AN EFFICIENT ALGORITHM TO FIND A JOINTLY OPTIMAL TIME-FREQUENCY SEGMENTATION USING TIME-VARYING FILTER BANKS

Cormac Herley [1], *Zixiang Xiong* [2] *and Kannan Ramchandran* [2] *and Michael T. Orchard* [2]

[1] Hewlett-Packard Laboratories, 1501 Page Mill Road, Palo Alto, CA 94304
[2] University of Illinois at Urbana-Champaign, 405 N. Mathews Ave., Urbana, IL 61801

ABSTRACT

We examine the question of how to choose a time-varying filter bank representation for a signal which is optimal with respect to an additive cost function. We present in detail an efficient algorithm for the Haar filter set which finds the optimal basis, given the constraint that the time and frequency segmentations are binary. Extension to multiple dimensions is simple, and use of arbitrary filter sets is also possible. We verify that the algorithm indeed produces a lower cost respresentation than any of the wavelet packet respresentations for compression of images using a simple Rate-Distotion cost.

1 INTRODUCTION

We address the problem of designing a linear transformation to make the task of compressing a signal easier. That is, given a fixed signal $\mathbf{x}$, and a fixed cost function $Cost(.)$, we seek an orthogonal transformation $\mathbf{A}$, such that $Cost(\mathbf{A}\cdot\mathbf{x})$ is small. An unconstrained search over all possible unitary $\mathbf{A}$ is not feasable, so we will restrict our attention to some library of transformations, with the properties that (a) we can search for the least-cost member of the library efficiently, and (b) we can transmit a description of any member for little additional cost.

The libraries of transformation that we use will be based on orthogonal filter bank tree structures. Paraunitary filter banks carry out an orthogonal transformation of the signal, and trees constructed from them correspond to bases where the trade-off between time and frequency resolution for the basis functions depends on the structure of the tree. This approach to viewing tree structured bases as tilings of the time-frequency plane is explored in [2].

An example of a tree structured basis built from a two channel filter bank is shown in stage 1 of Figure 1, which is a uniform tree grown to a depth of three for a signal of length 8. A tree of this kind is the starting point for the algorithms we will consider. An algorithm was used in [1, 5] which pruned this tree based on comparisons between the costs of pairs of branches at one level and the cost of their parent branch at the previous level. This gave the best wavelet packet, or best frequency selective tree that could be found for that signal with respect to the chosen cost function. A "double-tree" generalization of this in [2] aimed at overcoming the restriction that the trees remain invariant for the duration of the signal, and searched a larger library of bases. In this work we find it possible to improve on that algorithm, and present a scheme for searching a larger library of bases for little additional complexity. We illustrate in detail the simplest form of the algorithm, which involves it's use for block transforms. A scheme developed by Villemoes [6] to find the best Walsh basis for a continuous-time signal appears to be closely related.

2 THE BLOCK TIME-FREQUENCY ALGORITHM

We will illustrate the algorithm using the Haar filters, where $H_0(z) = (1 + z^{-1})/\sqrt{2}$ and $H_1(z) = (1 - z^{-1})/\sqrt{2}$. Filters like this, where the filter length is equal to the number of channels, are called block transforms because the low and high outputs are related to the input by a block operation

$$\begin{pmatrix} b_i \\ b_{i+N/2} \end{pmatrix} = \frac{1}{\sqrt{2}} \cdot \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \cdot \begin{pmatrix} a_{2i} \\ a_{2i+1} \end{pmatrix}.$$

The operator $\mathbf{A} = 1/\sqrt{2}[1\ 1; 1\ -1]$ is often thought of as mapping time domain points to frequency domain points. An advantage of block transforms is that no special treatment is necessary at the boundaries, and the decomposition can be performed on finite length signals without complications. Thus, in the case of the tree for an eight point signal as shown in stage 1 of Figure 1, $(b_0\ \ b_4)$ is the transformed version of $(a_0\ \ a_1)$, $(b_1\ \ b_5)$ is the transformed version of $(a_2\ \ a_3)$ and so on. The coefficients b_i are transformed in pairs to give the c_i; these are transformed in pairs to give the d_i, the deepest level of the tree.

In order to choose a basis, we must select eight coefficients from the total of 32 represented by the a_i, b_i, c_i and d_i (in general for a signal of length 2^j we must select 2^j from $(j+1)2^j$). We cannot just select the 8 coefficients with the lowest cost, since to form a basis they must be the coefficients of 8 mutually orthogonal basis functions. In the case of the Haar filters two basis functions are orthogonal if their coefficients do not share a common ancestor. For example, a_0 and b_4 could not appear together in the same basis, since (b_0, b_4) is found by transforming (a_0, a_1), and hence a_0 is an ancestor of b_4.

Since we have a two-tap filter, the points at the second level (the b_i) depend on only two points in the first level (the a_i), so we can compare the a_i and the b_i in pairs, without any consideration of the neighbouring pairs. The same is true of the other levels of the tree: each c_i depends on only two b_i, and so on. In general we can compare points from level k and level $k+1$ in pairs. We illustrate the situation in Figure 1 stage 1. For level 1 *vs.* level 2 we write the cost of the winners

$$e_i = \min(Cost(a_{2i}, a_{2i+1}), Cost(b_i, b_{i+4})). \tag{1}$$

Similarly for level 2 *vs.* level 3 ($p = 2(i/2 - \lfloor i/2 \rfloor)$)

$$f_i = \min(Cost(b_{2i}, b_{2i+1}), Cost(c_{2i-p}, c_{2i+2-p})), \tag{2}$$

and finally

$$g_i = \min(Cost(c_{2i}, c_{2i+1}), Cost(d_{2i}, d_{2i+1})). \tag{3}$$

We have written the winners of the decisions of stage 1, as a new cost tree as shown in Figure 1 stage 2, which has one less level than the tree we started with. The new tree is populated with the costs of the winning pairs from the first stage. Since (e_{2i}, e_{2i+1}) and (f_i, f_{i+2}) represent the costs of alternative representations for sets of *four* points in the original signal, we can compare the e_i and the f_i in pairs just as we did the a_i and the b_i in (1). Similarly the f_i and the g_i can be compared in pairs much as in (2). This reduces the size of the problem by another level. Thus, after two stages we have reduced it to a two point problem and the winning cost will be

$$C_{min} = \min(Cost(h_0, h_1), Cost(i_0, i_1)). \qquad (4)$$

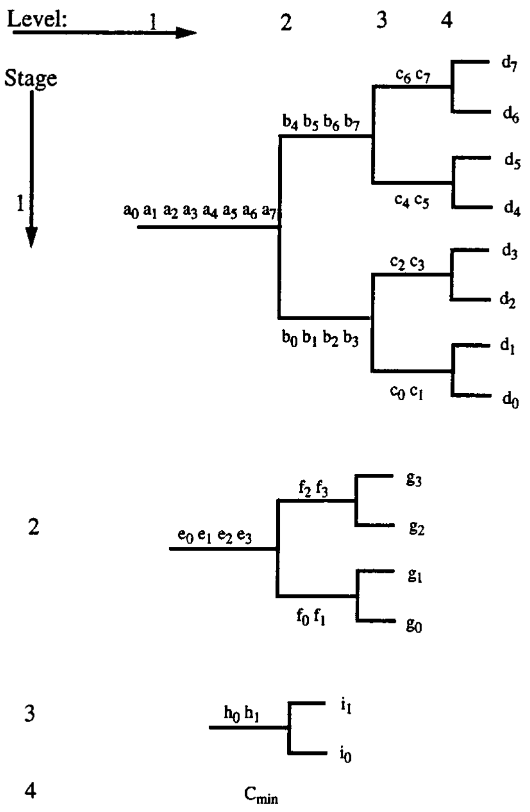

Figure 1: The Block Time-frequency algorithm for an eight point signal and a two-point block transform. The original tree (stage 1) has 4 levels; we compare the costs in pairs between adjacent levels, and store the cost of the winner in stage 2. Stage 2 has a cost tree of three levels. Again, comparing costs in pairs gives stage 3 which is a trivial two-point problem.

The heart of the algorithm is then for each block of points in a tree, compare the cost of the block to the cost of the transformed block, and keep the cost of the winner. This is done for each level in the tree, and for each stage in the algorithm. At stage k the first level of the cost tree contains the best costs if the tree were constrained to have depth less than or equal to stage k. If the original signal is of length 2^j, and the problem is reduced by 2 at each stage, we will have reduced it to a one-point problem after j stages. This last number is the overall cost. At any stage we thus deal with a tree over a $2^{j-stage+1}$ point signal, so we will have $(j - stage + 1)$ levels; at each level there will be $2^{j-stage}$ blocks. Our forward algorithm is thus (assuming that the stage 1 cost has been calculated):

```
for stage = 1:j
   for level = 1:(j-stage+1)
      for block = 1:2^{j-stage}
         time = Get_cost(stage,level,block);
         freq = Get_tr_cost(stage,level+1,block);
         if time <= freq
            Put_cost(stage+1,level,block) = time;
            Put_win(stage+1,level,block) = 0;
         else
            Put_cost(stage+1,level,block) = freq;
            Put_win(stage+1,level,block) = 1;
         end
      end
   end
end
```

In the pseudocode description of the algorithm we leave unspecified the data structure to store the costs at the various stages. We merely assume that the function Get_cost(stage,level,block) returns the cost of the block at the appropriate stage and level and that the function Get_tr_cost(stage,level+1,block) returns the cost of the transformed block (which is one level deeper). Similarly, Put_cost(stage+1,level,block) stores the winning cost to be used at the next stage, and we record whether the winning decision was "time" or "frequency" with Put_win(stage+1,level,block) = 0 or 1. Extension to longer block transforms is trivial.

The single number that is the output of the forward algorithm, Cost(j+1,1,1) is the cost of the winning basis. To use this we need to know in addition which of the coefficients should be sent to the decoder, and a description of the time-varying tree. The decoder should then receive the quantized coefficients, put them into the inverse tree and reconstruct the signal at the advertised cost. To determine the coefficients to send we employ an algorithm that works backwards through the decisions made by the forward algorithm. Essentially, we retrace the winning decisions and eliminate all of the coefficients that contributed only to the losing cost for each decision.

Observe from the forward algorithm, that the cost at stage k, level l depends on a block from stage $k-1$ at level l or $l+1$. Once we know that the decision that generated the cost at stage k, level l was "time" (resp. "frequency") we can eliminate a block of cost points from stage $k-1$ at level $l+1$ (resp level l). This eliminated block implies that we can eliminate 2 blocks at stage $k-2$, and 2^{k-1} blocks from the original stage 1 cost tree. Our reverse algorithm is to step through the stages in reverse order; at each level, and for each block we retrieve the decision that led to that cost. This allows us to eliminate points in the original stage 1 cost tree that did not contribute to that cost. When eliminating points in the cost tree, we also eliminate points in the winner tree, so that we don't waste time tracing the decisions that led to points that were ultimately eliminated. In summary the reverse algorithm is:

```
for stage = j:-1:1
   for level = 1:(j-stage+1)
      for block = 1:2^{j-stage}
         if Get_win(stage,level,block) == 0
            Elim_coeff(1,level+stage, block);
            Elim_win(stage-1,level+1,block);
         elseif Get_win(stage,level,block) == 1
            Elim_coeff(1,level, block);
            Elim_win(stage-1,level,block);
         end
      end
   end
end
```

We again illustrate the idea for the same eight point signal as before. Consider (4) which is the last decision of the forward algorithm, and suppose that the outcome was "frequency," which implies that $i_0 + i_1 < h_0 + h_1$, the final cost is $i_0 + i_1$ and we can eliminate all of the e_i from stage 2, and all of the a_i from stage 1, since these do not contribute to the winning cost. Even though we do not yet know the coefficients of the best basis, we can already eliminate these, based on the last decision. Equally, if we wish to construct a time-frequency tiling we can now draw a horizontal line to represent the single frequency split; this has been done in Figure 2 (a), where the surviving trees are depicted in the upper and lower branches.

At the next stage we have two decisions, the results of the decisions between the f_i and the g_i (those between the e_i and the f_i are no longer of interest). Suppose that the decision for the lower frequency branch is $f_0 + f_1 < g_0 + g_1$; this implies that we can eliminate $d_0, \cdots d_3$ since these contribute only to the losing cost. Similarly if the decision for the upper branch was $f_2 + f_3 > g_2 + g_3$ we can elimiate $b_4 \cdots b_7$. The eliminations have been carried out in Figure 2 (b) where the lower frequency branch has been split in time and the upper in frequency. Finally, when we have retraced as far as stage 1, the remaining decisions affect only two points, and the winning costs are shown in Figure 2 (c). An example of the the tilings generated by the wavelet expansion, the best single-tree expansion, and the expansion produced by the new algorithm are shown in Figure 4. As cost function we used mean squared error when all coefficients were quantized using a uniform quantizer of step size 2. The darkness of a tile represents the magnitude of the cost of the corresponding basis function.

2.1 Extent and complexity

Since the goal of this work was to improve on the algorithms in [1, 5] and [2] it is natural to ask how the algorithm compares, both in terms of the number of bases that it searches, and in terms of complexity of the search operation. In Table 1 recursions are given for the number of bases searched by each of the mentioned algorithms; in each of the cases we assume a signal that has length equal to a power of 2, and use of the Haar filters. The number of bases grows extremely quickly as function of the length of the signal. For example for a signal of length 64 points, using the Haar filters, there are $2.1e + 11$ singletree bases, $9.8e + 14$ double-tree and $6.4e + 16$ Block Time-frequency bases. The order of the computational complexity for a signal of length N and tree of depth d is $O(Nd)$ for the single-tree algorithm, $O(Nd^2)$ for the double-tree and $O(Nd)$ for the Block Time-frequency algorithm, and $O(N2^d)$ for the non-block version.

A further point of interest is the nature of the splittings generated by the algorithms. As mentioned earlier the time-frequency segmentations of [1, 5] were stationary: we get one split for the whole signal. Those of [2] are illustrated in Figure 3 (a), where we perform arbitrary binary frequency splits over various time segments. Observe that there is some asymmetry about the splits generated, in that there are many frequency splitting trees grown over time segments, but the converse is not true. This is rectified by the Block Time-frequency algorithm which gives the time-frequency segmentations shown in Figure 3 (b).

2.2 Non-block transforms

The Block Time-frequency algorithm represents an efficient way of carrying out the time-frequency splits illustrated in Figure 3 (b), for the case of block transforms. The key to making the algorithm simple was the fact that no special treatment is necessary at the boundaries in block transforms, and the tree can be varied at will. When non-block transforms are used special consideration is necessary if we wish to change the structure of the tree.

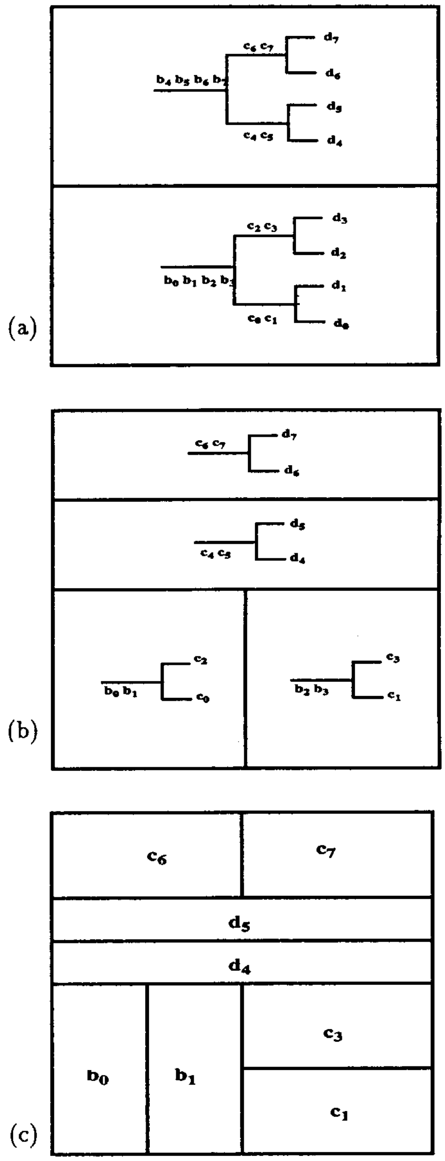

Figure 2: The backtracking algorithm to determine the coefficients of the least cost basis for an 8 point signal. (a) After stage 3,frequency segmentation decision, and cost trees for the low and high frequency regions. (b) After stage 2 segmentations with cost trees in each region. (c) After stage 1, the trees are reduced to single points. These are the coefficients to be sent to the decoder.

ST	$S(N) = S(N/2)^2 + 1,\ S(2) = 2$
DT	$D(N) = D(N/2)^2 + S(N) - S(N/2),$ $D(2) = 2$
TF tree	$F(N) = 2 * F(N/2)^2 - F(N/4)^4,\ F(2) = 2$

Table 1: Comparison of number of bases searched by the Single-tree, Double-tree and Time-frequency tree (assume the Haar filters are used.)

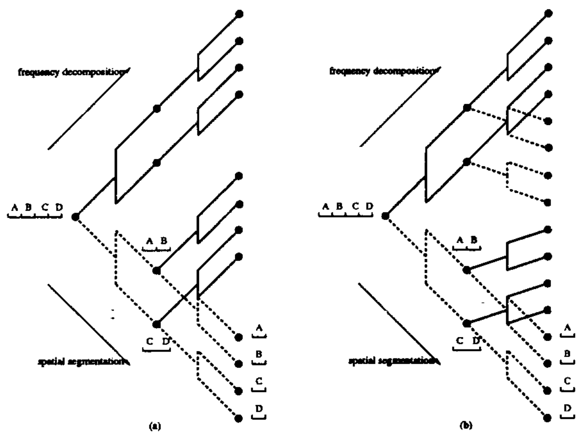

Figure 3: The time and frequency splits of the double-tree and Block Time-frequency algorithms. (a) Double-tree (b) Block Time-frequency.

We can for example use the orthogonal boundary filters developed in [3]. Also we can no longer use the algorithm directly; however, we can carry out the equivalent splittings as in Figure 3 (b) using non-block filters. In general the complexity of the algorithm will be greater than in the block transform case, but the increase need not be prohibitive, especially if we restrict the tree to some maximum depth. This is reasonable in practice, as we generally do not wish to continue splitting to the point where the subsignals become extremely short. The algorithm and complexity are explored in [4].

3 EXPERIMENTAL RESULTS

Using the extension to two-dimensions, and length-4 Daubechies filters we compressed a composite image made up of "Lena", "House", "Aerial" and "Scene" and also the images "Lena" and "Barbara." Using Rate + Distortion as cost, first order entropy as rate, and a single uniform quantizer we get the experimental results listed in Table 2.

REFERENCES

[1] R.R. Coifman and M.V. Wickerhauser. Entropy-based algorithms for best basis selection. *IEEE Trans. Information Theory*, 38(2):713–718, March 1992.

[2] C. Herley, J. Kovačević, K. Ramchandran, and M. Vetterli. Tilings of the time-frequency plane: Construction of arbitrary orthogonal bases and fast tiling algorithms. *IEEE Trans. on Signal Proc.*, 41(12):3341–3359, Dec. 1993.

[3] C. Herley and M. Vetterli. Orthogonal time-varying filter banks and wavelet packets. *IEEE Trans. on Signal Proc.*, 42(10):2650–2663, Oct. 1994.

[4] C. Herley, Z. Xiong, and K. Ramchandran. Joint space-frequency segmentation for least-cost image representation. *IEEE Trans. on Image Proc.*, 1994. Submitted.

[5] K. Ramchandran and M. Vetterli. Best wavelet packet bases in a rate-distortion sense. *IEEE Trans. on Image Proc.*, 2(2):160–175, 1992.

[6] L. Villemoes. A fast algorithm for adapted Walsh bases. *Journal of Appl. and Comput. Harmonic Analysis*, 1994. Submitted.

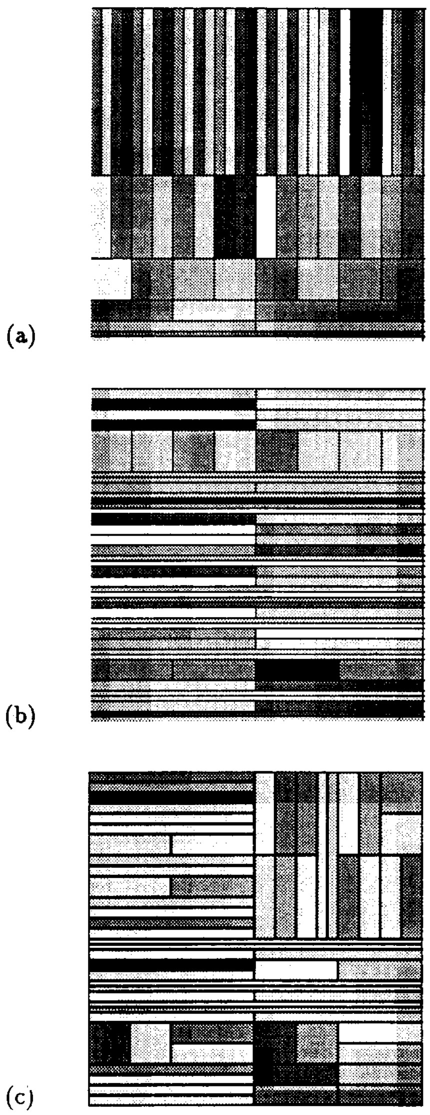

Figure 4: Tilings for the expansion of a high frequency 64-point sinusoid and spike using different bases. (a) Expansion using Discrete Wavelet Transform (cost 0.1585). (b) Expansion using the single-tree algorithm (cost 0.1008). (c) Expansion based on time-frequency segmentation (cost 0.0696).

	composite		Lena		Barbara	
	rate	PSNR	rate	PSNR	rate	PSNR
PCM	3.45	34.92	3.48	34.98	3.65	34.91
WT	1.48	36.46	0.95	37.55	1.63	36.74
ST	1.47	36.50	0.94	37.60	1.40	37.08
DT	1.44	36.53	0.94	37.60	1.40	37.08
TF	1.33	36.55	0.85	37.63	1.32	37.10

Table 2: Compression results for images using various alernative bases: Space domain, wavelet transform, single-tree, double-tree and Time-frequency tree.