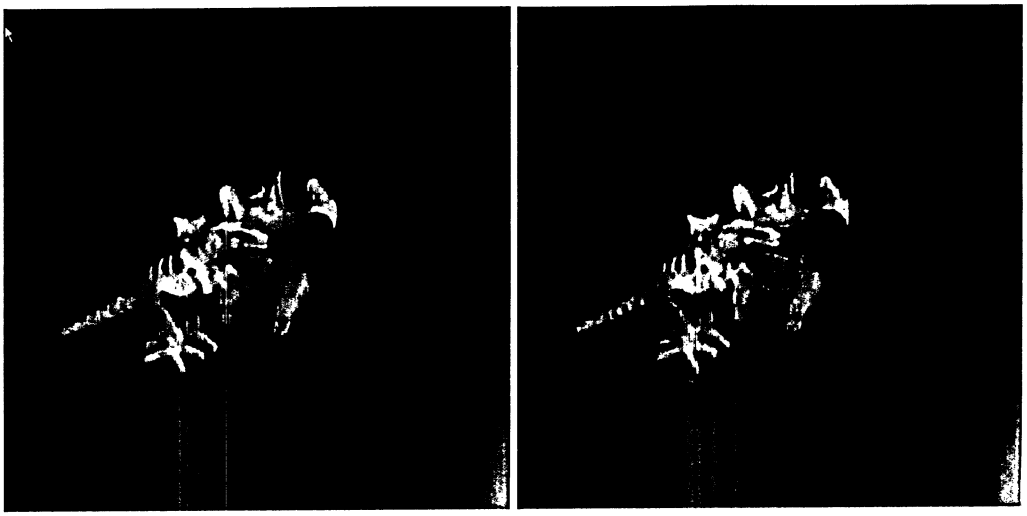
The arithmetic code plays an important role to encode the significant map and allows the entropy coder to incorporate learning into the bit stream itself.

6.1.4 Results

Figure 6.6 shows some impressive results, where Lena (size 512×512 and 8bpp) is encoded using the (a) JPEG2000 with $PSNR = 34.70$db and at the same bit rate the standard (b)JPEG has a $PSNR = 21.89$db. It is possible to appreciate the blocking artifacts in the JPEG. Lena was encoded with the JPEG2000 at the bit rate $0.03125 \equiv 1 : 32$.

Figure 6.6: Detail of Lena, a) JPEG2000 $PSNR = 34.70$db, b) JPEG $PSNR = 21.89$db

Another impressive result is shown in Figure 6.7 (256×256 pixels and 8bpp) where the JPEG2000 (a) with a $PSNR = 38.51$db, outperforms the JPEG (b) with a $PSNR = 35.71$db. Again from the picture it is possible to appreciate the undesirable blocking artifacts.



a) b)

Figure 6.7: Godzilla vs. Robot, a) JPEG2000 $PSNR = 38.51$db, b) JPEG $PSNR = 35.71$db

6.2 Geometric Wavelets (GW)

Geometric wavelets use a BSP (Binary Space Partition) technique and a Geometric Wavelet (GW) approximation, once the sparse representation is found, several techniques are used to encode the data.

This method is applied to 8 bits gray scale images but it could be extended to color images in the same way the JPEG2000 has been applied to different types of images (i.e. 8bits/pixel, 24bits/pixel). Usually a YV12 or YUV12 is used in order to compress a 24bits/pixel color images in a 12 bits/pixel of chrominance and luminance .

6.2.1 Binary Space Partitioning (BSP)

The BSP can be summarized as follows:

- given an image f in $\Omega \subset \mathbb{R}^2$and $\Omega = [0,1]^2$
- the algorithm subdivides Ω into two subsets Ω_0 and Ω_1 using a bisecting line and minimizing a given functional
- the algorithm continues partitioning each region recursively until it reaches a given measure or there is no enough pixels to subdivide.

- the algorithm constructs a binary tree with the partitioning information

To approximate the image f at any region Ω_i they use two bivariate linear polynomials defined by:

$$Q_{\Omega_i} = A_i\ x + B_i\ y + C_i. \tag{6.2.1}$$

The functional used to find the best subdivision for a given region is the following:

$$\mathrm{F}(\Omega_0, \Omega_1) = \arg\min_{\Omega_0, \Omega_1} \|f - Q_{\Omega_0}\|^2_{\Omega_0} + \|f - Q_{\Omega_1}\|^2_{\Omega_1}. \tag{6.2.2}$$

Where Ω_0 and Ω_1 represent the subsets resulting from the subdivision of Ω, with the constraints $\Omega = \Omega_0 \cup \Omega_1$ and $\Omega_0 \cap \Omega_1 = \emptyset$ (this constraints are obviously satisfied in this case). Notice that for simplicity the subindexes related with each step were omitted and this is a general step in the recursive algorithm then Ω_0 and Ω_1 should be consider as children for a given region Ω which is called the father.

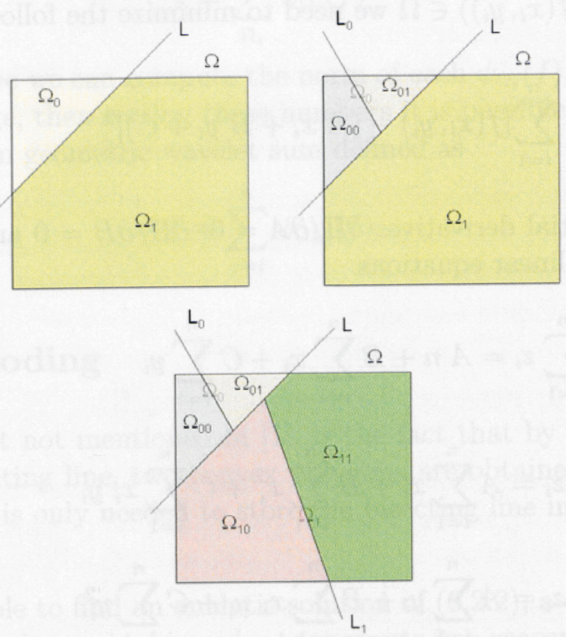


Figure 6.8: Two partition levels using bisecing lines

Figure (6.8) shows the steps of the BSP algorithm, first Ω is subdivided by L in two sub-domains Ω_0 and Ω_1, then Ω_0 is subdivided by L_0 in Ω_{00} and Ω_{01} after that it continues subdividing Ω_1 in Ω_{10} and Ω_{11}, and so on. Then it is possible to represent this in a tree structure as it is shown in Figure (6.9).

Basically the algorithm needs to encode the information of the geometry, in this case the line that cuts each sub-domain, and the approximation function in each sub-domain, that is represented by the polynomial coefficients.

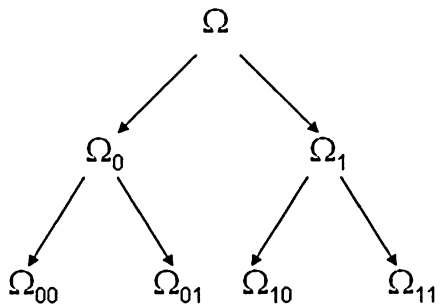


Figure 6.9: BSP tree representation

The polynomial interpolation is obtained using the least square method, computing the difference between the image and the polynomial at a defined region Ω, then given n pixels $(x_i, y_i, z_i = f(x_i, y_i)) \in \Omega$ we need to minimize the following:

$$\Pi = \sum_{i=1}^{n} [f(x_i, y_i) - (A\ x_i + B\ y_i + C)]^2 \tag{6.2.3}$$

and taking the the partial derivatives $\partial\Pi/\partial A = 0$, $\partial\Pi/\partial B = 0$ and $\partial\Pi/\partial C = 0$ we can solve the following linear equations.

$$\sum_{i=1}^{n} z_i = A\ n + B \sum_{i=1}^{n} x_i + C \sum_{i=1}^{n} y_i$$

$$\sum_{i=1}^{n} x_i\ z_i = A \sum_{i=1}^{n} x_i + B \sum_{i=1}^{n} x_i^2 + C \sum_{i=1}^{n} x_i\ y_i$$

$$\sum_{i=1}^{n} y_i\ z_i = A \sum_{i=1}^{n} y_i + B \sum_{i=1}^{n} x_i\ y_i + C \sum_{i=1}^{n} y_i^2$$

Both [1] and [19] used the same approach to approximate the function in each region, but in [19] the procedure is applicable to the whole image, then the cost to apply the algorithm is very high.

6.2.2 Geometric Wavelets (GW)

In [1] they use the local difference to define the ***geometric wavelets***. The local difference computes the difference between the actual partition and the previous one giving us an idea of the degree of change, if the difference is large then the new

partition is capturing new details, and if the difference is small, the new partition does not add new information. The GW is defined as follows:

$$\psi_{\Omega_0}(f) \triangleq \mathbb{1}_{\Omega_0}(Q_{\Omega_0} - Q_{\Omega}) \tag{6.2.4}$$

Where $\mathbb{1}_{\Omega_0}$ is the characteristic function that gives 1 in Ω_0 and 0 in the rest. Ω_0 here means one of the children.

They show that it is possible to reconstruct the function f using GW due to the term cancelations.

$$f = \sum_{\Omega_i} \psi_{\Omega_i}(f) \tag{6.2.5}$$

But using the BSP tree we can compute the norm of each $\psi_{\Omega_i}(f)$, which is a measure of the degree of change, then sorting these numbers it is possible to approximate the function by the n-term geometric wavelet sum defined as

$$f \approx \sum_{j=i}^{n} \psi_{\Omega_{k_j}}(f) \tag{6.2.6}$$

6.2.3 GW Encoding

An important concept not mentioned in [1], is the fact that by cutting each convex polygon using a bisecting line, two convex polygons are obtained. An this is one of the reasons of why it is only needed to store the bisecting line information.

Since is not possible to find an analytic solution of (6.2.2), a quantization schema over the bisecting lines is required in order to compute in a reasonable time each step. Both in [1] and [19] they use the normal line representation instead of the standard Cartesian representation (slope, $y = mx + b$) as it is shown in Figure (6.10).

$$\rho = x\ \cos(\theta) + y\ \sin(\theta) \tag{6.2.7}$$

Using this representation is supposed to reduce the number of trials to find the minimum using a brut force algorithm over the quantized (ρ, θ) space.

The idea is to add each step a new set of pixels, they state in Section (2.1) in [1], that this is possible using determined $\Delta\rho$ and $\Delta\theta$.

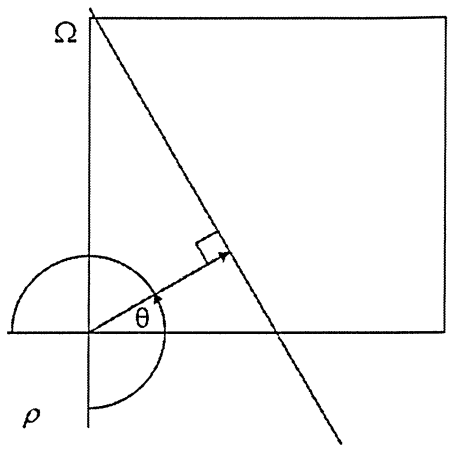


Figure 6.10: Bisecting line

The range of this parameters defined in [1] is $\theta \in [-\pi/2, \pi]$ and $\rho > 0$ while in [19] they allow $\rho < 0$ to be negative, the first approach is better due to they do not need to encode negative values, it means that it is a save of one bit.

Due to the quantized schema θ and ρ can be determined by the index of θ, what it means for a given θ_i we can find fixed numbers j_{max} and j_{min} of ρ_{ij} and therefore it is possible to give a general index k to encode θ_i and ρ_{ij}.

6.2.4 GW sparse representation and encoding

Once the BSP tree is generated the next step is to use (6.2.6) to prune the tree and discard irrelevant information by computing $\|\psi_{\Omega_{k_i}}(f)\|$.

Due to the fact that the leaves are necessary for the reconstruction, in [1] they impose the requirement that if a child appears in the tree then father has to appear too. But it is not necessary that both children appear in the tree, if one is no significant enough, could be excluded from the sparse representation, then all its descendant should be excluded too. This is an improvement with respect to [19] due to there if a partition is done both children appears in the tree independent of whether one child is significant or not.

Actually they do not use the this approach to prune the tree, they apply a rate-distortion (R-D) [?] optimization which is supposed to increases de peak signal to noise ratio (PSNR) by 0.1 dB in some cases.

After the tree is pruned it is encoded using the following information:

- Tree-structure information
  - Number of children
  - Information to distinguish each child node.
- The quantized coefficients Q_Ω
- The bisecting line information of each Ω if it has a child.
- Header information

The tree-structure is encoded using the fact that with a high probability a significant node does not have a significant child, in a similar way like 'zero-trees' [?] . Therefore using Huffman code [10] to encode the three different values (Zero-children='1', One-Child='01', Two-Children='00') it is possible to save in some cases 1 bit, due to normally it is necessary 2 bit to encode 3 different states.

The quantized coefficient Q_Ω that represent the wavelet polynomials are determined by three real numbers (A_i, B_i, C_i) that can be stored with 12 bytes using the standard 4-bytes float representation. But in [1] they show an algorithm to store at a rate of 1.5 bytes per polynomial on average, using the standard Graham-Schmidt method to obtain the orthonormal base representation. This could be the greatest improvement with respect to [19].

The bisecting line is encoded using the fact that $\Delta\theta$ and $\Delta\rho$ are determined by the size of the region Ω, then it is only necessary to store an integer index k to reconstruct the values of θ_i and ρ_{ij}. The decoder has enough information to restore these values.

In order to deal with the time consuming algorithm in [1] they tile the image in squares of 128×128 and they apply the BSP algorithm on each tile. The main disadvantage of doing this is that blocking artifacts appears at the tiles' boundaries. Another disadvantage is that connected areas could be disconnected missing the possibility to improve the approximation.

Chapter 7

Results

7.1 Results Illustrating Properties of the Algorithm

We present a collection of test cases to illustrate, test and compare the VGS algorithm to some current techniques. The results also indicate advantages and some shortcomings of the VGS (at least in its present form).

The reader will note that only scalar approximations are discussed, i.e. no results on vector approximations are presented. The reason for this is that results are quite similar in some cases and it will take us too long to describe.

When reporting distortion values (based on PSNR) for a sequence of images (equivalently: a video sequence, or "vector" input set) we make use of a global measure of error. More precisely, we use (B.0.7) from Appendix B.

7.1.1 The Algorithm Step by Step

In order to illustrate the algorithm we introduce an example where geometric figures were used to show how the algorithm converges. The simple four(so $d = 4$) geometrical images take a reduced number of gray levels. Figure 7.1 shows the original 256 gray-levels images of 128×128 pixels. In this test we will not perform compression and the approximations are steps of the algorithm constructing the full tree.

Figure 7.2 shows the VGS approximation using the scalar Haar case approximation at different stages of convergence. The first approximation (top left) shows the approximation with only one component and clearly the algorithm splits the domain in two atoms in each image, the number of gray-levels is two for this first step under this particular case. The second approximation (top right) shows the approximation

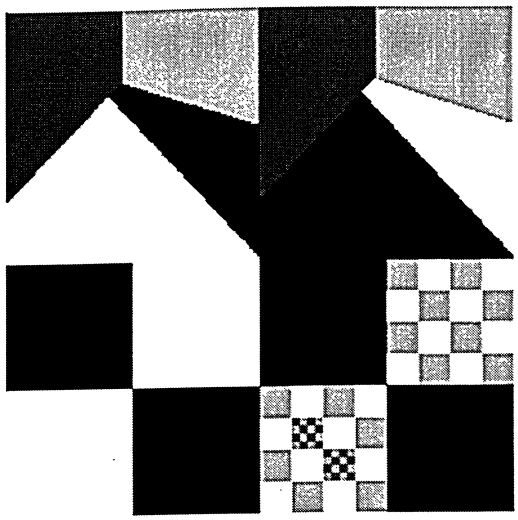


Figure 7.1: Test Set 6

with two components, it is possible to see that the top images are approximated using a partition associated with the bottom images. The third approximation (bottom left) shows that the bottom left image is completely approximated and the top images have almost converged. The last approximation (bottom right) is the approximation using 7 components and it is almost a perfect reconstruction but the algorithm needs one more iteration to complete the bottom right image.

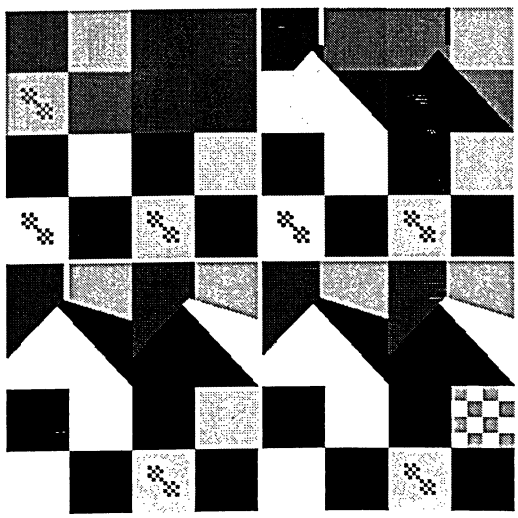


Figure 7.2: Scalar Haar Approximations using, Top left: one component, Top right: two components, Bottom left: three components, Bottom right: 7 components

Figure 7.3 shows the VGS approximation using the scalar Martingale Difference

case at different stages of convergence. The first approximation (top left) shows the approximation with only one component, the difference between this method and the previous one, is that the approximating functions ψ in this case can take 3 values in each node. It is clear that in this case the bottom images were selected to approximates the whole image, and in the second and third approximations it is possible to appreciate how the algorithm tries to approximate the top images based on the bottom images. The last approximation shows almost a complete reconstruction with only 7 components, each taking three different values at each node.

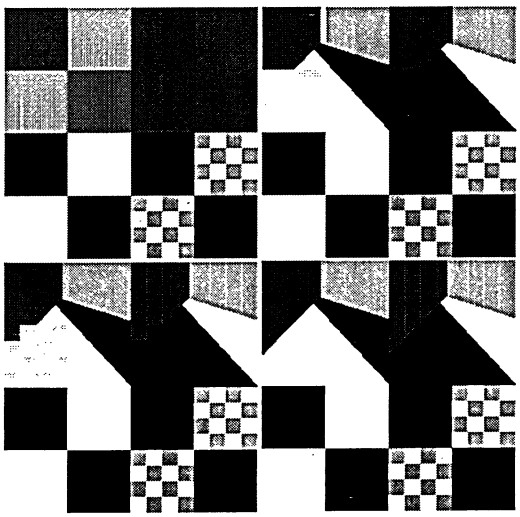


Figure 7.3: Martingale Difference Approximations using, Top left: one component, Top right: two components, Bottom left: three components, Bottom right: 7 components

7.1.2 The Haar Approximation

A set of images was specifically created with the objective of verifying the behavior of the algorithm under the situation where different images are composed basically of trends represented by areas of high statistical spatial correlation associated with low frequencies bands. Faces are situated in the middle of the images but almost use the entire area. Although images seem to be very similar, the smooth variations of them make the vector take many values. Figure 7.4 shows the original set composed of 256 gray-levels images of size 128×128 pixels.

The VGS approximation using the scalar Haar case with a PSNR=22 is shown in Figure 7.5, the number of bits used to encode the 9 images is 90237, but recall that the total number of bits is equal to $16384 \times 9 \times 8 = 1179648$ then the compression is 92.3% with approximately 0.611 bpp. It is possible to appreciate how the algorithm

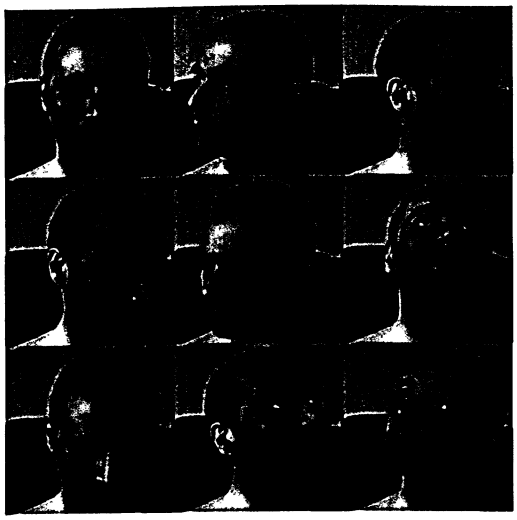


Figure 7.4: Test Set 2

manages to capture the information of all images simultaneously.

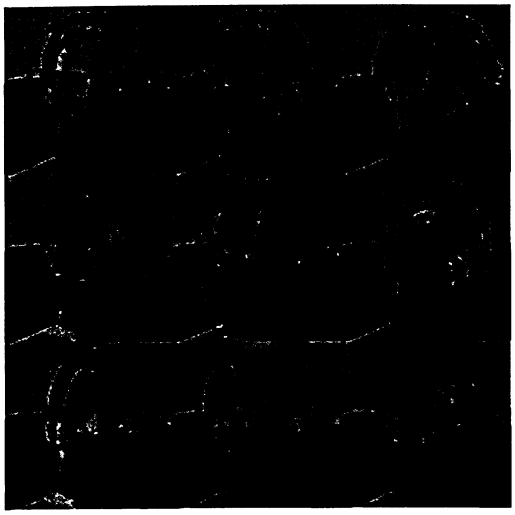


Figure 7.5: Scalar Haar Approximation, 0.611bpp and PSNR=22

Figure 7.6 shows the approximation with PSNR=28 and 1.008bpp, at this point some details become more clear, specially the high frequencies that seems to be well captured by this algorithm.

In Figure 7.7 a detail of the middle image using different approximations is shown, it is clear that the algorithm performs better with edges rather than smooth areas. These last regions are captured using almost the full tree while edges are captured in the first iterations of the algorithm.

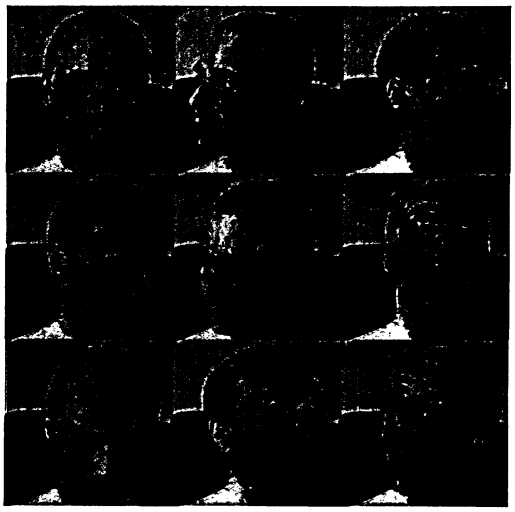


Figure 7.6: Scalar Haar Approximation, 1.008 bpp and PSNR=28

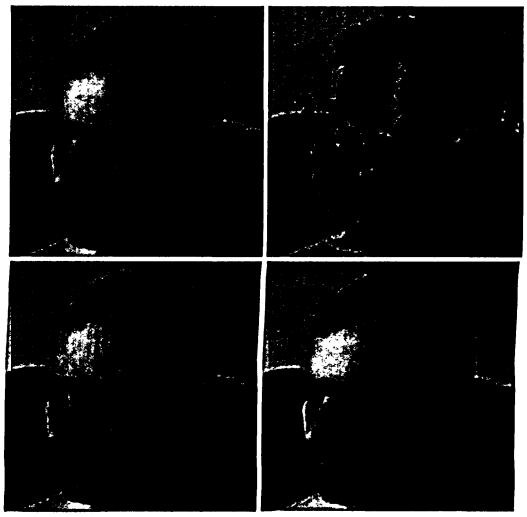


Figure 7.7: Detail of the middle image, Top left: Original, top right: PSNR=22 and 0.611bpp, bottom left: PSNR=34 and 1.617bpp, bottom right: PSNR=40 and 2.633bpp

7.1.3 Counting Bits

Focusing in the compression application one interesting test is to compare the cost of the different maps using a variable number of images, then we will show how the cost of storing each map is affected when the number of images increases from one to nine (for the Test Set 2). Applying the VGS-Haar case algorithm using the faces set, previously used, as the input set, Figure 7.8 shows the comparison between the cost associated with different maps per image vs. the number of images for many distortion values: PSNR=30db, PSNR=40db and PSNR=50db.

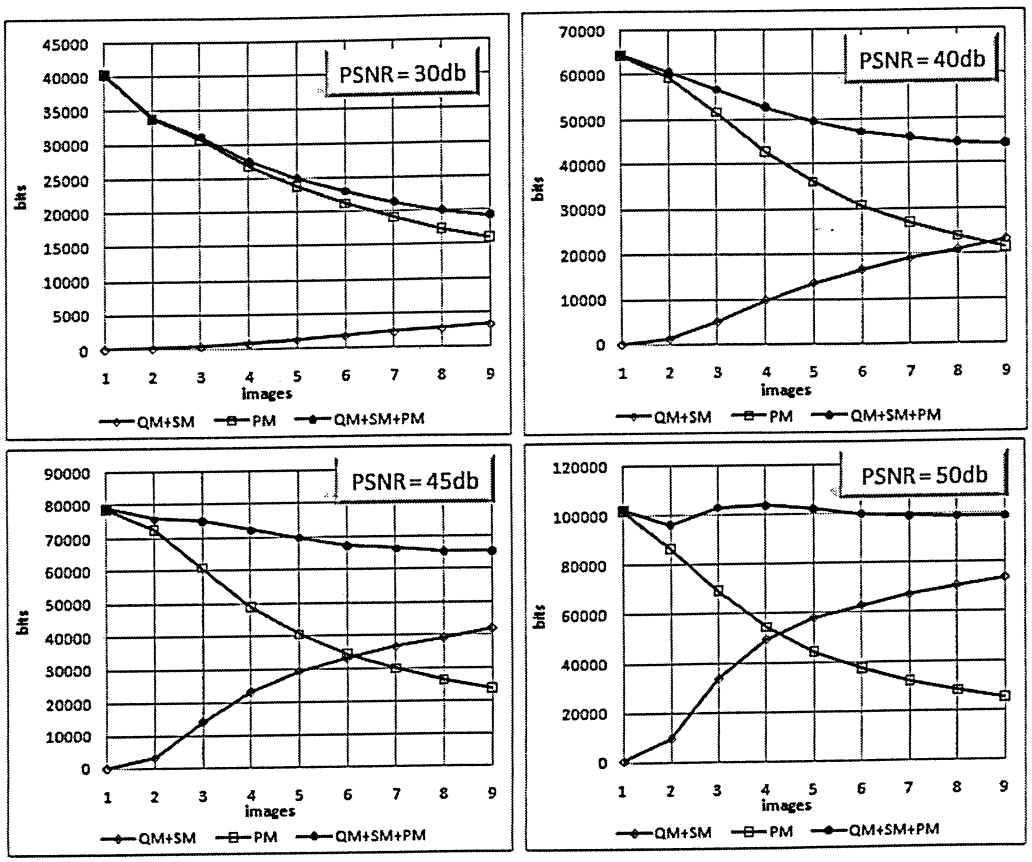


Figure 7.8: Mapping cost per image vs. number of images, Top left: PSNR=30db , top right: PSNR=40db, bottom left: PSNR=45db and bottom right: PSNR=50db

Where "QM", "SM" and "PM" are shorthand notations to denote the costs associated with the quantization map $C_{\mathcal{M}_Q}$, the cost of the significant map $C_{\mathcal{M}_S}$ and the cost of the partition map $C_{\mathcal{M}_\Pi}$ respectively. We would like to emphasize the fact that the cost shown is the average cost per image and is calculated as the specific total cost divided by the number of images (this point is introduced and discussed in Chapter

2. Note that the comparison is done using the partition map $\mathcal{M}_\Pi$ and the sum of the significant map and the quantization map $\mathcal{M}_S + \mathcal{M}_Q$. The reaso for combining tehse two costs into a single number is that the significant map is approximately 10% of the quantization map and it is not large enough to be compared with the partition map in the same graph.

The top left chart in Figure 7.8 shows that for a low PSNR, the cost consists mostly of the partition map and although it should be decreasing asymptotically to zero as the number of images increases to infinity, the number of images selected in this set is not large enough to appreciate this property.

Remark 22. *The maximum cost associated to the partition map $\mathcal{M}_\Pi$ is fixed and equal to the cost of storing two images.*

The top right chart in Figure 7.8 begins to show that at some point the quantization map and the significant map require at least the same number of bits as the partition map using 9 images. Also it is possible to see that there is an optimum value for the total cost per image for a given distortion.

The bottom left chart with a $PSNR = 45$, clearly shows that there exist a number of images for what the number of bits is minimum for this given distortion, unfortunately it seems to be bigger that 9 images.

In the bottom right chart, it is possible to observe that practically there is no minimum value for the total cost per image. This pattern follows from the fact that the significant and quantization maps' costs are increasing very fast in terms of the number of images. In turn, this happens as the required PSRN value is relative high and requires many components to be included in order to achieve such high quality reconstruction.

We have assumed that the significant map is small enough and then it is possible to add it to the quantization map. Figure 7.9 shows that the significant map is approximately the 10% (average) of the quantization map, and this ratio decreases when the number of images increases.

Considering that a video sequence could be an optimal situation for our algorithm, we propose the following test set shown in Figure 7.10.

This test set was down-sampled from 640 × 480 pixels to 128 × 128 using bi-cubic interpolation. Now we will show the behavior of the VGS-Haar case algorithm for the video sequence using a variable number of images.

Figure 7.11 shows the comparative between the different costs per image for a fixed $PSNR = 30db$. Where, as we have done before, the "TOTAL" is equal to $C_{\mathcal{M}_Q} + C_{\mathcal{M}_S} + C_{\mathcal{M}_\Pi}$. Recall that we are using the average cost per image, instead of using the total cost. This not only allow us to determine the optimal number of images for

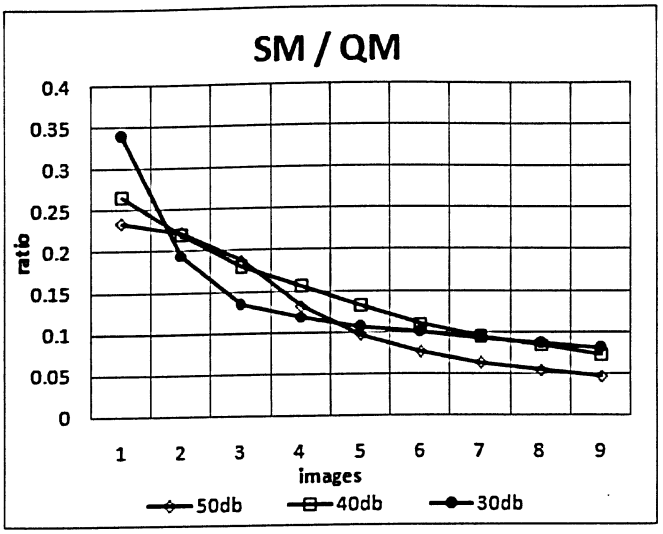


Figure 7.9: Significant map/quantization map ratio, ratio vs. number of images

which VGS will deliver best compression performance but also provides a reasonable bit- scale to compare with other methods (the is discussed in Chapter 2.)

It is possible to see that there is a minimum for 16 images but for this low quality the cost of the quantization map increases so slowly that the total cost is practically a constant value. Now increasing the PSNR we can appreciate in detail the minimum number of images needed to achieve the minimum number of bits.

Figure 7.12 shows that for different distortion values of PSNR, we can find a minimum cost using different numbers of images. The top left chart of Figure 7.12 shows that this minimum is located at 12 images and for the rest of the charts the minimum is found at 9 images. Also in the bottom left and the bottom right charts (45db and 50db), it is possible to see that the cost associated to partition map $C_{\mathcal{M}_\Pi}$ decreases in both at the same rate, but the cost associated to the quantization map $C_{\mathcal{M}_Q}$ rises almost at twice for 50db.

There is another important aspect we can infer from these charts, if we observe in detail the cost associated to the quantization map plus the cost associated to the significance map $C_{\mathcal{M}_Q} + C_{\mathcal{M}_S}$ we can appreciate that there are two local minima one at 9 and the other at 12. The reason of this can be explained because in a video sequence the number of frames per second (fps) is selected to ensure that most of the common life activities are captured in the video. Sometimes the action is slower than the number of frames per second and the result is, practically, a sequence of two or more equal frames.

Adding a new image identical to a previous one in the original set, almost does not increase the encoding cost, just a few bits in the indexing map. Therefore, as a result

Figure 7.10: Video sequence: length 1 second, frame size: 128×128, color depth: 8bpp, 25 fps (frames per second)

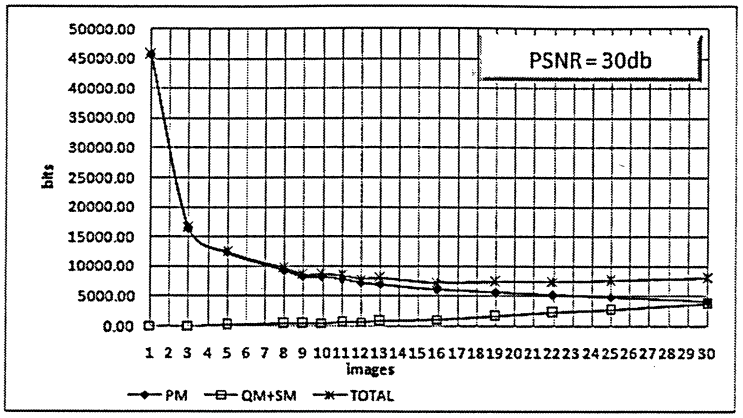


Figure 7.11: Average mapping cost per image comparative for a PSNR=30db

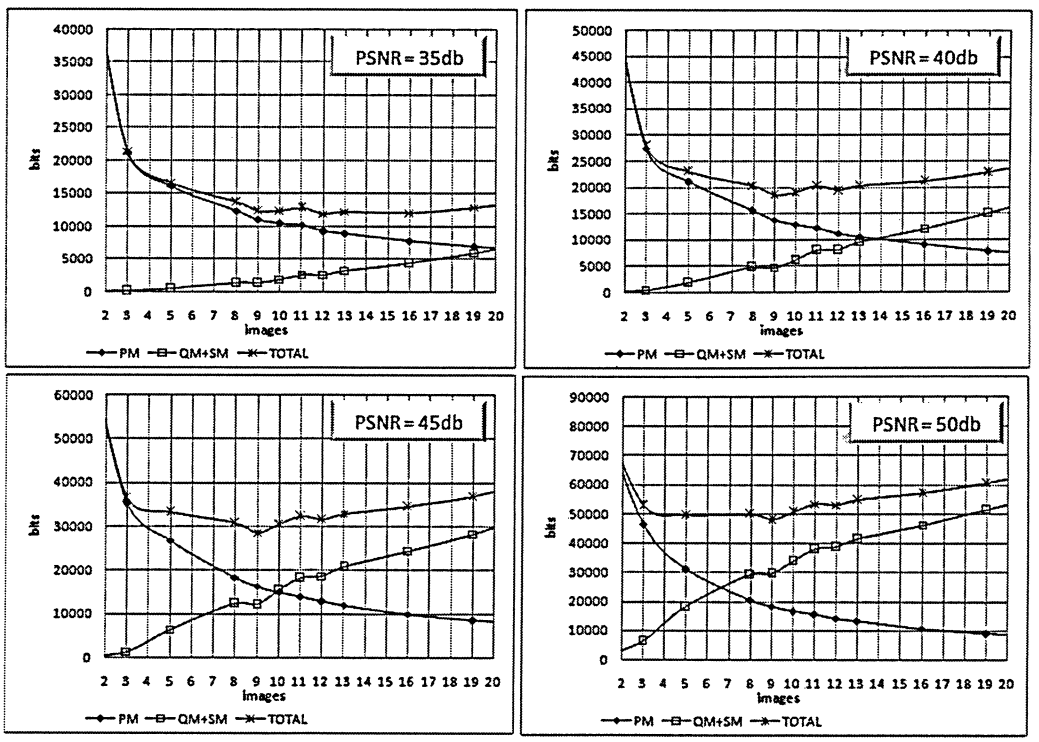


Figure 7.12: Average mapping cost per image comparatives for PSNR=35db, 40db, 45db and 50db

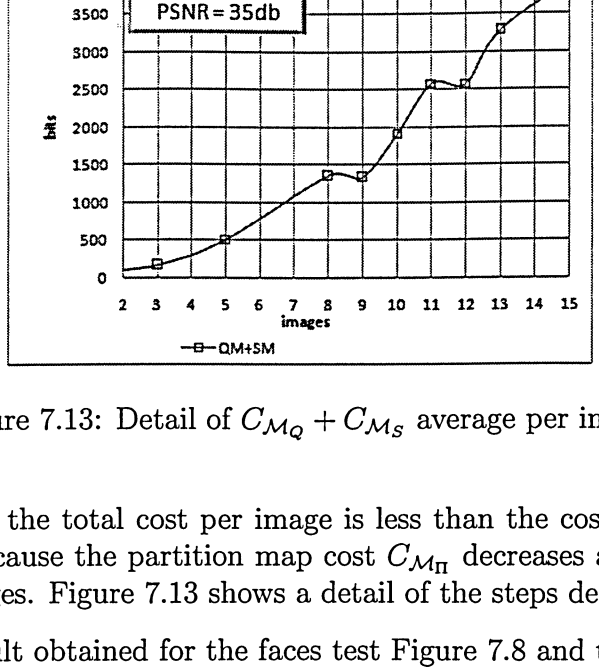


Figure 7.13: Detail of $C_{\mathcal{M}_Q} + C_{\mathcal{M}_S}$ average per image

of this phenomena, the total cost per image is less than the cost without including this new image, because the partition map cost $C_{\mathcal{M}_\Pi}$ decreases asymptotically with the number of images. Figure 7.13 shows a detail of the steps described above.

Comparing the result obtained for the faces test Figure 7.8 and the results obtained with the video sequence Figure 7.12, it is clear that the algorithm needs half of the bits or less (it depends on the distortion) to encode the video sequence compared with the faces.

7.1.4 Selecting the Best Case

In order to compare the performance of the different cases (Haar, Full Bathtub and Martingales Difference) proposed in this thesis, we have used four different sets: the first contains four completely different images from real life, the second is formed by four images designed to take all possible different vector values, the third is composed of 9 images from a video sequence Figure 7.10, and the last is a set of 16 images designed to simulate a video sequence.

The first two sets expose the algorithm to a difficult situation where practically there is no common information among the images.

The first set is shown in Figure 7.14 is composed of "Lena", "Barbara", "Boats" and "Peppers".

Figure 7.15 shows the second test called "the worst case" because it was designed to be the worst case for our algorithm because the combination of the images takes all possible values, therefore the partition associated to the full tree takes 128×128 different values.

Figure 7.14: 4 Images: size 128 × 128 each one, color depth: 8bpp

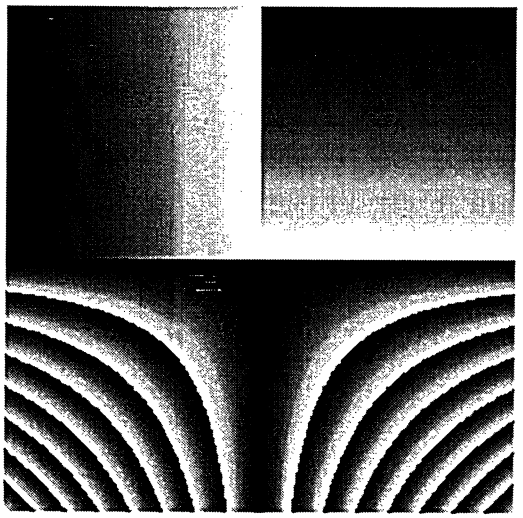


Figure 7.15: Worst case: size 128 × 128 each one, color depth: 8bpp

The last set "Godzilla vs. Robot" shown in Figure 7.16, was designed to simulate a video sequence where the frames are different.

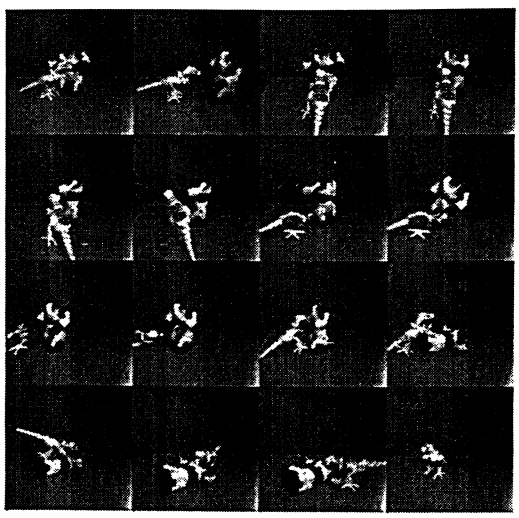


Figure 7.16: Godzilla vs. Robot: size 128×128 each one, color depth: 8bpp

We will compare four different VGS cases, the Haar scalar approximation case ($HAAR$), the Martingales Difference scalar approximation case (MD), the Full Bathtub scalar approximation case (FB) and the Leaves Average case (AVG). Notice that the comparison will be done using the Total number of bits, it means that is the total cost, not the the cost per image. The results are discussed next.

Figure 7.17 shows the comparisons on the first set, the best method on this set is the MD algorithm, the HAAR case is performing well too, the FB is not as good as the others, but definitely the AVG case is the worst case, specially for low distortion levels.

The results on the second set are shown in Figure 7.17, this case the difference between the MD and the HAAR case is much more notorious, and the same as in Figure 7.17, the worst case is the Haar case, specially for high quality approximations.

Figure 7.17 shows the comparison chart for the video sequence set, in this case as before the FB and the HAAR case have a similar behavior, but the AVG approximation case outperforms the others by $\approx 20\%$. Also the FB case seems to be the worst case for this set.

The results for the last set are shown in the chart in Figure 7.20. Here the advantage of the AVG approximation case is not obvious, it seems to be the best method for high quality approximations and to perform badly for low quality approximations.

There is a possible explanation for the behavior of the AVG case, the AVG algorithm

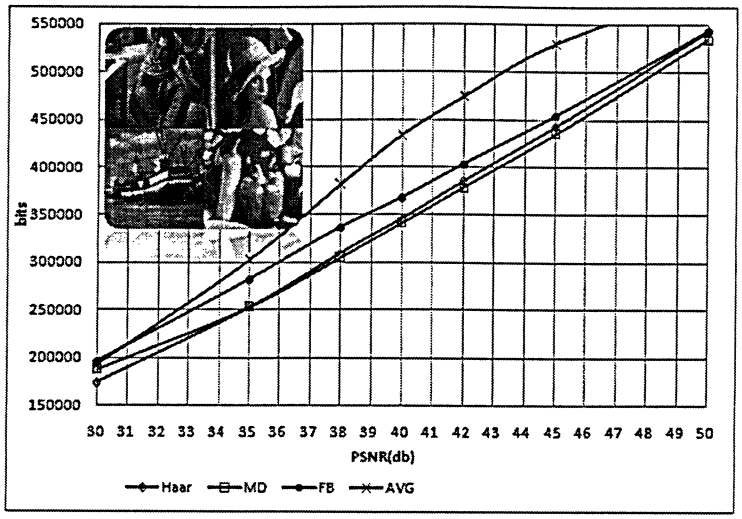


Figure 7.17: Total bit costs for the 4 images set vs distortion

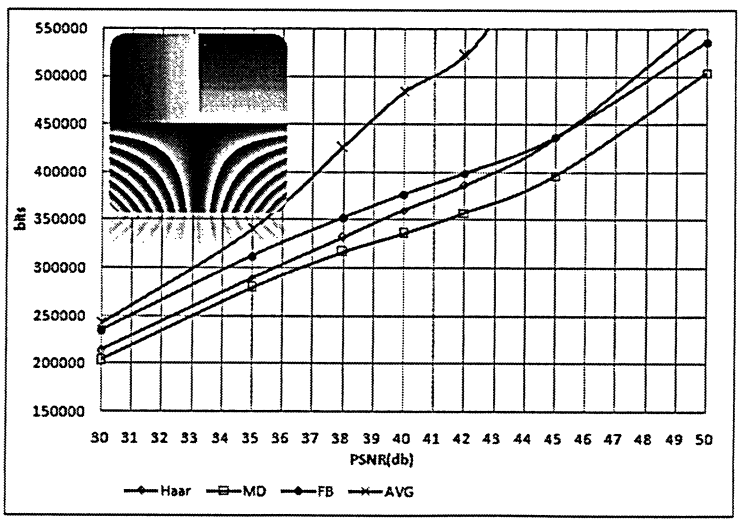


Figure 7.18: Total bit costs for the worst set vs distortion

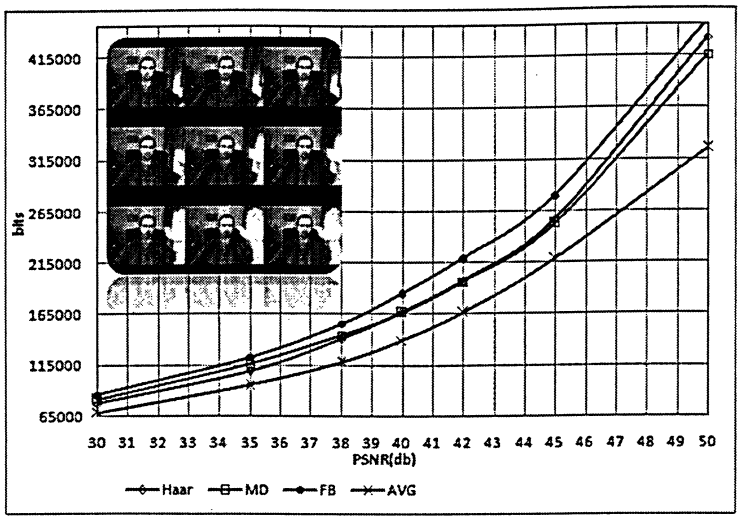


Figure 7.19: Total bit costs for the video sequence set vs distortion

(see Section 5.9) is built on a given partition and the encoding is done taking the difference between the values of two consecutive images in the same atom (see Section ??) due to the images' similarity. In the 4 images set and the worst set there is no similarity at all, furthermore they were specially created with this objective in mind. Therefore it is understandable that the AVG algorithm can not perform well under those conditions. The video is the ideal case where there is a high correlation between frames, and although the last set is not a video there exist a relative correlation between frames, specially the background that remains practically constant.

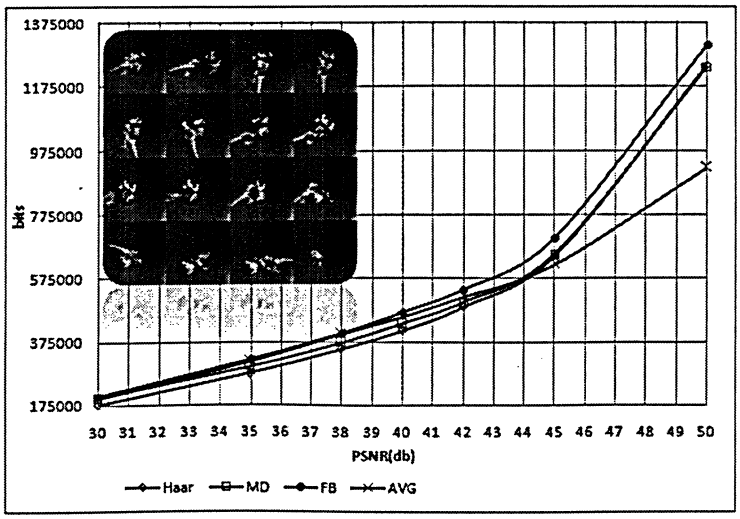


Figure 7.20: Total bit costs for Godzilla vs. Robot set vs distortion

A possible explanation of why the MD case performs better, in almost all cases, than the FB and the HAAR case is the following: MD uses three values to approximate the input vector in a node one for each child, the third value is fixed and it seems to be close to the original value that is the average in this child, therefore the scalar inner product due to the second VGS function $\psi^{(1)}$, that is added to the node in order for the algorithm to be convergent, is smaller than the inner product due to $\psi^{(0)}$. Then the closer this third value is to the average value in the child the smaller the second inner product is stored, and then it could be eliminated by the quantization. This third value seems to be far from the average value in the FB case, then the second inner product is not smaller than the MD case and needs more bits to encode them.

A similar explanation could be done with respect to the HAAR case, considering the case MD takes three values and the third value is close to the average value in the child, then the MD could approximate three values (not always) with only one inner product, while the HAAR needs at least two inner products to approximate three different values.

7.1.5 The Outer Supremum: b'

In this section we will evaluate the performance of the optimization algorithm introduced in Section 3.3.1. We study the relation between the number of bits gained using more iterations for the optimization of the vector b'. We will apply the VGS-Haar scalar approximation to the video sequence set (see Figure 7.10) and use different number of iterations to construct different approximations.

Figure 7.21 shows that more iterations helps to find a better approximation. It is possible to appreciate in the chart that for a fixed bit budget the distortion is about $\approx$ 1db when using 10 iterations instead of 1000 iterations, specially at the high quality rate. Moreover, for a fixed distortion the compression using 1000 iterations is $\approx$ 10% of the compression using 10 iterations. But using a 100 number of iterations is similar to 1000, then for reasons of speed we will use 100 iterations for almost every test in this thesis.

Notice that this improvement using more iterations in order to find a better b' is telling us that there exists more structure that is not captured by the algorithm due to the greediness and a global optimization could be a solution for this problem.

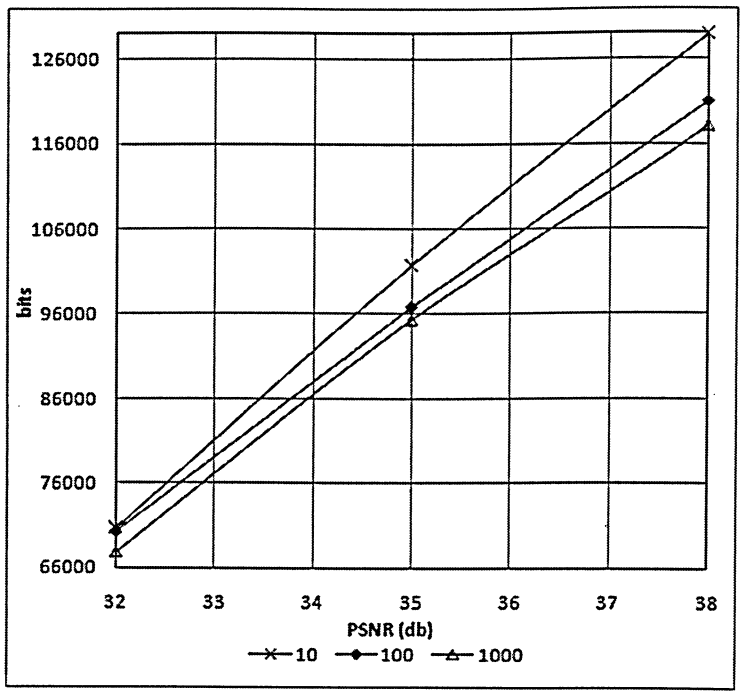


Figure 7.21: Total bit cost vs. distortion for 10, 100 and 1000 iterations

7.2 Comparisons

7.2.1 JPEG2000 Static Comparison

In this section we will compare the JPEG2000 ([13], [11]) and our algorithm in a static environment using the video sequence set from Figure 7.10. When we say "static" we mean that JPEG2000 does not make use of any temporal correlation among frames (this temporal correlation will be accounted for when we compare with MPEG). Nonetheless, we have run JPEG200 in the most favorable situation, namely we have collected the input set into a single larger image.

Due to the fact that 9 images is the best number of images for the VGS algorithm, we will use $d = 9$; also we will use the VGS-AVG (Leaves Average Approximation, see Chapter 5) algorithm that seems to be the best for video images. There are two versions of our algorithm and both outperform the JPEG2000 results. The first version is the standard VGS-AVG approximation and the second one is the VGS-AVG approximation using the Lempel-Ziv algorithm to encode the partition and the quantization map (lossless compression).

Figure 7.22 shows the total bit cost vs distortion, comparing the JPEG2000 with our algorithms. It is clear from the figure that the VGS-AVG using the Lempel-Ziv

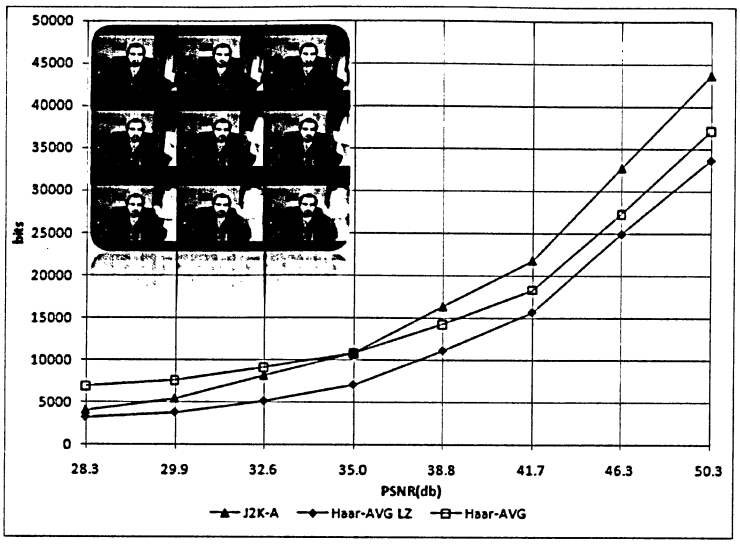


Figure 7.22: Bit cost vs. distortion per image for JPEG2000, VGS-HAAR and VGS-HAAR-LZ

encoding algorithm outperforms the others for this special video sequence. Table 7.2.1 shows numerically the same information as in Figure 7.22. The bit cost for a given distortion and the last column contains the difference between the JPEG2000 and the VGS-Haar-AVG LZ, it is possible to see that the VGS-Haar-AVG LZ is within $20\% - 35\%$ better than JPEG2000 for this case.

PSNR(db)	JPEG2000	Haar-AVG	Haar-AVG LZ	DIFF
28.3	4096	6888	3209	22%
29.9	5461	7650	3779	31%
32.6	8192	9103	5135	37%
35.0	10922	10758	7088	35%
38.8	16384	14260	11143	32%
41.7	21837	18325	15719	28%
46.3	32768	27277	24981	24%
50.3	43686	37131	33740	23%

Table 7.1: Numerical bit cost vs. distortion comparison

7.2.2 MPEG4-3 Comparisons

The video sequences have been sampled at a slow rate of 15fps (frames per second) and this is not the standard sampling rate which is at least 30fps. This means that

our video sequences have more variation among the frames. Clearly this represents a disadvantageous comparison situation for the VGS algorithm.

Hand Video Sequence

Figure 7.23 shows the original video sequence for this example, it was down sampled from 640×480 pixels to 128×128 using bi-cubic interpolation, the uncompressed video size is $450,796$ bytes

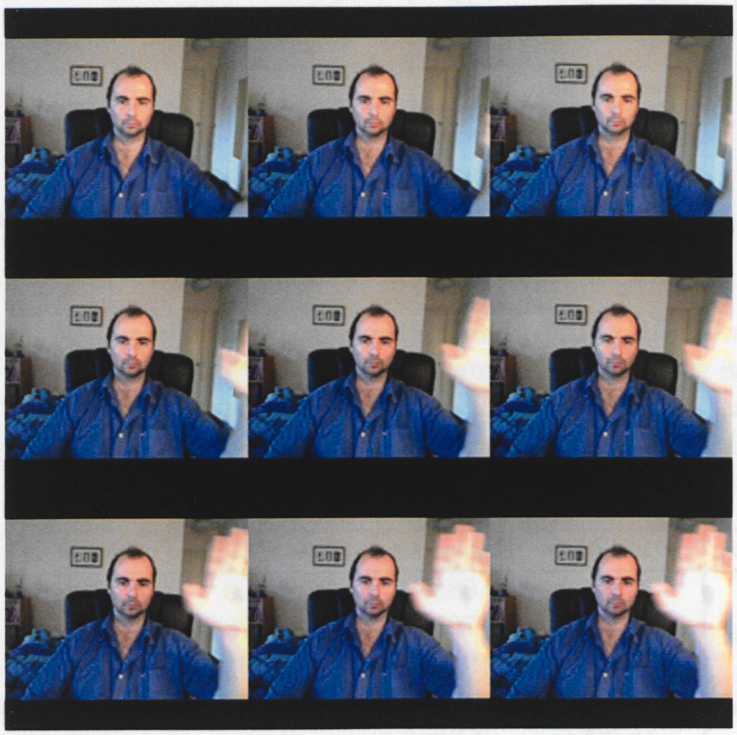


Figure 7.23: Original Video sequence: Hand, frame size:128×128, 15fps(frames per second), color depth: 24bits

Different values of PSNR and compressed sizes are shown in Table 7.2.2 for the VGS approximation and for the MPEG4 algorithm. Notice that in the present case (hand video sequence) the VGS compares favorably in terms of the PSNR. We remark, that the image quality (once sufficiently amplified) seems to be better in the MPEG4-3 approximation.

Figure 7.24 shows the VGS approximation with a $PSNR = 36.79db$

Figure 7.25 shows in detail the comparison between the MPEG4 and the VGS approximation.

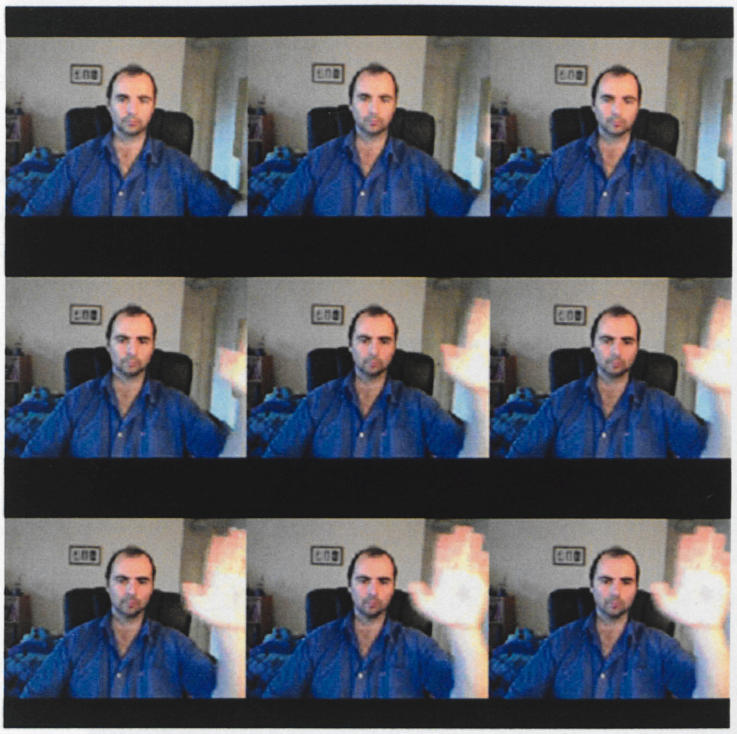


Figure 7.24: VGS Video Hand approximation PSNR=36.79db

Figure 7.25: Video Hand approximation detail, a) Input, b) MPEG4-3 approximation PSNR=36.417, c) HAAR-AVG LZ approximation PSNR=36.79

More Examples of Video Sequences

Next we provide several more examples of different video sequences. The information displayed is the same than the one displayed for the previous hand video sequence and hence should be self-explanatory.

Notice that the videos sequences: Doll and Doll 2 are slow varying videos and VGS outperforms MPEG. The remaining example, Princess, is a faster paced video an therefore there is more variation among the frames. The performance of the VGS algorithm degrades accordingly for this example.

Figure 7.26: Original Video sequence: Doll, frame size:128 $\times$ 128, 15fps(frames per second), color depth: 24bits

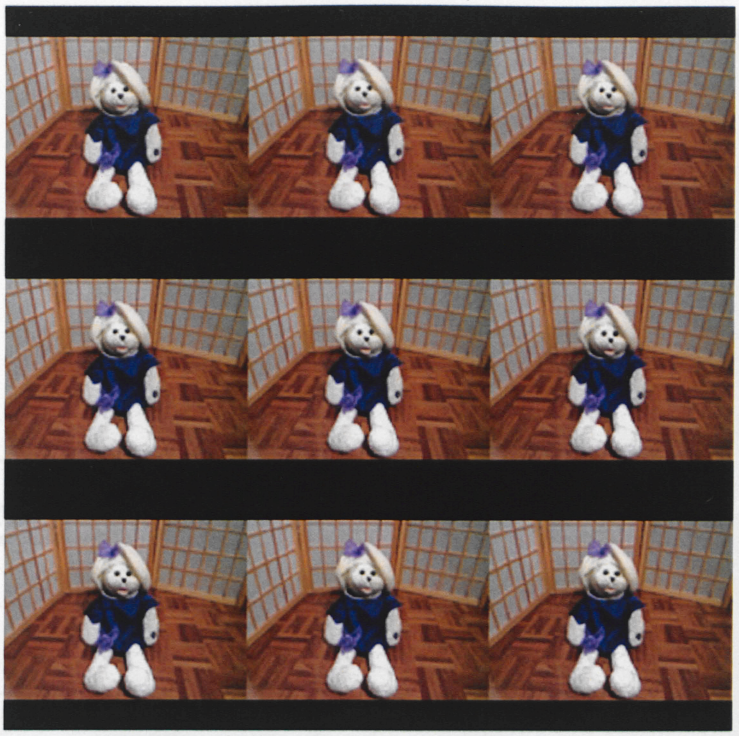


Figure 7.27: Original Video sequence: Doll 2, frame size:128 × 128, 15fps(frames per second), color depth: 24bits

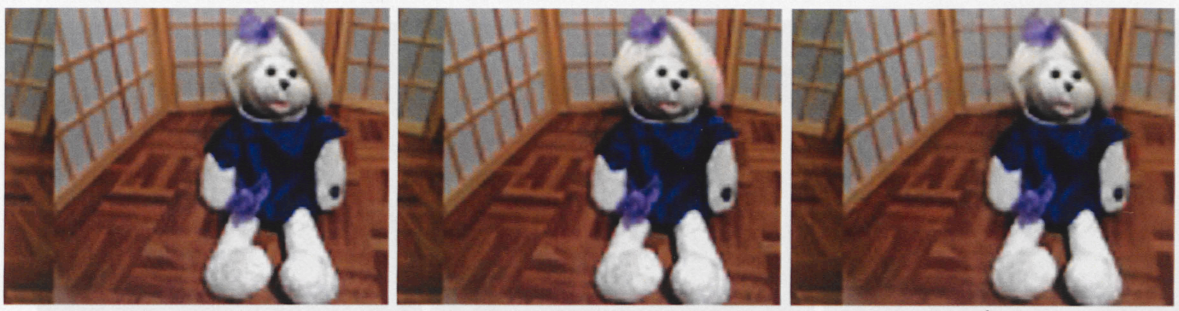


Figure 7.28: Video Doll2 approximation detail, a) Input, b) HAAR-AVG LZ approximation PSNR=38.97, c) MPEG4-3 approximation PSNR=34.93

Figure 7.29: Original Video sequence: Princess, frame size:128 × 128, 15fps(frames per second), color depth: 24bits

Figure 7.30: Video Princess approximation detail, a) Input, b) HAAR-AVG LZ approximation PSNR=33.88, c) MPEG4-3 approximation PSNR=36.67

Video	HAAR-AVG LZ				MPEG4-3		Diff %	YCbCr PSNR		
	PSNR	$C_{\mathcal{M}_\Pi}$	$C_{\mathcal{M}_Q}$	Total	PSNR	Total		Y	Cr	Cb
Hand	36.79	13721	4003	17724	36.417	19422	8.74%	39	45	45
	37.36	14390	4654	19044	36.417	19422	1.95%	40	45	45
	36.48	14581	4746	19327	36.417	19422	0.49%	41	42	42
	37.9	15474	5476	20950	36.417	19422	-7.87%	41	45	45
Doll	37.46	15492	3363	18855	35.29	19245	2.03%	42	45	45
Doll2	38.97	12714	3538	16252	34.93	20302	19.95%	42	42	42
	40.79	14510	5058	19568	34.93	20302	3.62%	45	45	45
Princess	33.88	19189	23145	42334	36.67	34460	-22.85%	37	45	45

Table 7.2: Video compression comparison, Haar-AVG LZ vs. MPEG4-3, the costs are measured in bytes and the distortion (PSNR) in decibels (db).

Chapter 8

Conclusions

The Vector Greedy Splitting algorithm is a new technique, this thesis is a first attempt to a systematic study of the algorithm. Many more mathematical and computational possibilities could be explored in the future.

We believe the VGS algorithm could be applied directly or jointly with other methods, in different applications like image compression, pattern recognition and image segmentation.

The optimal elements $\psi^{(0)}$ constructed by the VGS algorithm cronstructs approximations with well defined edges. They adapt to any given particular geometry of the edges present in the input vector. Of course, the combination of possibilities in spatial variation among the input vectors is problematic (mostly because of the combinatorial number of possibilities and the associated optimization problems). We have observed that the optimal VGS functions $\psi^{(0)}$ do not handle well smooth regions. This can be explained as follows. The optimization is carried over a large dictionary $\hat{\mathcal{C}}_A$ which does not impose any smooth constraints on its member functions. This fact and the use of the Bathtub theorem implies that the optimal functions chosen, $\psi^{(0)}$, are not smooth. In the future, one could explore the following options to alleviate this problem: $\psi^{(1)}$ may be chosen with some smooth constraints, this is possible as the function takes more than two values, smooth constraints are imposed by minimizing higher moments of the function. One could also explore the possibility of a global optimization or different ways of splitting a given atom A as suggested in Appendix C in Section C.4.

The comparison with the JPEG and JPEG2000 suggests that there are instances (given by certain type of generic video sequences) where the VGS algorithm can be successfully applied in the image compression field. The performance of the algorithm basically is related to the common information among the images.

Even though the algorithm is computationally intensive in its present form, it is comparable to some of the techniques used today for video compression.

The main conclusion of this thesis is that VGS is worthwhile, and may be considered as a possible tool for image processing.

Appendix A

Bathtub Theorem

Theorem 1. *Bathtub Principle. For a given number u and $X(w)$, a real valued measurable function, defined on Ω (a general measure space), set:*

$$\mathcal{D}_{u_1} = \{\varphi : 0 \leq \varphi(w) \leq 1 \ \text{ for all } \ w \ \text{ and } \ \int_\Omega \varphi(w)\, dP(w) = u_1\}. \tag{A.0.1}$$

Then the minimization problem

$$I = \inf_{\varphi \in \mathcal{D}_u} \int_\Omega X(w)\, \varphi(w)\, dP(w), \tag{A.0.2}$$

is solved by

$$\varphi_1(w) = 1_{\{X(w) < y_1\}}(w) + c\ 1_{\{X(w) = y_1\}}(w), \tag{A.0.3}$$

and

$$I = \int_{X(w) < y_1} X(w)\, dP(w) + c_1\ P(X = y_1). \tag{A.0.4}$$

Where

$$y_1 \equiv \sup_t \{P(X < t) \leq u_1\}, \tag{A.0.5}$$

and c_1 satisfies

$$c_1\ P(X = y_1) = u_1 - P(X < y_1). \tag{A.0.6}$$

Notice that when when F_X is continuous, y_1 satisfies $F_X(y_1) = u_1$. The above theorem is introduced an discussed in [16].

In the thesis we also need the dual version of Theorem 1, we present this result in the following Corollary.

Corollary 1. *Assume the same hypothesis as Theorem 1, then the maximization problem*

$$I = \sup_{\varphi \in \mathcal{D}_{u_2}} \int_{\Omega} X(w)\ \varphi(w)\ dP(w),$$

is solved by

$$\varphi_2(w) = \chi_{\{X > y_2\}}(w) + c_2\ \chi_{\{X = y_2\}}(w), \tag{A.0.7}$$

and

$$I = \int_{X > y_2} X(w)\ dP(w) + c_2\ P(X = y_2).$$

Where

$$y_2 = \inf\{t : P(X > t) \leq u_2\},$$

and

$$c_2\ P(X = y_2) = u_2 - P(X > y_2).$$

Notice that when when F_X is continuous, y_2 satisfies $F_X(y_2) = 1 - u_2$.

Appendix B

Measure of Quality

$$\mathrm{MSE}[i] = \frac{1}{N_s} \sum_{w \in \Omega} (X[i] - \hat{X}[i])^2 \tag{B.0.1}$$

where N_s is the number of samples in Ω (considering Ω finite). Then the total MSE is defined by

$$\mathrm{MSE}_T \equiv \sum_{i=1}^{d} \mathrm{MSE}[i] \tag{B.0.2}$$

it is possible to check that the summation is over Ω then defining the error in a given atom A

$$\mathrm{MSE}[i]^A = \frac{1}{N_s} \sum_{w \in A} (X[i] - \hat{X}[i])^2 \tag{B.0.3}$$

then

$$\mathrm{MSE}[i] = \frac{1}{N_s} \sum_{A \in \mathcal{A}} \sum_{w \in A} (X[i] - \hat{X}[i])^2 \tag{B.0.4}$$

and of course

$$\mathrm{MSE}_T = \frac{1}{N_s} \sum_{A \in \mathcal{A}} \sum_{w \in A} \sum_{i=1}^{d} (X[i] - \hat{X}[i])^2 \tag{B.0.5}$$

defining recursively the split step for a given $A \in \mathcal{A}$

$$\mathrm{MSE}_T^{n+1} = \mathrm{MSE}_T^n - \mathrm{MSE}_T^A + \mathrm{MSE}_T^{A_0} + \mathrm{MSE}_T^{A_1} \tag{B.0.6}$$

then

$$\mathrm{PSNR} = 10 \ \log_{10} \left(\frac{V^2}{\mathrm{MSE}_T} \right) \tag{B.0.7}$$

Appendix C

Convergence Proof

C.1 Convergence of Vector Greedy Splitting Approximation

In this Appendix we prove the convergence to X of the approximation $X_{\mathcal{T}_\setminus}(w)$ for almost all points $w \in \Omega$ as $n \to \infty$. We keep the base domain Ω arbitrary and will assume, for the main results, that X takes a finite number of distinct values. We have provided some of the arguments in a more general setting than the one used in the rest of the thesis. The reason for doing so is to isolate the minimal hypothesis for the corresponding proof. Moreover, some of the notation used is only meaningful for this Appendix as may conflict similar notions used throughout the rest of the thesis. As a final remark, we indicate that several notions defined elsewhere in the thesis have been repeated here, this is done to make this Appendix as self contained as possible and provide an easier reading of the result (as otherwise the reader may have to chase the definitions all over the manuscript).

Assume $\sigma(X) \subseteq \mathcal{A}$; after several general results on the VGS algorithm, the vector case will be tackled by reducing it to the scalar case. Recall the definition

$$[X, Y] = \int_\Omega \langle X(w), Y(w) \rangle \, dP(w) \tag{C.1.1}$$

where X, Y are $\mathbb{R}^d$-valued random variables, notice that in the case of $d = 1$ we have

$$[X, Y] = \int_\Omega X(w) Y(w) \, dP(w) \tag{C.1.2}$$

so the notation [,] does not indicate the value of d explicitly, therefore, readers will need to determine it from the context. In some cases, to emphasize the fact that $d = 1$, we will write $[X, Y]_1$ in the case of (C.1.2).

For a given $A \in \mathcal{A}$ we will consider a generic collection of functions satisfying the following conditions

$$\mathcal{D}_A = \{\psi : \psi(w) = 0 \text{ if } w \notin A, \int_\Omega \psi \, dP = 0, \int_\Omega \|\psi\|^2 \, dP = 1\}. \tag{C.1.3}$$

We will assume there is available a routine, which we will call `bestSplit`, such that for a given $A \in \mathcal{A}$ provides a finite number of *best functions* $\psi_A^{(i)} \in \mathcal{D}_A$, $i = 0, \ldots, I_A - 1$, and a partition of A into a finite number of *best children* $A_k \in \mathcal{A}$, $k = 0, \ldots, I_A$ (this notation is meant to convey the fact that the integer value I_A will depend on A). We will also require

$$[\psi_A^{(i)}, \psi_A^{(i')}] = 0 \text{ whenever } i \neq i'. \tag{C.1.4}$$

More properties of `bestSplit` will be specified as they are needed in later developments.

Our vector approximations are always initialized as follows

$$\psi_\emptyset^{(0)} = c \, 1_\Omega \tag{C.1.5}$$

and

$$c[i] = \frac{\int_\Omega X[i] \, dP}{\sqrt{\sum_{k=1}^{d} \left(\int_\Omega X[k] \, dP \right)^2}}. \tag{C.1.6}$$

Therefore

$$[X, \psi_\emptyset^{(0)}] \, \psi_\emptyset^{(0)}[i] = \int_\Omega X[i] \, dP. \tag{C.1.7}$$

Remark 23. *In practice we will have $I_A = 1$, for all A, in the Haar case and $I_A \leq 2$, for all A, for the full bathtub and top - bottom bathtub cases. Also $I_\emptyset = 1$ for all cases.*

Definition 10. *We will say that such $\mathcal{D}_A$ is admissible (for a given $A \in \mathcal{A}$) if the best function $\psi_A^{(0)} \in \mathcal{D}_A$ satisfies:*

$$\textit{if } X \textit{ is not constant on } A \textit{ then and } [X, \psi_A^{(0)}] \neq 0. \tag{C.1.8}$$

Remark 24. *Notice that if X is constant on A then*

$$[X, \psi_A^{(0)}] = 0, \tag{C.1.9}$$

this last equation follows from (C.1.3).

From now on we will assume the given collections $\mathcal{D}_A, A \in \mathcal{A}$ are admissible. We will impose further conditions on $\mathcal{D}_A$ whenever they are needed.

Lets define the VGS algorithm, we will indicate how the algorithm constructs a sequence of partitions Π_n indexed by $n = 0, 1, 2, \ldots$. The index n will be referred as the $n-$th. iteration of VGS. The partitions are defined recursively as indicated next. Start by setting $\Pi_0 = \{\Omega, \emptyset\}$ (notice that we explicitly include $\emptyset$ in Π_0, this will include $\psi_\emptyset$ in all our approximations) and assume, inductively, that Π_k, $k \leq n$ ($\Pi_k \subseteq \mathcal{A}$) have been constructed and are finite. Now we describe how to generate Π_{n+1}. Consider $A^* \in \Pi_n$ such that it satisfies

$$|[X, \psi^{(0)}_{A^*}]| \geq |[X, \psi^{(0)}_A]| \text{ for all } A \in \Pi_n. \tag{C.1.10}$$

Now, if

$$[X, \psi^{(0)}_{A^*}] = 0, \tag{C.1.11}$$

the algorithm VGS terminates and $\Pi_p \equiv \Pi_n$ for all $p \geq n$. Otherwise, i.e. $[X, \psi_{A^*}] \neq 0$, we set

$$\Pi_{n+1} = \Pi_n \backslash \{A^*\} \cup_{k=0,\ldots,I_A} \{A^*_k\} \tag{C.1.12}$$

where, as indicate previously, the sets A^*_k are the best children of A^*.

Remark 25. *Notice that all partitions defined above are finite. Assume the number of best children is uniformly bounded by a number M, then*

$$|\Pi_{n+1}| \leq |\Pi_n| + M. \tag{C.1.13}$$

Define the tree $\mathcal{T}_n$ as follows

$$\mathcal{T}_n \equiv \cup_{k=0}^{n} \Pi_k. \tag{C.1.14}$$

We then define the associated approximation by

$$X_{\mathcal{T}_n} \equiv \sum_{A \in \mathcal{T}_n} \sum_{i=0}^{I_A - 1} [X, \psi^{(i)}_A] \, \psi^{(i)}_A. \tag{C.1.15}$$

Therefore, the approximation given by the VGS algorithm at iteration n uses the best functions associated to the partitions created up to, and including, iteration n, more explicitly, we use the functions $\psi^{(i)}_A$ with $A \in \cup_{j=0,\ldots,n} \Pi_j$. Notice that, by our previous conventions, we are including $\psi^{(0)}_\emptyset$ in all VGS approximations.

Remark 26. *It follows from our definitions that if $A \in \Pi_n$ and X restricted to A is constant then A will never be best split by VGS at iterations $p \geq n$.*

Requiring the appropriate conditions, we will prove a series of results that taken together will prove

$$\lim_{n\to\infty} X_{\mathcal{T}_n}(w) = X(w) \quad \text{for almost all } w \in \Omega. \tag{C.1.16}$$

The above limit will actually be finite if X is a simple function, namely if X takes a finite number of values. For a given X consider its range of values on $A \in \mathcal{A}$

$$\mathcal{R}_A(X) \equiv \{X(w) : w \in A\}. \tag{C.1.17}$$

Therefore X is simple if $|\mathcal{R}_\Omega(X)| < \infty$, for some results we will need to assume that X is simple; whenever this hypothesis is needed, it will be indicated explicitly.

We will define an increasing sequence of orthonormal systems $\mathcal{H}_n$, for $n \geq 0$ corresponding to the n-th. iteration of the VGS algorithm, as follows: $\mathcal{H}_0 \equiv \{\mu_0 \equiv \psi_\emptyset^{(0)}\}$ also, assume, recursively that $\mathcal{H}_n = \{\mu_0, \ldots, \mu_{k_n}\}$ has been constructed. We then let,

$$\mathcal{H}_{n+1} \equiv \mathcal{H}_n \cup_{i=0,\ldots,I_{A^*}-1} \{\psi_{A^*}^{(i)}\} \tag{C.1.18}$$

where A^* is the set in (C.1.10), also set $\mu_{k_n+i+1} \equiv \psi_{A^*}^{(i)}$ for $i = 0, \ldots, I_{A^*} - 1$.

Lemma 1. *Given $A \in \mathcal{Q}_{n_0}$ and if X restricted to A is not constant, then there exists $n_1 > n_0$ such that VGS splits A before or at iteration n_1.*

Proof. From

$$\sum_{k=0,\ldots} |[X, u_k]|^2 \leq ||X||^2, \tag{C.1.19}$$

it follows

$$\lim_{k\to\infty} [X, u_k] = 0. \tag{C.1.20}$$

By hypothesis $|[X, \psi_A^{(0)}]| > 0$, then using (C.1.20) find n_1 such that $[X, u_k]| < |[X, \psi_A]|$ for all $k > n_1$; by the definition of VGS we then know that VGS will best split A before or at iteration n_1. □

To proceed further we need to introduce some more assumptions, namely we will consider only the cases when the best functions $\psi_A^{(i)}$ take only $I_A + 1$ values and if $\mathcal{R}_A(\psi_A^{(j)}) = \{r_{j,0}, r_{j,1}, \ldots, r_{j,I_A}\}$, $j = 0, \ldots, I_A - 1$ then define

$$A_k \equiv \{w \in A : \psi_A^{(0)} = r_{0,k}\}, \tag{C.1.21}$$

and also assume

$$A_k = \{w \in A : \psi_A^{(j)} = r_{j,k}\} \text{ for all } j = 0, \ldots, I_A - 1 \tag{C.1.22}$$

notice that $A_k \neq \emptyset$. Moreover, we also assume the best children A_k are only of the following form (where c_1 and c_2 are arbitrary constants):

$$A_k = \{w \in A : c_1 \leq X[b'](w) \leq c_2\} \text{ for some } b' \in S^d. \tag{C.1.23}$$

Remark 27. *Equation (C.1.23) could be replaced by the weaker hypothesis* $A_k \in \sigma(X)$.

Under these last assumptions we then have the following.

Corollary 2. *If* X *is a simple function that takes* q *distinct values, then VGS terminates in a finite number of steps* N *which satisfies* $N \leq q \leq |\Omega|$.

Proof. Let $\mathcal{R}_\Omega(X) = \{x_1, \ldots, x_q\}$ be the finite set of the range values of X on Ω. Also define the generating sets $C_{x_i} \equiv \{w \in \Omega : X(w) = x_i\}$ for $i = 1, \ldots, q$. It is easy to see that any $A \in \Pi_n$, where $n \geq 0$ is arbitrary, can be written as a finite union of generating sets. It then follows that each best split of a given $A \in \Pi_n$ will produce best children which can be written as union of a strictly smaller number of generating sets. From Lemma 1 this process will continue until the remaining atoms are made up of a single generating set or X is constant on all these atoms. In any of these cases X will be constant for all the given atoms and hence VGS terminates. If we let N denote the smallest integer such that $\Pi_n = \Pi_N$ for all $n \geq N$, a simple counting argument proves that $N \leq q$. □

The argument used in the proof of Corollary 2 also proves the following corollary.

Corollary 3. *Let* X *be a simple function and denote with* Π_N *the terminating (i.e.* $\Pi_n = \Pi_N$ *for all* $n \geq N$*) partition, which exists by Corollary 2, then:*

$$X \text{ restricted to each } A \in \Pi_N \text{ is constant.} \tag{C.1.24}$$

In the next section we will provide general conditions under which

$$X_{\mathcal{T}_n}(w) = \frac{1}{P(A)} \int_A X(w)\, dP(w) \text{ for all } A \in \Pi_n \text{ and for all } n \geq 0, \tag{C.1.25}$$

holds.

Clearly (C.1.25) and Corollary 3 will prove the following theorem.

Theorem 2. *Assume* X *is a simple function, then under the hypothesis for which (C.1.25) holds we will have*

$$X_{\mathcal{T}_N}(w) = X(w) \text{ a.e. in } \Omega. \tag{C.1.26}$$

C.2 Reduction to Scalar case

The aim of this section is to prove (C.1.25). To achieve this goal we will go through a series of results and notation. We will also need to introduce more specific properties of `bestSplit`.

We assume, as usual, that an event $A \in \mathcal{A}$ is fixed and, for simplicity, remove it from the notation. In particular ψ_A will be written as ψ.

At this point we need to specify the routine `bestSplit` even further; for this we will assume that the best function $\psi^{(0)}$ is given by

$$\psi^{(0)} = a\ \varphi_1 + b\ \varphi_2 \tag{C.2.1}$$

for two *fixed* densities φ_i, i = 1, 2. Moreover, we also have $b = ||b||\ \hat{b}'$ where $\hat{b}'$ is the optimal b' given by

$$\hat{b}'_i = \frac{\frac{1}{u_2}\int_A X[i]\ \varphi_2\ dP - \frac{1}{u_1}\int_A X[i]\varphi_1\ dP}{\sqrt{\sum_{k=1}^{d}\left(\frac{1}{u_2}\int_A X[k]\ \varphi_2 dP - \frac{1}{u_1}\int_A X[k]\ \varphi_1 dP\right)^2}}. \tag{C.2.2}$$

where, as usual, $u_i = \int_A \varphi_i dP$. Also, an expression for $||b||$ is provided in Chapter II.

We will have to distinguish between a given vector valued random variable ψ and the associated scalar basis vector ψ_s. So if $\psi_A = \psi = a\varphi_1 + b\varphi_2$, we will use the following notation for the associated scalar function

$$\psi_{A,s} = \psi_s = d_1\varphi_1 + d_2\varphi_2, \tag{C.2.3}$$

of course,

$$\int_\Omega \psi_s(w)\ dP(w) = 0, \tag{C.2.4}$$

$$\int_\Omega \psi_s^2(w)\ dP(w) = 1. \tag{C.2.5}$$

Proposition 3. *Fix an atom A and everything will be relative to this atom. Then*

$$[X, \psi^{(0)}]\ \psi^{(0)}[i](w) = [X[i], \psi_s^{(0)}]_1\ \ \psi_s^{(0)}(w)\ \ a.e.\ in\ \Omega. \tag{C.2.6}$$

Proof. The proof follows, essentially, by plugging the expression (C.2.2) into the left hand side of (C.2.6) and comparing with the right hand side of (C.2.6). Computing we obtain,

$$[X, \psi^{(0)}] = ||b||\ u_2\left(\frac{1}{u_2}\int_A X[\hat{b}']\ \varphi_2\ dP - \frac{1}{u_1}\int_A X[\hat{b}']\ \varphi_1\ dP\right) = \tag{C.2.7}$$

$$||b||\ u_2 \left[\sum_{i=1}^{d} \hat{b}'_i \left(\frac{1}{u_2}\int_A X[i]\ \varphi_2\ dP - \frac{1}{u_1}\int_A X[i]\ \varphi_1\ dP\right)\right], \tag{C.2.8}$$

then using (C.2.2) in this last expression we obtain

$$[X, \psi^{(0)}] = ||b||\ u_2 \sqrt{\sum_{k=1}^{d} \left(\frac{1}{u_2}\int_A X[k]\ \varphi_2 dP - \frac{1}{u_1}\int_A X[k]\ \varphi_1 dP\right)^2}. \tag{C.2.9}$$

Now, in order to write $[X, \psi^{(0)}]\ \psi^{(0)}[i]$ we do the following manipulations

$$\psi^{(0)} = a\varphi_1 + b\varphi_2 = \left(\frac{-b\ u_2\ \varphi_1}{u_1} + b\varphi_2\right) = ||b||\ u_2\ \hat{b}'\ \left(\frac{\varphi_2}{u_2} - \frac{\varphi_1}{u_1}\right). \tag{C.2.10}$$

Therefore

$$\psi^{(0)}[i] = ||b||\ u_2\ \hat{b}'_i\ \left(\frac{\varphi_2}{u_2} - \frac{\varphi_1}{u_1}\right). \tag{C.2.11}$$

Using (C.2.9) and (C.2.11) we obtain

$$[X, \psi^{(0)}]\ \psi^{(0)}[i] = ||b||^2\ u_2^2\ \left(\frac{\varphi_2}{u_2} - \frac{\varphi_1}{u_1}\right)\left(\frac{1}{u_2}\int_A X[i]\ \varphi_2\ dP - \frac{1}{u_1}\int_A X[i]\ \varphi_1\ dP\right). \tag{C.2.12}$$

We concentrate now on the right hand side of (C.2.6). Let $d'_2 = \frac{d_2}{|d_2|}$, so $d'_2 \in \{-1, 1\}$ and write the scalar basis function as follows

$$\psi_s = d_1\ \varphi_1 + d_2\ \varphi_2 = d_2\ u_2\ \left(\frac{\varphi_2}{u_2} - \frac{\varphi_1}{u_1}\right) = \tag{C.2.13}$$

$$|d_2|\ u_2\ d'_2\ \left(\frac{\varphi_2}{u_2} - \frac{\varphi_1}{u_1}\right).$$

Notice now that

$$|d_2| = ||b||. \tag{C.2.14}$$

Therefore

$$\psi_s^{(0)} = ||b||\ u_2\ d'_2\ \left(\frac{\varphi_2}{u_2} - \frac{\varphi_1}{u_1}\right). \tag{C.2.15}$$

Moreover

$$[X[i], \psi_s^{(0)}]_1 = d_2\ u_2\ \left(\frac{1}{u_2}\int_A X[i]\ \varphi_2\ dP - \frac{1}{u_1}\int_A X[i]\ \varphi_1\ dP\right) = \tag{C.2.16}$$

$$[X[i], \psi_s^{(0)}]_1 = ||b||\ u_2\ d'_2 \left(\frac{1}{u_2}\int_A X[i]\ \varphi_2\ dP - \frac{1}{u_1}\int_A X[i]\ \varphi_1\ dP\right).$$

Therefore, (notice that $d_2'^2 = 1$)

$$[X[i], \psi_s^{(0)}]_1 \, \psi_s^{(0)} = (||b||\, u_2\, d_2')^2 \left(\frac{\varphi_2}{u_2} - \frac{\varphi_1}{u_1}\right) \left(\frac{1}{u_2}\int_A X[i]\, \varphi_2\, dP - \frac{1}{u_1}\int_A X[i]\, \varphi_1\, dP\right) = \tag{C.2.17}$$

$$||b||^2\, u_2^2 \left(\frac{\varphi_2}{u_2} - \frac{\varphi_1}{u_1}\right) \left(\frac{1}{u_2}\int_A X[i]\, \varphi_2\, dP - \frac{1}{u_1}\int_A X_i\, \varphi_1\, dP\right).$$

We just checked that (C.2.17) is equal to (C.2.12). □

Using (C.1.7) and Proposition 3 we obtain

Proposition 4.

$$\sum_{A\in\mathcal{T}_n} [X, \psi_A^{(0)}]\, \psi_A^{(0)}[i](w) = \sum_{A\in\mathcal{T}_n} [X[i], \psi_{A,s}^{(0)}]_1\, \psi_{A,s}^{(0)}(w). \tag{C.2.18}$$

We need to prove a similar result for the case when we add more functions to the node, we will only consider the case of $\psi_A^{(1)}$, i.e. will consider only those nodes for which $I_A \leq 2$; in fact if $I_A = 1$ there is no function $\psi_A^{(1)}$ and so we may assume $I_A = 2$ without loss of generality, this restriction will cover all the cases considered in this thesis. From our hypothesis, it follows that $\psi_A^{(0)}$ (as well as $\psi_A^{(1)}$) takes only three distinct values. In particular we can write the scalar function as follows

$$\psi_s^{(0)} = \sum_{k=0}^{2} a_k \mathbf{1}_{A_k}. \tag{C.2.19}$$

Consider now

$$\psi_s^{(1)} = \sum_{k=0}^{2} e_k \mathbf{1}_{A_k} \tag{C.2.20}$$

to be the unique function satisfying the following three conditions

$$||(\psi_s^{(1)})^2|| = 1, \qquad \int \psi_s^{(1)} dP(w) = 0, \qquad [\psi_s^{(1)}, \psi_s^{(0)}]_1 = 0. \tag{C.2.21}$$

In the present scalar case it is easy to find the solution to the above system of equations. Here is the explicit solution:

$$e_2 = \frac{-\, e_0\, P(A_0)\, (a_1 - a_0)}{P(A_2)\, (a_1 - a_2)}, \tag{C.2.22}$$

$$e_1 = \frac{-\, e_0\, P(A_0)\, (a_2 - a_0)}{P(A_1)\, (a_2 - a_1)}, \tag{C.2.23}$$

and

$$e_0^2 = \frac{P(A_1)\ P(A_2)\ (a_2 - a_1)^2}{P(A_0)\ [P(A_1)P(A_2)(a_2 - a_1)^2 + P(A_0)P(A_2)(a_2 - a_0)^2 + P(A_0)P(A_1)(a_1 - a_0)^2]}. \tag{C.2.24}$$

Therefore taking $e_0 = \pm\sqrt{e_0^2}$ and using the above equations gives a solution (the $\pm$ sign does not affect the scalar components, i.e. the inner product times the basis element).

Define now $\psi^{(1)}$ as follows

$$\psi^{(1)}[i](w) \equiv \frac{[X[i], \psi_s^{(1)}]_1\ \psi_s^{(1)}(w)}{\sqrt{\sum_{j=1}^{d}[X[j], \psi_s^{(1)}]_1^2}} \tag{C.2.25}$$

Proposition 5. *The constructed vector valued function $\psi_A^{(1)}$ is 0 outside A and it takes three distinct values on A, the pre-images of these constant values are exactly the best children of A, namely the sets A_k $k = 0, 1, 2$. Moreover,*

$$[\psi^{(0)}, \psi^{(1)}] = 0, \qquad ||\psi^{(1)}|| = 1 \tag{C.2.26}$$

and

$$[X, \psi^{(1)}]\ \psi^{(1)}[i](w) = [X[i], \psi_s^{(1)}]_1\ \psi_s^{(1)}(w) \tag{C.2.27}$$

Proof. The fact that the best children of A are given as pre-images of constant values of $\psi^{(1)}$ follows directly from (C.2.20) and (C.2.25). From the definition (C.2.25) it follows that in order to prove (C.2.27) it is enough to prove the following equality

$$[X, \psi^{(1)}]^2 = \sum_{i=1}^{d}[X[i], \psi_s^{(1)}]_1^2. \tag{C.2.28}$$

This last equation as well as the other properties follow from simple computations. □

The previous definitions and arguments prove the following.

Proposition 6.

$$X_{\mathcal{T}_n}[k](w) \equiv \sum_{A\in\mathcal{T}_n}\sum_{i=0}^{I_A-1}[X, \psi_A^{(i)}]_1 \quad \psi_A^{(i)}[k](w) = \sum_{A\in\mathcal{T}_n}\sum_{i=0}^{I_A-1}[X[k], \psi_{A,s}^{(i)}]_1\ \psi_{A,s}^{(i)}(w). \tag{C.2.29}$$

Therefore, in order to prove (C.1.25) it is enough to prove the following identity

$$\sum_{A\in\mathcal{T}_n}\sum_{i=0}^{I_A-1}[X[k],\psi^{(i)}_{A,s}]\ \psi^{(i)}_{A,s}(w)=\frac{1}{P(A)}\int_A X(w')\ dP(w') \tag{C.2.30}$$

for almost all $w\in A$ and for all $A\in\Pi_n$ and for all $n\geq 0$.

Equation (C.2.30) will follow from the general result, for scalar expansions, provided below. We will need the analogous definition to the one in (C.2.31) but for the scalar case.

We will define an increasing sequence of orthonormal systems $\mathcal{G}_n$, for $n\geq 0$ corresponding to the n-th. iteration of the VGS algorithm, as follows: $\mathcal{G}_0\equiv\{u_0\equiv 1_\Omega\}$ also, assume, recursively that $\mathcal{G}_n=\{u_0,\ldots,u_{k_n}\}$ has been constructed. We then let,

$$\mathcal{G}_{n+1}\equiv\mathcal{G}_n\cup_{i=0,\ldots,I_{A^*}-1}\{\psi^{(i)}_{A^*,s}\} \tag{C.2.31}$$

where A^* is the set in (C.1.10), also set $u_{k_n+i+1}\equiv\psi^{(i)}_{A^*,s}$ for $i=0,\ldots,I_{A^*}-1$.

For the *scalar case* equation (C.2.30) (and so (C.1.25 is also true) follows from the following proposition

Proposition 7. *Let $\mathcal{G}_n=\{u_0,\ldots,u_{k_n}\}$, also define*

$$V_n\equiv span\ \{u\in\sigma(u_0,\ldots,u_{k_n})\}, \tag{C.2.32}$$

and

$$U_n\equiv span\ \mathcal{G}_n. \tag{C.2.33}$$

Assume,

$$dim\ U_n=dim\ V_n \tag{C.2.34}$$

then for a given random variable X (i.e. real valued measurable function) we have

$$E(X|u_0,\ldots,u_{k_n})=P_{U_n}X=\sum_{i=0}^{k_n}[X,u_i]_1\ u_i. \tag{C.2.35}$$

Proof. From the definition of conditional expectations we have

$$E(X|u_0,\ldots,u_{k_n})=P_{V_n}X, \tag{C.2.36}$$

where $P_{V_n}X$ denotes the projection of X onto the closed subspace V_n. Notice that $\mathcal{G}_n\subseteq V_n$, therefore (C.2.34) implies that $\mathcal{G}_n$ is an orthonormal basis for V_n, the result then follows. □

To complete the scalar case, it then only remains to establish (C.2.34) for the three cases studied in this thesis, namely Haar, MD and Full Bathtub, this is done through the following proposition.

Proposition 8. *The VGS algorithm satisfies (C.2.34).*

Proof. Notice that for for $n = 0$ we are taking $k_0 = 0$ and $u_0 = 1_\Omega$ therefore $\dim U_0 = 1$, clearly we also have $\dim V_0 = 1$. We will proceed by induction, so assume (C.2.34) holds for n and we will prove it holds for $n+1$. Consider the case that X is constant for all $A \in \Pi_n$ we have $V_k = V_n$ for all $k \geq n$, therefore the results holds from the inductive hypothesis. Otherwise, i.e. X is not constant on Π_n, it then follows from (C.3.2) and (C.3.3) that $\dim V_{n+1} = \dim V_n + I_A$ but we also have from construction, namely (C.2.31), $\dim U_{n+1} = \dim U_n + I_A$. □

C.3 Properties of bestSplit

In this section we will write the complete properties satisfied by `bestSplit` as used in VGS and connect its properties with the development of the present Appendix.

Let $\mathcal{C}_A$ denote the collection of VGS functions for the given event A, notice that $\mathcal{C}_A$ is admissible. The routine `bestSplit` provides a finite number of *best functions* $\psi_A^{(i)} \in \mathcal{C}_A$, $i = 0, \ldots, I_A - 1$, and a partition of A into a finite number of *best children* $A_k \in \mathcal{A}$, $k = 0, \ldots, I_A$. We will also require

$$[\psi_A^{(i)}, \psi_A^{(i')}] = 0 \text{ whenever } i \neq i'. \tag{C.3.1}$$

Moreover, the best functions $\psi_A^{(i)}$ take only I_A+1 values and if $\mathcal{R}_A(\psi^{(j)}) = \{r_{j,0}, r_{j,1}, \ldots, r_{j,I_A}\}$, $j = 0, \ldots, I_A - 1$ then define

$$A_k \equiv \{w \in A : \psi^{(0)} = r_{0,k}\}, \tag{C.3.2}$$

also,

$$A_k = \{w \in A : \psi^{(j)} = r_{j,k}\} \text{ for all } j = 0, \ldots, I_A - 1 \tag{C.3.3}$$

notice that $A_k \neq \emptyset$. The best children A_k are only of the following form (where c_1 and c_2 are arbitrary constants):

$$A_k = \{w \in A : c_1 \leq X[b'](w) \leq c_2\}. \tag{C.3.4}$$

Finally, best split also satisfies (for a definition of $\hat{\mathcal{C}}$ the reader should refer to (3.3.13) and Remark 9)

$$[X, \psi_A^{(0)}] = \sup_{\psi \in \hat{\mathcal{C}}_A} [X, \psi]. \tag{C.3.5}$$

Remark 28. *In reality, (C.3.5) as described is not the full story as, for each of the three cases (Haar case, Full Bathtub and MD case) we require some conditions and/or special constraints.*

The construction of $\psi_A^{(0)}$ is fully described in Chapter 3. Therefore, in order to completely specify `bestSplit` (for the cases considered in the thesis), it remains only to describe the construction of $\psi_A^{(1)}$, this is done by (C.2.25).

Given the above properties of `bestSplit` we can prove the following proposition.

Proposition 9. $[X, \psi_A^{(0)}] = 0$ *if and only if* X *restricted to* A *is constant.*

Remark 29. *If* A *is a given atom in a VGS partition, notice that the above proposition proves that VGS does not split* A *any further if and only of* X *restricted to* A *is constant.*

Proof. Assume first that $[X, \psi_A^{(0)}] = 0$ and that X restricted to A is not constant, this implies that there exists $\psi \in \mathcal{C}_A$ such that $[X, \psi] \neq 0$. Without loss of generality we may assume $[X, \psi] > 0$, given (C.3.5), this contradicts our assumption.

Conversely, assume that X restricted to A is constant, then the fact that $\mathbf{E}(\psi^{(0)})$ implies $[X, \psi_A^{(0)}] = 0$. □

C.4 New Formulas for Alternative VGS

So far the VGS algorithm runs by optimizing $[X, \psi]$ which gives the best VGS function ψ^0, a-posteriori we obtain ψ^1, it is reasonable to directly optimize the functional

$$[X, \psi^{(0)}]^2 + [X, \psi^{(1)}]^2 \tag{C.4.1}$$

by proposing the Bathtub solutions for $\psi^{(0)}$ and for each of this proposed functions computing $\psi^{(1)}$ using (C.2.25). Notice that $[X, \psi^{(1)}]^2$ can be computed via (C.2.28).

Bibliography

[1] D. Alani, A. Averbuch and S. Dekel, *"Image coding with geometric wavelets"*, IEEE Transactions on Image Processing, Vol.16, No.1, pp. 69-77, 2007.

[2] P.J. Catuogno, S.E. Ferrando, A.L. Gonzalez *"Adaptive martingale approximations."*, *in press* (2007).

[3] Heon Ho Choe *"Computational Ergodic Theory"*. Springer Verlag (2005).

[4] A. Cohen, W. Dahmen, I. Daoubechies and R. De Vore *"Tree approximation and and optimal encoding"*, Applied and Computational Harmonical Analysis, Vol. 11,pp. 192-226, 2001.

[5] Geoff Davis and Aria Nosratinia, *"Wavelet-based image coding: an overview"*, Applied and Computational Control, Signals, and Circuits, Vol. 1, No. 1, Spring 1998.

[6] D.L. Donoho, M. Vetterli, R.A. DeVore and I. Daubechies, *"Data compression and harmonic analysis."*, IEEE Transaction on Information Theory, Vol 44, No.6, pp. 2435-2476, 1998.

[7] S.E.Ferrando, E.J. Doolittle, A.J.Bernal, L.J.Bernal. *"Probabilistic matching pursuit with gabor dictionaries"*, Signal Processing, Vol. 80, Issue 10, pp. 2099-2120, 2000.

[8] R. Gonzalez, R. Woods, *"Digital Image Processing"*, Second Edition, Printice Hall, pp. 282-510, 2003.

[9] Y. Huang, I. Pollak, M.N. Do, and C.A. Bouman. *"Fast search for best representations in multitree dictionaries."* IEEE Transactions on Image Processing, Vol. 15, No. 7, pp. 1779-1793, July 2006.

[10] Huffman, D.A. *"A method for the construction of minimum redundancy codes."* In Proceedings IRE, Vol. 40, pp. 1098-1101, 1962.

[11] *"Independent JPEG Group"*, [Online]. Available: http://www.ijg.org/.,

[12] A. Jensen and A. la Cour-Harbo *"Ripples in Mathematics. The Discrete Wavelet Transform"*. Springer (2001).

[13] *"The JPEG Committee"*, [Online]. Available: http://www.jpeg.org/.

[14] Evgeny Klavir *"Adaptive Vector Greedy Splitting Algorithm"* Ryerson University, M.Eng. Thesis, 2007.

[15] S. Kirkpatrick, C. D. Gelatt ans M. P. Vecchi, *"Optimization by simulated annealing"* Science, Vol. 220, No. 4598, pp. 671-680, 1983.

[16] E.H. Lieb and M. Loss. *"Analysis"*, 2nd. edition. Graduate Studies in Mathematics, Volume 14. American Mathematical Society (2001).

[17] S. Mallat and Z. Zhang (1993). *"Matching pursuits with time-frequency dictionaries"*. IEEE Transactions of Signal Processing, Vol. 41, pp. 3397-3415.

[18] S. Mallat. *"A Wavelet Tour of Signal Processing"*. Academic Press, second edition (1999)

[19] H. Radha, M. Vetterli and R. Leonardi *"Image compression using binary space partitioning trees"*, IEEE Transactions on Image Processing, Vol. 5, No. 12, pp. 1610-1624, 1996.

[20] A. Said and W. Pearlman, *"A new, fast, and efficient image codec based on set partitioning in hierarchical trees"*, IEEE Transactions on Circuits and Systems For Video Technology, Vol. 6, No. 3, pp. 243-250, 1996.

[21] A. Said and W. Pearlman, *"An image multiresolution representation for lossless and lossy compression"*, IEEE Transaction on Image Processing, Vol. 5, No.9, pp. 1303-1310, 1996.

[22] J.M. Shapiro, *"Embedded image coding using zerotrees of wavelet coefficients"*, IEEE Transactions on Signal Processing, Vol.41 - No.12, pp.3445-3462, 1993.

[23] R. Shukla, L. Dragotti, M.N. D and M. Vetterli, *"Rate-distortion optimized tree-structured compression algorithms for piecewise polynomial images"*, IEEE Transactions on Image Processing, Vol. 14, No.3, pp. 343-359, 2005.

[24] Taubman, D. *"High performance scalable image compression with EBCOT"*, IEEE Transactions on Image Processing, Vol. 9, No.7, pp. 1158 - 1170, 2000.

(4) 3L-11-4