ADAPTIVE ORTHONORMAL BASES FOR VIDEO COMPRESSION

Ariel J. Bernal[1] *and Sebastian E. Ferrando*[2]

Ryerson University, Department of Mathematics, 350 Victoria St. Toronto, Ontario, M5B 2K3, CANADA. [1] arielbernal@gmail.ca
[2] ferrando@ryerson.ca

ABSTRACT

The paper describes the construction of a vector valued orthonormal basis adapted to an input sequence of video frames. The construction relies on an optimization step that singles out common discontinuities present in the input vector. This is achieved by constructing a sequence of partitions in the common domain of the input sequence. The resulting approximation is a vector valued martingale that converges pointwise to the given set of images. Output from a software implementation, based on standard test suites of video sequences, is described.

1. INTRODUCTION

We introduce an algorithm for the simultaneous approximation of a given collection of images defined on a common, arbitrary domain Ω. The set of input images is collected into a single input vector. The algorithm is based on an optimized construction of vector valued basis functions adapted to arbitrary geometrical discontinuities of this input vector. The paper introduces the basic algorithm and concentrates in describing the application to video compression.

The algorithm to be introduced will be called Vector Greedy Splitting Algorithm (VGS), it constructs a tree which is associated to a sequence of partitions of Ω. Elements of a partition of Ω will be called *atoms*. References [1], [2] and [3] 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.

As an alternative to the above described situation, the approach introduced in this paper 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.

Mathematical properties of the algorithm for the scalar case are described in [4], further mathematical developments for the vector case will be described elsewhere. Our approach is closely related to restricted non-linear approximations and greedy algorithms, these relationships are explained in [4].

The paper is organized as follows, Section 2 provides the basic definitions and computational setup. Section 2.1 briefly describes the optimization that forms the core of the VGS construction. Section 3 describes formally the VGS algorithm and it describes how it can be used to approximate the input vector. Section 4 describes how to do transform compression with the VGS algorithm. Sections 5 and 6 describe the data structures needed to bit encode the VGS approximation. Section 7 illustrates the performance of the algorithm in several standard data sets. Section 8 summarizes the paper.

2. GENERAL NOTATION AND DEFINITIONS

Given a set of inputs images, we consider each such a set as a vector valued random variable in a Hilbert space $L^2(\Omega, R^d)$ associated to the probability space $(\Omega, \mathcal{A}, P)$. $\mathcal{A}$ is a given σ-algebra. Elements from $L^2(\Omega, R^d)$ are vector valued random variables $X\colon \Omega \to R^d$, $X(w) = (X[1](w), \ldots, X[d](w))$, the components $X[i]$ will be the given input images.

In the applications of this paper we will consider: the images $X[i]$ as coming from a sequence of video frames, hence d represents the number of frames used as input to our VGS construction. Ω will be the set of pixels, P the uniform measure on Ω and $\mathcal{A}$ the collection of all subsets of Ω.

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

$$[X, Y] \equiv \int_\Omega \langle X(w), Y(w)\rangle \, dP(w), \tag{1}$$

where $\langle\ ,\ \rangle$ is the Euclidean inner product in R^d, defined by, $\langle X(w), Y(w)\rangle = \sum_{i=1}^d X[i](w)\, Y[i](w)$.
Remark: We will write $[\ ,\ ]_1$ (instead of simply $[\ ,\ ]$) whenever we are dealing with the case of $d = 1$.
Definition: A function $\psi_A\colon \Omega \mapsto R^d$ is called a (vector valued) Haar function on A if there exists $A \in \mathcal{A}$ and the

following conditions are satisfied

$$\psi_A(w) = a\,\mathbf{1}_{A_0}(w) + b\,\mathbf{1}_{A_1}(w) \quad \forall w \in \Omega, \tag{2}$$

where $a, b \in R^d$ and

$$A_0, A_1 \in \mathcal{A}, i = 0, 1, A_0 \cap A_1 = \emptyset, A_0 \cup A_1 = A. \tag{3}$$

We also require

$$\int_\Omega \psi_A(w)\,dP(w) = 0, \int_\Omega \|\psi_A(w)\|^2\,dP(w) = 1. \tag{4}$$

Whenever A is understood we will avoid the use of the subscript by writing ψ instead ψ_A. We denote with $\mathcal{C}_A \subset L^2$ the space of all Haar functions on A.

2.1. INNER PRODUCT MAXIMIZATION USING THE BATHTUB THEOREM

The VGS algorithm introduced in the next section relies on the maximization of the inner products $[X, \psi]$. The goal of this section is to setup for computation the quantity $[X, \psi]$ for the case when $\psi \in \mathcal{C}_A$. To this end we introduce the following notation $u_0 = P(A_0),\ u_1 = P(A_1)$. Using (4) it follows that

$$a = \frac{-b\,u_1}{u_0},\ \ \|b\| = \sqrt{\frac{u_0}{P(A)\,u_1}}. \tag{5}$$

For a given set of input signals X and a given $A \in \mathcal{A}$ we would like to compute $\sup\limits_{\psi \in \mathcal{C}_A} [X, \psi]$. Replacing the definition of ψ in equation (2) we obtain

$$[X, \psi] = \|b\|\ P(A)(\frac{1}{P(A)}\int_A \langle X(w), b'\rangle\,dP(w) -$$

$$\frac{1}{u_0}\int_{A_0} \langle X(w), b'\rangle\ dP(w)) \text{ where } b' = \frac{b}{\|b\|}. \tag{6}$$

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

Equations (6) and (5) imply that the inner product depends on the quantities b', u_0 and A_0; notice that $u_0 \in (0, P(A))$. It follows that the supremum depends only on the same list of variables and can be written as iterated suprema as follows

$$\sup_{\psi \in \mathcal{C}_A} [X, \psi] = \sup_{b'}\ \sup_{u_0}\ \sup_{A_0 \in \mathcal{A}, P(A_0) = u_0} [X, \psi]. \tag{7}$$

It can be proven by means of the bathtub theorem [5] that we can simplify (7) to the following computation

$$\sup_{\psi \in \mathcal{C}_A} [X, \psi] = \sup_{b'}\left[\ \sup_{y_0}\ [X, \hat{\psi}]\right], \tag{8}$$

where y_0 is an independent variable belonging to $Range(\langle X, b'\rangle)$ and $\hat{\psi}(b', y_0) = a\mathbf{1}_{\hat{A}_0} + b\mathbf{1}_{\hat{A}_1}$

$$\hat{\psi}(b', y_0) = a\mathbf{1}_{\{\langle X(w), b'\rangle < y_0\}} + b\mathbf{1}_{\{\langle X(w), b'\rangle \geq y_0\}}, \tag{9}$$

a and b are functions of b' and u_0 given by (5).

It can be seen that the suprema in the right hand side of (8) is realized for some $\hat{b'} \in S^d$ and a range value $\hat{y}_0$ (and corresponding optimal values of a and b) under the sole assumption that $X \in L^2(\Omega, R^d)$. We will use the notation $\psi^{(0)} \equiv \hat{\psi}(\hat{b'}, \hat{y}_0) \in \mathcal{C}_A$. Therefore $[X, \psi_A^{(0)}] = \sup_{\psi \in \mathcal{C}_A} [X, \psi]$.

Given the above, we will say that A *best splits* into $\hat{A}_0$ and $\hat{A}_1$, this splitting is used in the next section to define the VGS algorithm.

Remark: For simplicity, we will drop the notation $\hat{}$ used to denote the optimal values of b', y_0 and A_0.

3. 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 $\Pi_0 = \{\Omega, \emptyset\}$.
- Given Π_n, Π_{n+1} is generated as follows: Consider $A^* \in \Pi_n$ such that it satisfies

$$|[X, \psi_{A^*}^{(0)}]| \geq |[X, \psi_A^{(0)}]| \text{ for all } A \in \Pi_n. \tag{10}$$

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

The algorithm builds a tree $\mathcal{T}$ where its nodes are atoms from the partitions Π_n. The formal definition is given by: $\mathcal{T}_n \equiv \bigcup_{i=0}^{n} \Pi_i$, $\mathcal{T} \equiv \mathcal{T}_\infty$. The parent-children relationship is given by the *best split* relationship mentioned above. The vector valued functions $\psi_A^{(0)}$, with $A \in \mathcal{T}_\infty$, can be collected in 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 $\mathcal{H}_n = \{\mu_0, \ldots, \mu_n\}$ has been constructed. We then let, $\mathcal{H}_{n+1} \equiv \mathcal{H}_n \bigcup \{\mu_{n+1}\}$ with $\mu_{n+1} \equiv \psi_{A^*}^{(0)}$ where A^* is the set in (10). We also set $\mathcal{H} \equiv \bigcup_{n \geq 0} \mathcal{H}_n$.

It will be important to introduce scalar valued Haar functions. If $\psi_A = \psi = a\mathbf{1}_{A_0} + b\mathbf{1}_{A_1}$ is a vector valued Haar function, we will use the following notation for the associated scalar function

$$\psi_{A,s} = \psi_s = d_0\mathbf{1}_{A_0} + d_1\mathbf{1}_{A_1},\ \ d_i \in R$$

we also require $\int_\Omega \psi_s(w)\,dP(w) = 0$ and $\int_\Omega \psi_s^2(w)\,dP(w) = 1$. This allows us to write the scalar basis function as follows

$$\psi_s = |d_1|\,u_1\,d_1'\left(\frac{\mathbf{1}_{A_1}}{u_1} - \frac{\mathbf{1}_{A_0}}{u_0}\right), d_1' \equiv d_1/|d_1| \in \{-1, 1\}.$$

Notice $|d_1| = ||b||$, therefore

$$\psi_s^{(0)} = ||b||\,u_1\,d_1'\left(\frac{\mathbf{1}_{A_1}}{u_1} - \frac{\mathbf{1}_{A_0}}{u_0}\right). \tag{11}$$

In short, $\psi^{(0)}$ specifies $\psi_s^{(0)}$ uniquely. Clearly, we can define an orthonormal system of scalar valued Haar functions, we will denote this system $\mathcal{G} = \{u_k\}$. We will assume u_k is the element in $\mathcal{G}$ naturally associated with μ_k, where this association is given by (11).

Conditional on given sets A_0, A_1 (and hence the value u_0, is fixed) one can find the best $\hat{b}'$ as the vector in S^d that maximizes (6). A computation gives:

$$\hat{c} \equiv \frac{1}{u_1} \int_{A_0} X(w)\, dP(w) - \frac{1}{u_0} \int_{A_0} X(w)\, dP(w), \quad (12)$$

then, $\hat{b}' \equiv \frac{\hat{c}}{||\hat{c}||}$. Using this expression for $\hat{b}'$ in $\psi^{(0)}(\hat{b}', \hat{y}_0)$ we can prove the following result: given any finite index set $I \subseteq N$ we have the fundamental identity

$$\sum_{k\in I} [X, \mu_k] \mu_k[i] = \sum_{k\in I} [X[i], u_k]_1\, u_k \text{ for all } i = 1, \ldots, d. \quad (13)$$

Given a tree $\mathcal{T}_n$ with $n \geq 0$, the associated *VGS approximation* is defined as follows

$$X_{\mathcal{T}_n} \equiv \sum_{A\in \mathcal{T}_n} [X, \psi_A^{(0)}]\, \psi_A^{(0)}. \quad (14)$$

Using (13) it can be shown that for any $n \geq 0$ and $w \in A$:

$$X_{\mathcal{T}_n}(w) = \frac{1}{P(A)} \int_A X(w) dP(w), \quad \text{for all } A \in \Pi_n. \quad (15)$$

Therefore the sequence $X_{\mathcal{T}_n}$ is a martingale with respect to the sigma algebra $\mathcal{F}_n \equiv \sigma(\Pi_n)$. Moreover, it can be seen that $\lim_{n\to\infty} X_{\mathcal{T}_n}(w) = X(w)$ for almost all $w \in \Omega$. In fact, if X takes only a finite number of distinct values there exists N such that $X_{\mathcal{T}_N}(w) = X(w)$ for almost all $w \in \Omega$.

4. TRANSFORM COMPRESSION: SCALAR AND VECTOR APPROXIMATIONS

In practice, and as a first step, we will run the VGS algorithm on an input vector in order to obtain a "full" tree $\mathcal{T}_N$ so that $||X - X_{\mathcal{T}_N}|| \approx 0$. The second 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. We consider two such approaches next.

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 $L^2(\Omega, R^d)$. 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 $L^2(\Omega, R)$) . On the other hand, for a fixed scalar error level ϵ_s, it is possible to generate scalar approximations $X[i]_n$, where now $n = n(i, \epsilon_s)$, for each component $X[i]$ in such a way that $||X[i] - X[i]_n|| \leq \epsilon_s$ 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 in more detail below.

The relation (13) is a basic result and shows 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 (for compression purposes) when one considers the optimized expansions as we explain next.

Let $h : N \to N$ be a re-ordering function for $\mathcal{H}$ in such a way that $|[X, \mu_{h(0)}]| \geq |[X, \mu_{h(1)}]| \geq \ldots$. We then define the n-term VGS optimized vector approximation by

$$X_n \equiv \sum_{k=0}^{n-1} [X, \mu_{h(k)}]\, \mu_{h(k)}. \quad (16)$$

In practice, the integer n is chosen to satisfy some error criteria, say a vector error level ϵ_v is given so we can find $n = n(\epsilon_v)$ so that $||X - X_n|| \leq \epsilon_v$.

One can define the same notions for the orthonormal system $\mathcal{G}$, let $g_i : N \to N$ be a 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 scalar approximations by

$$X[i]_n = \sum_{k=0}^{n-1} [X[i], u_{g_i(k)}]_1\, u_{g_i(k)}. \quad (17)$$

Given an scalar error level ϵ_s we can find integers n_i such that $||X[i] - X[i]_{n_i}|| \leq \epsilon_s$ for all $i = 1, \ldots, d$.

To sum up, there are two possible optimized approximations, the optimized VGS approximation given by (16) (which we call the *vector approximation*) and the d optimized scalar VGS approximations given by (17) (which we call the *scalar approximations*). These two approximations are obtained by pruning the tree and keeping only the *active nodes* for further processing. It should be clear that we call the active nodes are the tree nodes associated to the inner products appearing in the optimized approximations. In each case, the pruning 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.
Once the pruning has been completed, we need to store the relevant information associated to each node. 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 the vector case one needs to store the following information at the active nodes: numbers of the form $[X, \psi_A^{(0)}]$ and a corresponding vector b'_A. In the scalar case one needs to store some (or all) of the following numbers: $[X[i], \psi_{A,s}^{(0)}]_1$, $i = 1, \ldots, d$.

There also exists another related approximation used in this work, it is called *leaves average approximation*, this approach uses the information on the tree leaves after the tree has been pruned and it is described in Section 6.3.

5. DATA STRUCTURES AND BIT COUNTING

The decoder of the VGS approximations will use the following three data structures: partition map, significance map and quantization map. Roughly speaking, the partition map encodes the partition associated to the tree *after* it has been pruned; the significance map relates the Haar functions associated to active nodes with the corresponding partition atoms and stores the children-parent information associated to active nodes. The quantization map stores the quantized information required for reconstruction at active nodes. When reporting numerical results we will actually indicate with a single quantity the cost of the significance map *plus* the cost of the quantization map. Moreover, whenever reporting bit costs for encoding the partition map we will use two different methods: theoretical estimated costs (by means of entropy encoding) and the cost resulting from Lempel-Ziv lossless encoding.

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

Definition: Consider $n \equiv |\Pi(\Omega)|$, where $\Pi(\Omega)$ is a finite partition of Ω, a function $\mathcal{M}_\Pi$: $\Omega \to N$ is called a *Partition Map* if for each $A_k \in \Pi(\Omega)$, $k = 1, \ldots, n$, it satisfies: $\mathcal{M}_\Pi(w) = v_k \quad \forall\, w \in A_k$, if $k \neq j \;\Rightarrow\; v_k \neq v_j$. The integers v_k will be called *symbols*.

We describe how the Partition Map is created by means of an example. Figure 1 displays a full tree obtained after three iterations, the partition associated to the full tree is shown in Figure 2 a). Assuming the nodes $\{1, 3, 6\}$ are

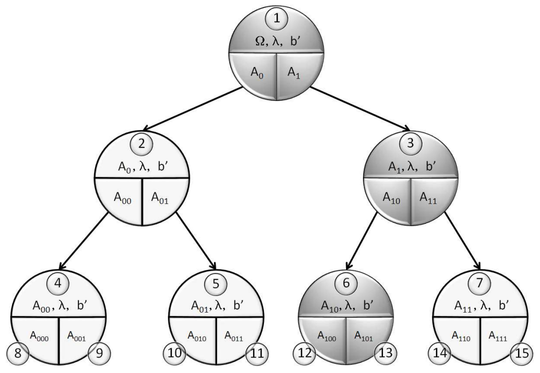

Figure 1. Full tree with active nodes marked.

the only active nodes, the resulting partition is shown in Figure 2 b).

Notice that node 2 is not active and hence the symbol associated to node 1 is not further changed in this case (unless a descendant of node 2 were actually active.)

Remark: The partition map shares the same domain as the input images, and the maximum number of atoms is equal to the number of pixels. In general, an upper bound for the number of bits needed to store the partition map is $\log_2 |\Omega|$.

The entropy encoding is straightforward, the symbols are the integer values assigned to atoms in $\Pi(\Omega)$. The associated entropy is denoted by $H_{\mathcal{M}_\Pi}$ and if $N_s \equiv |\Omega|$ then the theoretical cost associated to $\mathcal{M}_\Pi$ is $C_{\mathcal{M}_\Pi} = H_{\mathcal{M}_\Pi} \times N_s$.

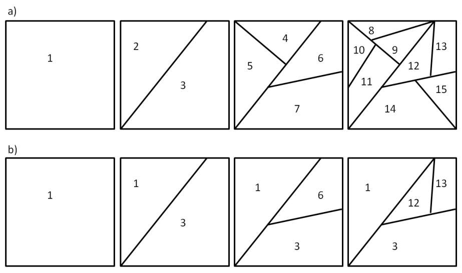

Figure 2. a) Partition using the full tree, b) Partition using the compressed tree.

5.2. Significance Map ($\mathcal{M}_S$)

The tree structure required to reconstruct the input vector is encoded by the data structure which we will call the *Significance Map* (SM). Notice that the significance map also needs to include links from nodes to the partition encoded by the partition map.

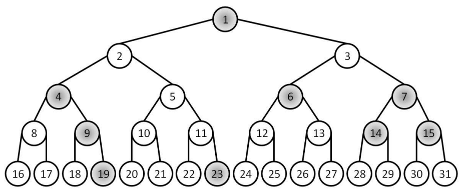

Figure 3. Compressed tree.

As we can see in Figure 3, if a node is active we *do not require the ancestors to be included.* 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. The resulting algorithm is rather complex though and it is best described by means of an example. We use three different types of symbols to encode the tree, they are used to create a string which will be called the *significant string* and denoted with $\mathcal{S}$. The symbols are: Q: Active node, V: Link to the partition and D: Dummy node.

We start visiting tree nodes using a preorder traversal method. Node 2 is not active and its right children is not active we then label this node with a V. Node 3 has both children active we then use the symbol D. The algorithm continues until the following string is constructed $\mathcal{S} = \{Q2, V2, Q2, V, Q2, V, Q2VV, Q2VV, D2,$ $Q2VV, Q2, Q2VV, Q2VV\}$. Figure 4 shows the decoded tree, for the purpose of reconstruction it is equivalent to the original tree. The number of symbols proposed is three, but if we associate the number of children to the symbol we may check that the sequence of symbols "$\{Q2VV\}$" has the highest probability, we could then introduce another new symbol to code $Q2VV$. This is the analogous of the zero tree symbol introduced in [1].

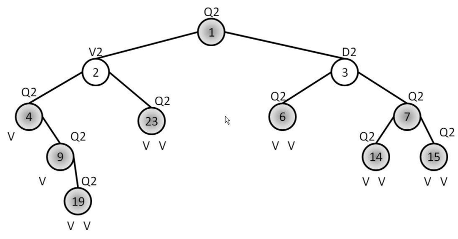

Figure 4. Equivalent decoded tree.

Definition A function $\mathcal{M}_S$: $\mathcal{S} \rightarrow Z$ is called a *Significance Map*. For a given $k \in Z$ define: $\mathcal{S}_k = \{s \in \mathcal{S} : \mathcal{M}_S(s) = k\}$. Also define the symbol set $\mathcal{J}_S$ by: $\mathcal{J}_S = \{\mathcal{S}_k \subset \mathcal{S} : \mathcal{S}_k \neq \emptyset\}$.

Using entropy encoding we find that $H_{\mathcal{M}_S} = -\sum_{\mathcal{S}_k \in \mathcal{J}_S} p_k \log_2 p_k$ where $p_k = \frac{|\mathcal{S}_k|}{|\mathcal{S}|}$. The theoretical cost associated to the significance map is given by $C_{\mathcal{M}_S} = H_{\mathcal{M}_S} \times |\mathcal{S}|$. 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.

5.3. Quantization Map ($\mathcal{M}_Q$)

In order to use entropy encoding we need to make use of a quantization method. The two techniques can be combined and performed simultaneously as in the case of the arithmetic coding, see [1], [6].

Let us use λ_k to denote, for the moment, the values of inner products (scalar or vector) and let $\mathcal{P}$ denote a queue containing the significant inner products. We have verified that the best quantization technique for our algorithm is the uniform quantization defined as follows: $\mathcal{V}(\lambda_k) = \left\lfloor \frac{\lambda_k}{c} \right\rfloor \times c$ and $c > 0$. We also set $Q \equiv \{\mathcal{V}(\lambda_i) : \lambda_i \in \mathcal{P}\}$, we can then define the quantization map as follows.

Definition: A function $\mathcal{M}_Q$: $Q \rightarrow Z$ is called a *Quantization Map*. Also, define the set of al values from Q which equal k, namely: $Q_k = \{q \in Q : \mathcal{M}_Q(q) = k$ and $k \in Z\}$ and then the symbol set is given by $\mathcal{J}_Q = \{Q_k \subset Q : Q_k \neq \emptyset\}$. This allows us to compute the We entropy encoding $H_{\mathcal{M}_Q}$ to find the average bit per symbol. Then the theoretical total cost associated with the quantization map can be computed as follows $C_{\mathcal{M}_Q} = H_{\mathcal{M}_Q} \times Q$.

As an example, let $\lambda_k = [X[i], \psi_{A,s}]_1$ denote the largest inner products kept after pruning a full tree by means of the scalar approximation. Figure 5 shows the values of λ_k sorted by $|\lambda_k|$ (values taken from a video sequence).

6. ENCODING FOR SCALAR, VECTOR AND LEAVES AVERAGES APPROXIMATIONS

6.1. Scalar approximation

Here we describe the bit cost associated with the scalar approximation. The information needed for the reconstruction includes the scalar inner products $[X, \psi_s]_1$, the partition and the tree. A node is considered active if at least one scalar product λ_i is required at the node.

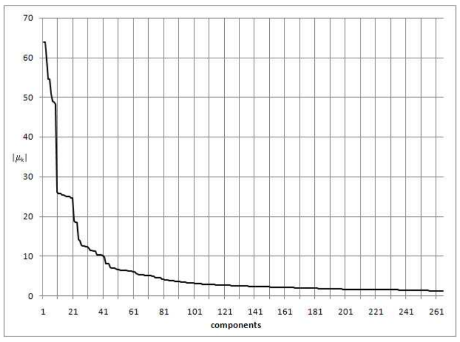

Figure 5. Scalar inner products distribution.

Indices information: The indexing information can be encoded using three different approaches. The first approach uses d bits to encode whether an 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 6 shows

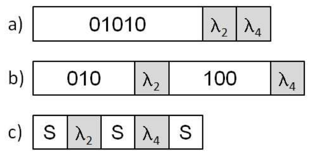

Figure 6. a) Binary encode, b) Indexing encode, c) Special character.

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 $\{\lambda_2, \lambda_4\}$ need to be stored. Therefore, for a given node n the associated cost of each model is calculated as follows: a) $C_{I_n} = d + k\ H_{\mathcal{M}_Q}$, b) $C_{I_n} = k\ \log_2 d + k\ H_{\mathcal{M}_Q}$ c) Consider that the special null character has $H_{\mathcal{M}_Q}$ bits then $C_{I_n} = d\ H_{\mathcal{M}_Q}$. 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 of inner products being used at node n. It is possible to evaluate a priori which method is the best for each node and then add two bits to the header of the node so the decoder can use the correct method. The total indexing cost is $C_I = \sum_n C_{I_n}$. The total cost C_T is given by $C_T = C_{\mathcal{M}_\Pi} + C_{\mathcal{M}_S} + C_I$. Where $C_{\mathcal{M}_\Pi}$ is the cost associated with the partition, $C_{\mathcal{M}_S}$ is the cost associated with encoding the tree and C_I is the indexing cost. Notice that the cost associated with the quantized coefficients $C_{\mathcal{M}_Q}$ (see section 5.3), is included in C_I. As indicated, when reporting actual values we will report $C_{\mathcal{M}_\Pi}$ and $C_{\mathcal{M}_S} + C_I$.

6.2. Vector Approximation

At a given active node A, the vector approximation needs to store $[\psi_A^{(0)}, X]$ and $\hat{b}'_A$ given by (12). This last vector can be encoded efficiently as we describe next. Notice that $\frac{1}{P(A_k)} \int_{A_k} X[i](w)\ dP(w) = \mathbf{E}_{A_k}(X[i])$, which is the expected value of $X[i]$ relative to the atom A_k, $k = 0, 1$.

Therefore, the value of the best $\hat{b}'_i$ is given by the normalized difference of two expected values $\mathbf{E}_{A_1}(X[i]) - \mathbf{E}_{A_0}(X[i])$. Define now $\Delta_i = \mathbf{E}_{A_1}(X[i]) - \mathbf{E}_{A_0}(X[i])$, so $\hat{b}'[i] = \Delta[i]/||\Delta||$.

Quantization Map: The quantization technique amounts to taking the integer part of the difference of the expected values defined above, $\mathcal{V}(\Delta[i]) = \lfloor \Delta[i] + 0.5 \rfloor$ Figure 7 shows and example of the relative frequency of the quantized differences $\Delta[i]$, a set of 9 images with a PSNR= 40 was used as the input vector. As we have done previously,

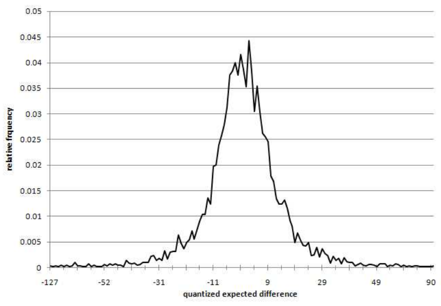

Figure 7. Relative frequency of the quantized difference of the expected values.

if $H_{\mathcal{M}_Q}$ denotes the average number of bits per symbol, then the theoretical total cost associated with the quantization map is given by: $C_{\mathcal{M}_Q} = H_{\mathcal{M}_Q} \times |Q|$, where Q was defined in Section 5.3. Finally, the total cost C_T is $C_T = C_{\mathcal{M}_\Pi} + C_{\mathcal{M}_S} + C_{\mathcal{M}_Q}$, where $C_{\mathcal{M}_\Pi}$ is the cost associated with the partition, $C_{\mathcal{M}_S}$ is the cost associated with the tree and $C_{\mathcal{M}_Q}$ is the cost associated with the quantized coefficients and $\hat{b}'$.

6.3. Leaves Average Approximation

Given a finite partition Π, resulting from an application of the VGS algorithm, we compute the integer part of the average of each input image over each atom $A_j \in \Pi$, $\lambda_{ij} = \left\lfloor \frac{1}{|A_j|} \sum_{w \in A_j} X[i](w) \right\rfloor$, so $\lambda_{ij} \in Z$. Define $\Lambda \equiv \{\lambda_{ij} \text{ for all } i = 1, \ldots, d \text{ and } j = 1, \ldots, n\}$ where $n = |\Pi|$ and d is the number of input images, then $|\Lambda| = n \times d$.

The *leaves average approximation* X_Π is then defined by $X_\Pi(w) = (\lambda_{1j}, \lambda_{2j}, \ldots, \lambda_{dj}) \quad \forall\, w \in A_j$. Therefore, if C_Λ is the cost associated to encoding the set of integers Λ, the total cost associated to this approximation is $C_T = C_\Lambda + C_{\mathcal{M}_\Pi}$.

7. VIDEO COMPRESSION RESULTS

This section illustrates the VGS algorithm applied to standard video sequences considered in the literature; more information about the video sequences used can be obtained from [7]. Table 1 indicates the videos considered in this paper.

Table 1. Video sequences used in the paper.

Sequence	Format	Frames	Resolution
Akiyo	QCIF	300	176×144
Foreman	QCIF	300	176×144
Flowers and Garden	CIF	250	352×288
Foreman	CIF	300	352×288

We present results for the following two methods: scalar approximation and average leaves, these methods will be denoted Haar VGS (HVGS) and Average VGS (AVGS) respectively. The AVGS method is considered with or without lossless compression. In general, we only provide results in order to illustrate several characteristics of the VGS algorithm and only briefly comment on how it compares to other approaches. Comparisons with MPEG can be found in [8]. Figure 8 displays a comparison between the cost of the PM (Partition Map) and the QM (Quantization Map) plus the SM (Significance Map) for the *Foreman* video. The method used is HVGS; the graph corresponds to a specific average distortion.

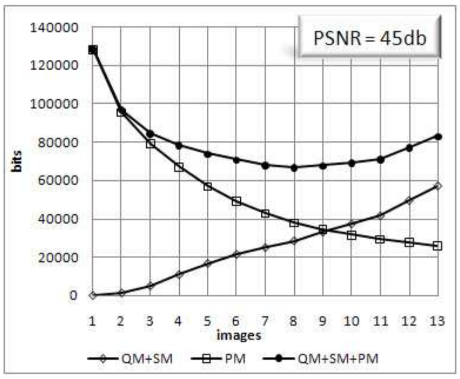

Figure 8. Foreman Avg.PSNR=45db - bits vs. d.

Figure 9 displays a rate-distortion graph for the video *Flowers and Garden* using HVGS .

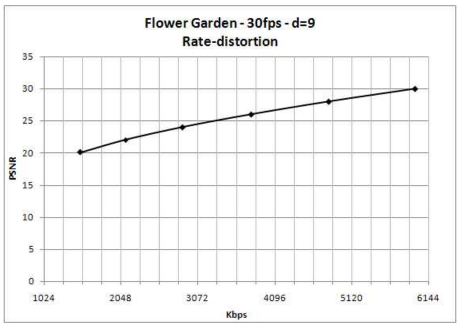

Figure 9. HVGS Flowers & Garden Rate-distortion graph.

Figures 10 and 11 illustrate several aspects of our methods applied to *Akiyo*. Figure 10 plots the bit rate as a function of increasing values of d for a given target average PSNR of 35 db; both methods, HVGS and AVGS are shown in the same graph.

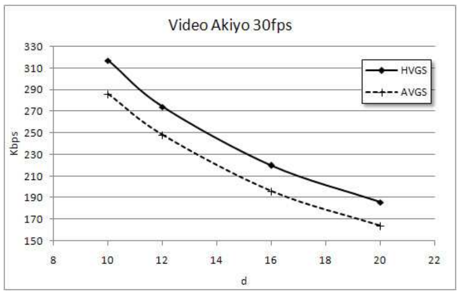

Figure 10. Akiyo Rate vs. d - Avg. PSNR=35db.

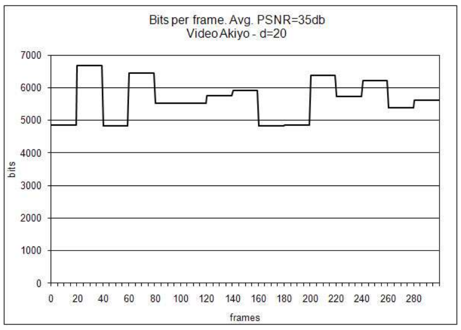

Figure 11. Akiyo Avg.PSNR=35db d=20 bit-rate=186.51Kbps.

Figure 11 displays the total number of bits averaged over the numbers of frames as the VGS algorithm iterates over the video frames. The value of $d = 20$ was used and the bit rate was 186.51 Kbps. The Figure illustrates how the number of bits oscillate, as a function of the frames, for a given bit rate target. The performance results described so far for the VGS algorithm are not competitive with state of the art video compression algorithms that make use of motion compensation and inter-frame prediction. It should be emphasized that we have implemented a rather naive version of the possibilities offered by VGS, in particular, there exists the possibility to perform more thorough optimizations leading to ternary trees ([8]). Notice also that we have used the same value of d for the whole video sequence, choosing dynamic values for d will also make the VGS more competitive. In order to enhance the performance of VGS we describe next several numerical experiments in which we partition the frames in tiles. For simplicity, we constraint the tiles to be fixed for the whole video sequence. Moreover, the tiles are of same size and shape (rectangular). Figure 12 shows a comparison between the bit-rate vs. d for the *Foreman* video sequence, with a target of $28db$ and using 3×3 tiles. The same figure also presents the comparison between the AVGS method with and without applying the Lempel Ziv algorithm to the stored data (this method is labeled AVGS-LZIV).

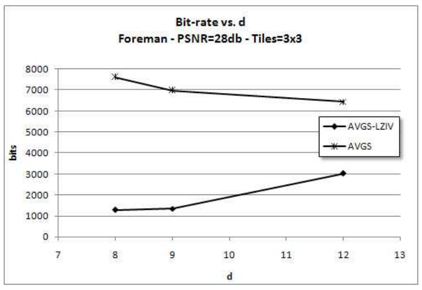

Figure 12. Foreman CIF Avg.PSNR=28db bit-rate vs. d Tiling=3x3.

Again, for the *Foreman* video sequence, Table 2 shows the bit-rate values (in Kbs/sec.) for a given target of 28db and different values of d and numbers of tiles. Values for both algorithms, AVGS and AVGS-LZIV, are displayed.

Table 2. Foreman CIF, bit-rate (kbits/sec.) values for different values of d for AVGS-LZIV and AVGS algorithms. Target PSNR $= 28$ db.

d	Tiles	AVGS-LZIV	AVGS
8	3x3	1279.20	7607.84
9	1×1	1452.30	6852.08
	1×3	1329.92	6917.11
	2×3	1453.30	6956.36
	3×1	1353.50	6931.84
	3×3	1341.51	6983.98
	4×4	1271.36	7049.05
	6×6	1427.99	6918.42
12	3x3	3028.24	6443.83

Table 3, for the video sequence *Flowers and Garden*, shows a bit-rate comparison between AVGS and AVGS-LZIV using different number of tiles for a fixed distortion target of PSNR $= 27db$.

Table 3. Flowers CIF, bit-rate (kbits/sec.) values for AVGS-LZIV and AVGS algorithms, PSNR = 27db.

Tiles	AVGS-LZIV	AVGS
1×1	5005.67	7093.59
2×2	5824.99	8721.78
4×4	6285.36	9390.17
6×6	6592.69	9753.52
1×3	6596.06	9172.63

Table 4, for the video sequence *Akiyo*, shows the bit-rate for different number of tiles using AVGS-LZIV for a fixed distortion target of PSNR $= 35$db.

Table 4. Akiyo QCIF, bit-rate (kbits/sec.) Tiles from 1×1 to 4×4, PSNR=35db.

	1	2	3	4
1	108.8	76.3	106.2	99.0
2	76.1	81.2	107.3	100.7
3	87.0	94.6	124.7	122.1
4	87.3	96.6	123.7	129.9

In order to provide support to the idea that different tiles require an specific value for d, we describe the variation of the bit-rate across different tiles for the video *Flowers and Garden*. In the first case, CIF, AVGS, d=9, tiles=4×4 and distortion PSNR=28db, the average bit-rate per tile (the average cost of a tile in one frame, because it is the same for the d images) is shown in Figure 13. It is possible to see the variation of the cost within each consecutive group of 16 tiles that conform each frame.

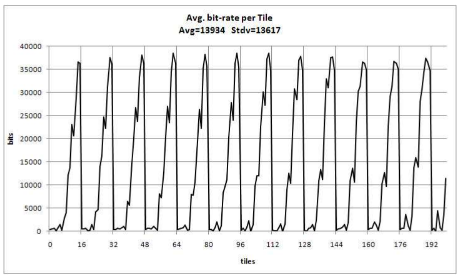

Figure 13. Bit-rate per tile - Flowers CIF Avg.PSNR = 28db, $d = 9$, 4×4 tiles. Average = 13934 Std.dev. = 13617.

Finally, Figure 14 shows the original Akiyo video sequence in QCIF format 176×144 and Figure 15 shows a detail of one frame and its reconstruction using AVGS with a distortion target of PSNR = 38db and $d = 9$.

Figure 14. Video Sample Akiyo QCIF.

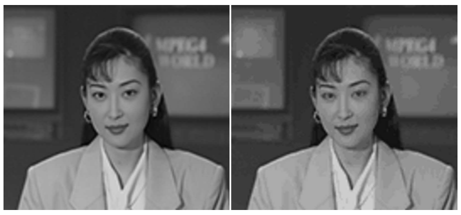

Figure 15. Akiyo reconstruction detail using AVGS - PSNR = 38db - $d = 9$. Left: original, right: reconstruction.

8. CONCLUSION

We have described the construction of an adapted orthonormal basis, the construction provides a simultaneous approximation to a given collection of video frames. The adaptivity allows fast decay of the inner products between the given images and the basis elements. There is the extra cost of storing the basis elements; this cost is ameliorated by imposing a tree structure to the construction. Numerical results illustrate the trade off between speed of convergence and the storage costs. We also provide results indicating the performance of the proposed algorithm on a set of standard video sequences. Competitive performance can be obtained at the expense of introducing more sophistication in the proposed technique, in particular we show how the results are improved by introducing tiles and lossless compression of the data structures.

9. ACKNOWLEDGMENTS

The research of S.E. Ferrando and A.J Bernal is supported in part by an NSERC grant.

10. REFERENCES

[1] J.M. Shapiro, “Embedded image coding using zerotrees of wavelet coefficients,” *IEEE Transactions on Signal Processing*, vol. 41, pp. 3445–3462, 1993.

[2] A. Averbuch D. Alani and S. Dekel, “Image coding with geometric wavelets,” *IEEE Transactions on Image Processing*, vol. 16, pp. 69–77, 2007.

[3] M. Vetterli H. Radha and R. Leonardi, “Image compression using binary space partitioning trees,” *IEEE Transactions on Image Processing*, vol. 5, pp. 1610–1624, 1996.

[4] S.E. Ferrando P.J. Catuogno and A.L. Gonzalez, “Adaptive martingale approximations,” *Journal of Fourier Analysis and Applications. In Press*, 2008.

[5] Elliott Lieb and Michael Loss, *Analysis*, American Mathematical Society, 2001.

[6] A. Said and W. pearlman, “An image multiresolution representation for lossless and lossy compression,” *IEEE Transactions on Image Processing*, vol. 5, pp. 1303–1310, 1996.

[7] Abhijeet Golwelkar and John Woods, “Motion-compensated temporal filtering and motion vector coding using biorthogonal filters,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, pp. 417–428, April 2007.

[8] A.J. Bernal, “Simultaneous approximations of images. applications to image and video compression,” *MASc Thesis*, vol. Ryerson University, 2007.