# Fast Search for Best Representations in Multitree Dictionaries.

Yan Huang Ilya Pollak* Minh N. Do Charles A. Bouman

***Abstract*— We address the best basis problem—or, more generally, the best representation problem: given a signal, a dictionary of representations, and an additive cost function, the aim is to select the representation from the dictionary which minimizes the cost for the given signal. We develop a new framework of multitree dictionaries which includes some previously proposed dictionaries as special cases. We show how to efficiently find the best representation in a multitree dictionary using a recursive tree pruning algorithm. We illustrate our framework through several examples, including a novel block image coder which significantly outperforms both the standard JPEG and quadtree-based methods, and is comparable to embedded coders such as JPEG2000 and SPIHT.**

## I. Introduction.

A number of research efforts have recently concentrated on developing adaptive algorithms for representing and approximating signals in overcomplete dictionaries. This paper addresses the *best basis problem*—or, more generally, the *best representation problem*: given a signal, a dictionary of representations, and an additive cost function, the aim is to select the representation from the dictionary which minimizes the cost for the given signal. This paradigm has been successfully used for problems in compression [23], [24], [37], [49], estimation [11]–[13], [19], [20], [25], [29], [33], [51], and time-frequency (or space-frequency) analysis [7], [14]–[17], [46], [48], [52].

The original papers on best basis search [8], [9] considered the wavelet packet bases [8] and bases of local cosines [10], [26], [27], [38] on dyadic intervals. In each of these two cases, all the bases in the dictionary can be organized using a single tree: a binary tree in 1-D and a quadtree in 2-D. This organization was exploited in [8], [9] to devise a fast recursive tree pruning algorithm to find the best basis for any additive cost function.

Since then, a number of efforts have sought to lift the restrictions that a fixed binary/quadtree structure imposes on the underlying dictionary. Search methods for various dictionaries that correspond to different sets of possible time-frequency or space-frequency tilings have been proposed, such as the double-tree algorithm [14], time-frequency trees [46], [52], space-frequency trees [15], adaptive Haar-Walsh tilings [24], anisotropic wavelet packets [2], [11], anisotropic cosine packets [2], and mixed isotropic/anisotropic packets [2].

The main contributions of the present paper are:

- a new framework of multitree dictionaries which includes some previously proposed dictionaries as special cases;
- a fast recursive algorithm to find the best representation of data in a multitree dictionary;
- several application examples, including a novel image coder, which typically reduces the bit rate by about 25-40% compared to JPEG and by about 10-20% compared to the quadtree-based approach of [37], and whose rate-distortion performance is comparable to that of embedded wavelet coders such as JPEG2000 and SPIHT.

We start our discussion in Section II with a simple example of an optimal rectangular tiling algorithm. A simple modification of this algorithm leads to a best wedgelet algorithm for arbitrary rectangular tilings which we present in Section III. Two further extensions of our basic tiling algorithm are described in Section IV. Section V applies our algorithm to the problem of image compression. In Section VI, we introduce the general framework of multitree dictionaries, and argue that the algorithms of Sections II, III, and IV are special cases of a general recursive algorithm for finding the best object in a multitree dictionary. In Section VII, we then discuss relationships of our framework and algorithms to previously proposed best basis algorithms, and to other application areas.

## II. Example 1: Optimal Rectangular Tilings.

### A. A Fast Recursive Tiling Algorithm.

We consider all images supported on a discrete rectangular domain $Q \subset \mathbb{Z}^2$. Suppose we are given an image $f$ and would like to segment it into rectangular tiles $P_1, P_2, \ldots, P_d$ so as to minimize a cost which is equal to the sum of the costs of the individual tiles:

$$\sum_{i=1}^{d} e(P_i), \tag{1}$$

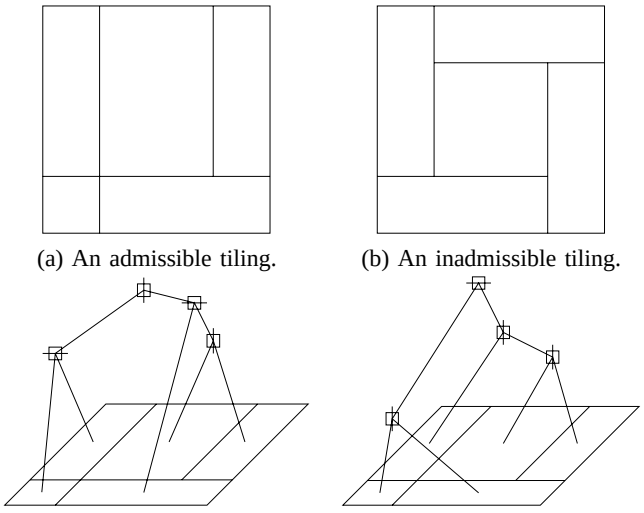
where $e$ is a cost function which is application specific and which depends on the image $f$.

We restrict our choice of tilings, and only consider those tilings that can be obtained through the following recursive binary splitting process:

- start with a tiling which consists of a single tile—namely, the whole image domain;

This work was supported in part by a National Science Foundation (NSF) CAREER award CCR-0093105, an NSF grant IIS-0329156, a Purdue Research Foundation grant, and an NSF CAREER award CCR-0237633.

Y. Huang, I. Pollak*, and C.A. Bouman are with the School of Electrical and Computer Engineering, Purdue University, 1285 EE Building, West Lafayette, IN 47907, phone 765-494-3465, 5916, and 0340, fax 765-494-3358, e-mail yanh,ipollak,bouman@ecn.purdue.edu. M.N. Do is with the Department of Electrical and Computer Engineering, Coordinated Science Lab, and Beckman Institute, University of Illinois at Urbana-Champaign, Urbana, IL 61801, phone 217-244-4782, fax 217-244-1642, e-mail minhdo@uiuc.edu. Corresponding author's e-mail: ipollak@ecn.purdue.edu.



(a) An admissible tiling. (b) An inadmissible tiling.

(c) A tree of splits. (d) Another tree of splits.

Fig. 1. An illustration of tilings and trees of splits. (a) An admissible tiling—i.e., a tiling that can be obtained via recursive binary splitting. (b) An inadmissible tiling. (c) A tree of splits that leads to the tiling in (a). (d) Another tree of splits that leads to the tiling in (a).

- for every tile in the tiling which consists of more than one pixel,
  either keep it and do not split it ever again,
  or split it into two smaller rectangular tiles;
- continue until all the tiles in the tiling either consist of one pixel or are labeled "never split again".

A rectangular tiling which can be obtained through this procedure is called an *admissible tiling*. An admissible tiling is illustrated in Fig. 1(a). The rectangular tiling depicted in Fig. 1(b) cannot be obtained through the binary splitting process described above, even though every tile in the tiling is a rectangle. This tiling is therefore not an admissible tiling.

The binary splitting process is conveniently visualized as a tree, with every node of the tree corresponding to a unique rectangular region of the image, as shown in Fig. 1(c).[1] We therefore use the terms *node* and *rectangular region* interchangeably. In particular, the entire image domain corresponds to the root of the tree. The yield of the binary tree—i.e., the set of all leaves—is then a tiling of the image. We therefore use the terms *leaf node* and *tile* interchangeably. The set of all such trees will give us the set of all admissible tilings (however, several different trees may correspond to the same tiling, as shown in Fig. 1(c,d)).

To efficiently solve our optimal tiling problem, we assign the cost given in Eq. (1) to every tree $t$ whose yield is an admissible tiling $\{P_1, \ldots, P_d\}$:

$$\text{COST0}(t) = \sum_{P \in \text{yield}(t)} e(P). \tag{2}$$

We then search over all trees to find one of the trees with the smallest cost. The optimal tiling is then the yield of this tree. Since our search space consists of multiple trees, we call it a *multitree dictionary*. Our efficient search algorithm exploits the fact that although the number of possible trees and tilings is very large (see [25], [53] and Appendix), the number of rectangular tiles is much smaller and manageable.

[1] In the figure, a short vertical (horizontal) line through a node signifies a vertical (horizontal) split.

```
(C*_P, s*_P) ← best_split_v0(P) {
    if C*_P has been computed
        get C*_P and s*_P from the global data structure TABLE;
    else {
        s*_P ← ∅;              //Initialize best left child s*_P
        C*_P ← e(P);           //Initialize best cost C*_P
        for (P', P'') = a partition of P into two rectangles {
            (C*_P', s*_P') ← best_split_v0(P');
            (C*_P'', s*_P'') ← best_split_v0(P'');
            if C*_P' + C*_P'' < C*_P {
                s*_P ← P';                  //Update s*_P
                C*_P ← C*_P' + C*_P'';      //Update C*_P
            }
        }
        record C*_P and s*_P in the global data structure TABLE;
    }
    return C*_P and s*_P;
}
```

(a) Recursive calculation of the optimal splits and corresponding costs.

```
B*_P ← best_tiling_v0(P) {
    get s*_P from the global data structure TABLE;
    if s*_P is the empty set
        B*_P ← {P};
    else
        B*_P ← best_tiling_v0(s*_P) ∪ best_tiling_v0(P\s*_P);
    return B*_P;
}
```

(b) Recursive generation of the best tiling.

Fig. 2. Pseudocode specification of a fast recursive search for the best rectangular tiling: (a) the recursive calculation of the optimal left children $s^*_P$ and the corresponding costs $C^*_P$; (b) the recursive generation of the best tiling. It is assumed that both routines have access to the same global data structure TABLE. The optimal tiling $\mathcal{B}^*_Q$ of an image domain $Q$ is obtained with $(C^*_Q, s^*_Q) \leftarrow$ best_split_v0$(Q)$, followed by $\mathcal{B}^*_Q \leftarrow$ best_tiling_v0$(Q)$.

To describe our search algorithm, let $C^*_P$ be the cost of the optimal tiling for a rectangle $P$. In particular, $C^*_Q = \min_t \text{COST0}(t)$ is the optimal cost for the entire image domain $Q$. Our algorithm makes the following recursive call, starting with $P = Q$:

$$C^*_P = \min\{e(P),\ \min(C^*_{P'} + C^*_{P''})\}, \tag{3}$$

where the inner minimization is done over all ordered pairs of rectangles $(P', P'')$ which partition the rectangle $P$:

$$P = P' \cup P'' \text{ and } P' \cap P'' = \varnothing.$$

We always assume that, if the split is horizontal, then $P'$ is on top of $P''$, and if the split is vertical, then $P'$ is to the left of $P''$.

The recursive call (3) terminates at the pixels:

$$\text{if } P \text{ is a pixel, then } C^*_P = e(P). \tag{4}$$

To avoid repetitive calculation, we store the optimal cost and the optimal split for each rectangle in a table. Before making a recursive call for any rectangle $P$, the table is

consulted to make sure that $P$ has not been visited before. If the original image domain is $N_1 \times N_2$, it has $O(N_1^2 N_2^2)$ different subrectangles, and therefore maintaining the table requires $O(N_1^2 N_2^2)$ memory. With this table, we only need to make one recursive call per rectangle. Since each recursive call involves $O(N_1 + N_2)$ comparisons to calculate $C_P^*$ via Eq. (3)—corresponding to $N_1 - 1$ horizontal splits and $N_2 - 1$ vertical splits—the computational complexity of the search algorithm is $O(N_1^2 N_2^2 (N_1 + N_2))$ which is $O(N^{2.5})$ for a square image[2] with $N$ pixels, $N_1 = N_2 = \sqrt{N}$.

The pseudocode for the search algorithm is shown in Fig. 2. The optimal left child of $P$ is denoted by $s_P^*$, and the optimal overall tiling by $\mathcal{B}_P^*$. Fig. 2(a) shows the pseudocode for the recursive calculation of the optimal splits and corresponding costs which are stored in a global data structure TABLE. Once this piece of pseudocode is executed, the optimal tiling is constructed using the routine in Fig. 2(b) which is assumed to have access to the same global data structure TABLE. Specifically, the optimal tiling $\mathcal{B}_Q^*$ of an image domain $Q$ is obtained with the following two commands:

$$\begin{aligned} (C_Q^*, s_Q^*) &\leftarrow \text{best\_split\_v0}(Q), \\ \mathcal{B}_Q^* &\leftarrow \text{best\_tiling\_v0}(Q), \end{aligned}$$

which call the two routines in Fig. 2.

### B. A Simple Cost Function.

The preceding discussion supposes that the individual costs $e(P)$ have been precomputed for every rectangle $P$. We analyze this computation using the following simple cost:

$$e(P) = \sum_{(n_1,n_2)\in P} (f(n_1,n_2) - f_P)^2 + w, \tag{5}$$

which results in the following overall cost of a tiling $\{P_1, \ldots, P_d\}$:

$$\sum_{i=1}^{d} \sum_{(n_1,n_2)\in P_i} (f(n_1,n_2) - f_{P_i})^2 + wd, \tag{6}$$

where

$f(n_1, n_2)$ is the pixel value at the location $(n_1, n_2)$;
$f_{P_i}$ is the average of the image $f$ over the rectangle $P_i$;
$d$ is the number of tiles in the tiling;
$w$ is an application-specific penalty on the number of tiles (such as, e.g., the average coding complexity in a compression application).

For this particular cost function (5), computing $e(P)$ for every rectangle $P$ can be done very efficiently by defining the following two variables:

$$\begin{aligned} \rho_1(f,P) &= \sum_{(n_1,n_2)\in P} f(n_1,n_2) = |P| f_P \\ \rho_2(f,P) &= \sum_{(n_1,n_2)\in P} f(n_1,n_2)^2, \end{aligned}$$

and noticing that, if we know these two variables for a pair of rectangles $(P', P'')$ which partition a rectangle $P$, we can calculate $e(P)$ in $O(1)$ time as follows:

$$\begin{aligned} \rho_1(f,P) &= \rho_1(f,P') + \rho_1(f,P'') \\ \rho_2(f,P) &= \rho_2(f,P') + \rho_2(f,P'') \\ e(P) &= \rho_2(f,P) - \rho_1^2(f,P)/|P| + w. \end{aligned}$$

This is used to compute all the costs in a bottom-up fashion, with both time and space complexity $O(N_1^2 N_2^2)$.

### C. Reducing the Computational Complexity.

The overall time complexity of the optimal tiling algorithm with the cost (5)—i.e., the computation of the costs and the recursive search combined—is $O(N_1^2 N_2^2 (N_1 + N_2))$. The overall space complexity is $O(N_1^2 N_2^2)$.

Note that reducing the number of admissible rectangular tilings may result in a lower computational complexity of the algorithm. For example, we can restrict the search space if we only allow a rectangle to be split into two congruent rectangles, as was done in, e.g., [11]. In other words, we can impose that during our recursive binary splitting process, an $n_1 \times n_2$ rectangle may only be split either into two $n_1/2 \times n_2$ rectangles, or into two $n_1 \times n_2/2$ rectangles. This "dyadic tiling" scenario is called "dyadic CART" in [11] and is similar to the anisotropic wavelet packets [2], [11].[3] It can be shown that in this case, the total number of possible rectangular tiles is $O(N_1 N_2)$, and therefore the computation of the costs has time and space complexity $O(N_1 N_2)$. The minimization in Eq. (3) is $O(1)$ since it now involves choosing one of no more than three options: horizontal split or vertical split or no split. Therefore, both the time and space complexity of the search is $O(N_1 N_2)$, which is also the overall complexity of the algorithm—i.e., the computation of the costs and the recursive search combined. In this case, the complexity is linear in the number of pixels.

Another way of reducing the computation time and memory requirements is restricting the split locations to only occur at multiples of some integer $M > 1$. In this case, the elementary cells in the resulting tilings will be $M \times M$ rectangles rather than single pixels. Our rectangular tiling algorithms, with $M = 16$, are illustrated in Fig. 3: Fig. 3(b) shows the result of the dyadic search, and Fig. 3(c) shows the result of the search for arbitrary split locations.

We also note that for any set of admissible tilings, a further reduction in computational complexity can be achieved by sacrificing optimality and using a suboptimal, greedy search method proposed in, e.g., [30], [31].

The problems addressed in the remainder of the paper exemplify many situations where the computation of the costs may be more complex than $O(1)$ per pixel and in fact may dominate the computational complexity of the overall algorithm.

[2]For a $N_1 \times N_2 \times \ldots \times N_D$ $D$-dimensional hyperrectangle, it is similarly shown that the complexity of the search is $O(N_1^2 \ldots N_D^2 (N_1 + \ldots + N_D))$, which is $O(DN^{2+1/D})$ for a $D$-dimensional hypercube with $N$ voxels, $N_1 = \ldots = N_D = N^{1/D}$.

[3]The scenario which is similar to the classical wavelet packets results from imposing that, furthermore, any horizontal split must be followed by a vertical one, and vice versa. In other words, if an $n_1 \times n_2$ rectangle resulted from a horizontal split, it is only allowed to be split into two $n_1 \times n_2/2$ rectangles; and if it resulted from a vertical split, it is only allowed to be split into two $n_1/2 \times n_2$ rectangles.

(a) Cameraman image.

(b) Best dyadic tiling, cost 0.57

(c) Best arbitrary tiling, cost 0.44

Fig. 3. A $256 \times 256$ cameraman image and its best rectangular tilings with the smallest cell size $16 \times 16$: (b) best dyadic tiling, cost 0.57; (c) best arbitrary tiling, cost 0.44.

## III. Example 2: Optimal Wedgelet Tilings.

### A. Algorithm Extension 1: State Variables.

In the best wedgelet algorithm [12], each tile can be represented using one of several wedgelets. In our image coding algorithm in Section V, we will allow the choice of several quantizers for encoding each tile. To model these choices, we introduce the concept of a *state variable*. To every tile $P$, we associate a state variable $x_P$ taking values in some finite set which, without loss of generality, we assume to be $\{1, 2, \ldots, X\}$ where $X$ is some fixed integer. Each term of the cost function is now allowed to depend on the corresponding state variable—in other words, we replace the cost given in Eq. (2) with the following:

$$\text{COST1}(t) = \sum_{P \in \text{yield}(t)} c(P, x_P). \tag{7}$$

Note that if we let $e(P) = \min_{x_P} c(P, x_P)$, this cost becomes the same as COST0 in Eq. (2). Therefore, the search for the best tree and the best tiling now consists of two steps: finding the best state for each tile $P$ via minimizing $c(P, x_P)$ with respect to $x_P$, and then applying our recursive algorithm of Fig. 2(a).

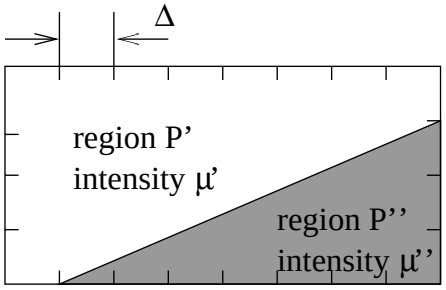



Fig. 4. A wedgelet.

### B. Wedgelet Experiments.

A wedgelet [12] is an image defined on a rectangular domain and consisting of two constant pieces which are joined together along a straight line, as illustrated in Fig. 4. We can represent a wedgelet on a domain $P$ as a quadruple $x_P = (P', P'', \mu', \mu'')$ where $P'$ and $P''$ are the two regions that the straight line partitions $P$ into, and $\mu'$ and $\mu''$ are the respective image intensities. Alternatively, $P'$ and $P''$ can be specified by the two endpoints of the line. It is typically assumed that the endpoints are restricted to a grid with some small step $\Delta$, as shown in Fig. 4.

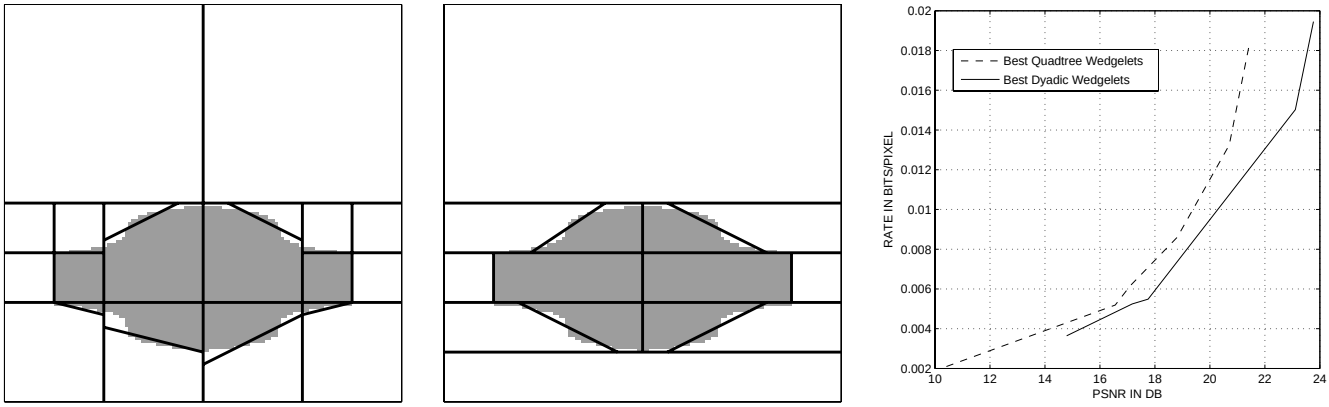
Given an image $f$, we can approximate the image values over a rectangular domain $P$ with a wedgelet $x_P = (P', P'', f_{P'}, f_{P''})$ where $f_{P'}$ and $f_{P''}$ are the average intensities of $f$ over the regions $P'$ and $P''$, respectively. We penalize any such approximation using the following simple cost function which is similar to Eq. (5):

$$\begin{aligned} c(P, x_P) \quad = \quad & \sum_{(n_1,n_2) \in P'} (f(n_1, n_2) - f_{P'})^2 \\ & + \sum_{(n_1,n_2) \in P''} (f(n_1, n_2) - f_{P''})^2 + 2w. \end{aligned}$$

In addition, we still allow approximating an image tile with a constant, and still use the cost in Eq. (5) in this case.

Our fast search algorithm can then find the optimal wedgelet tiling. Fig. 5 depicts some examples for a binary image. Fig. 5(a) shows the best quadtree wedgelet tiling. This strategy was proposed in the original wedgelet paper [12]. Allowing more possibilities for split locations leads to more compact and more precise wedgelet tilings. The best dyadic wedgelet tiling is shown in Fig. 5(b) and allows each rectangle to be split into two congruent rectangles either horizontally or vertically.

We assumed the following simple approximation for the number of bits required to encode our wedgelet tilings:




(a) Quadtree wedgelets. (b) Dyadic wedgelets. (c) Rate-distortion curves.

Fig. 5. Two best wedgelet tiling examples for an $128 \times 128$ binary image: (a) Quadtree wedgelets, SNR=17.1 dB, rate = 0.0062 bits per pixel; (b) Dyadic wedgelets, SNR = 17.8 dB at 0.0055 bits per pixel. Panel (c) shows the rate-distortion curves for this image, for the quadtree wedgelets (dashed) and the dyadic wedgelets (solid).

- one bit per node to encode whether it is an internal node or a leaf;
- one bit per leaf node to encode whether it is a constant tile or a wedgelet;
- one bit per leaf node to encode the intensity (this is a reasonable approximation, since our input image is binary);
- $\log_2(((M+N)/\Delta)^2)$ bits per wedgelet leaf node of size $M \times N$, to encode the position of the wedgelet partition;
- in addition, for dyadic wedgelet tilings, we spend one bit per internal node to encode whether it is split horizontally or vertically.

With these assumptions, the quadtree tiling of Fig. 5(a) produces SNR of 17.1 dB and rate 0.0062 bits per pixel, whereas Fig. 5(b) has both a higher SNR of 17.8 dB and a lower rate of 0.0055 bits per pixel. Note also that the quadtree tiling has 16 tiles whereas the dyadic tiling has only eight tiles. Dyadic tilings outperform quadtree tilings, achieving lower rates at the same SNR's and higher SNR's at the same rates for this image, as shown in Fig. 5(c). The curves in Fig. 5(c) were obtained by varying the split penalty $w$.

## IV. Further Extensions of the Optimal Tiling Algorithm.

### A. Algorithm Extension 2: Incorporating Internal Nodes into the Cost.

Recall that in previous sections, the trees played an auxiliary role since the cost only depended on the yield of the tree—i.e., the leaf nodes—but was independent of the internal nodes of the tree. However, in some applications the internal structure of the tree matters. For example, in the wedgelet experiments of the previous section as well as in the compression experiments which will be discussed in Section V, the structure of the tree must be encoded, and the encoding costs may be different for two different trees which correspond to the same tiling. We would like to be able to include these costs in the cost function optimized by our algorithm. To model this and a variety of other such situations where the internal structure of the tree is important, we now equip every node $P$ with a state $x_P$, and use a cost function $\bar{c}$ to penalize the split of a node $P$ with a state $x_P$ into nodes $P'$ and $P''$ with states $x_{P'}$ and $x_{P''}$, respectively. Our new cost for any tree $t$ is:

$$\begin{aligned} \text{COST2}(t) \quad = \quad & \sum_{P \in \text{internal-nodes}(t)} \bar{c}\left( \begin{array}{ccc} & (P, x_P) & \\ & \diagup \quad \diagdown & \\ (P', x_{P'}) & & (P'', x_{P''}) \end{array} \right) \\ & + \sum_{P \in \text{yield}(t)} c(P, x_P), \end{aligned} \tag{8}$$

where

in the first summation, the nodes $P'$ and $P''$ are the children of the node $P$ on the tree $t$;
$x_P$, $x_{P'}$, and $x_{P''}$ are the state variables associated with the nodes $P$, $P'$, and $P''$, respectively;
$c$ and $\bar{c}$ are application-specific cost functions.

Note that this cost is a generalization of $\text{COST1}(t)$ in Eq. (7). Indeed, if we set $\bar{c} \equiv 0$, then $\text{COST2}(t) = \text{COST1}(t)$. Note also that, in the cost (5,6) which we used in our tiling experiments, the penalty $w$ can be interpreted as a split cost function $\bar{c}$ which assigns a constant penalty $w$ to each split.

We let $\bar{C}^*_{P,x}$ be the cost of the optimal tree for a rectangle $P$, given $x_P = x$, and we let $\bar{C}^*_P$ be the cost of the overall optimal tree for $P$, i.e., $\bar{C}^*_P = \min_x \bar{C}^*_{P,x}$. The optimal tree is found using the recursion in Eq. (9). This recursion is similar to Eqs. (3,4) and can therefore be implemented using the pseudocode in Figs. 6 and 7 which are extensions of Figs. 2(a) and 2(b), respectively.

### B. Algorithm Extension 3: Dynamic Programming Over a Sequence of Blocks.

If an image is partitioned into $K$ blocks $Q_1, Q_2, \ldots, Q_K$—as in, for example, JPEG and [37]—our algorithm can be used to find the optimal tiling within each block. In [37], it was assumed that each block is handled independently. However, as argued in [5], [40], it is sometimes advantageous to assume that pairs of consecutive blocks are interdependent. In order to

$$\bar{C}^*_{P,x} = \begin{cases} c(P,x), & \text{if } P \text{ is an elementary cell,} \\ \min\left\{ c(P,x),\ \min\limits_{P',P'',x',x''} \left[ \bar{c}\left( \begin{array}{ccc} & (P,x) & \\ & \diagup\ \diagdown & \\ (P',x') & & (P'',x'') \end{array} \right) + \bar{C}^*_{P',x'} + \bar{C}^*_{P'',x''} \right] \right\}, & \text{otherwise.} \end{cases} \tag{9}$$

```
(C̄*_{P,x}, s̄*_{P,x}) ← best_split_v2(P, x) {
  if C̄*_{P,x} has been computed
    get C̄*_{P,x} and s̄*_{P,x} from the global data structure TABLE;
  else {
    // Initialize
    s̄*_{P,x} ← ((∅, 0), (∅, 0));
    C̄*_{P,x} ← c(P, x);
    for x' = 1 : X, x'' = 1 : X, (P', P'') = a partition of P {
      (C̄*_{P',x'}, s̄*_{P',x'}) ← best_split_v2(P', x');
      (C̄*_{P'',x''}, s̄*_{P'',x''}) ← best_split_v2(P'', x'');
      if C̄*_{P',x'} + C̄*_{P'',x''} + c̄ ( (P,x) / (P',x') (P'',x'') ) < C̄*_{P,x} {
        // Update
        s̄*_{P,x} ← ((P', x'), (P'', x''));
        C̄*_{P,x} ← C̄*_{P',x'} + C̄*_{P'',x''} + c̄ ( (P,x) / (P',x') (P'',x'') );
      }
    }
    record C̄*_{P,x} and s̄*_{P,x} in the global data structure TABLE;
  }
  return C̄*_{P,x} and s̄*_{P,x};
}
```

Fig. 6. Pseudocode for the recursive calculation of the optimal splits and states and the corresponding costs for COST2 of Section IV-A.

```
t*_{P,x} ← best_tree_v2(P, x) {
  get s̄*_{P,x} ≡ ((P', x'), (P'', x'')) from the global TABLE;
  if P' is the empty set
    t*_{P,x} ← [(P, x)];
  else
    t*_{P,x} ← [ (P,x) / best_tree_v2(P', x') best_tree_v2(P'', x'') ];
  return t*_{P,x};
}
```

Fig. 7. Pseudocode for the recursive generation of the best tree for Section IV-A.

model this new assumption, we let $t_1, \ldots, t_K$ be the trees corresponding to the blocks $Q_1, \ldots, Q_K$, respectively, and assign the following cost to this collection of trees $\{t_1, \ldots, t_K\}$:

$$\text{COST-BLOCKS}(t_1, \ldots, t_K) = \sum_{k=2}^{K} \bar{\bar{c}}(Q_k, x_{Q_k}, Q_{k-1}, x_{Q_{k-1}}) + \sum_{k=1}^{K} \text{COST2}(t_k). \tag{10}$$

Let $\bar{\bar{C}}^*_{1:i,x}$ be the optimal cost for $i$ blocks, given that $x_{Q_i} = x$. In other words, $\bar{\bar{C}}^*_{1:i,x}$ is defined as the result of minimizing COST-BLOCKS$(t_1, \ldots, t_i)$ subject to $x_{Q_i} = x$. Then we have the following recursion for $\bar{\bar{C}}^*_{1:i,x}$:

$$\bar{\bar{C}}^*_{1:i,x} = \begin{cases} \bar{C}^*_{Q_1,x} & \text{for } i = 1, \\ \min\limits_{x'} (\bar{\bar{c}}(Q_i, x, Q_{i-1}, x') + \bar{C}^*_{Q_i,x} + \bar{\bar{C}}^*_{1:i-1,x'}) & \text{for } i = 2, \ldots, K, \end{cases} \tag{11}$$

where $\bar{C}^*_{Q_i,x}$ is computed through the recursion (9), using the pseudocode in Fig. 6. The overall optimal cost, which we denote $\bar{\bar{C}}^*_{1:K}$, is found from:

$$\bar{\bar{C}}^*_{1:K} = \min_x \bar{\bar{C}}^*_{1:K,x}.$$

This recursive calculation is performed using the dynamic programming algorithm of Fig. 8, similar to those used in [5], [40].

```
(t*_1, ..., t*_K) ← best_tree_sequence(Q_1, ..., Q_K) {
  // Initialization
  for x = 1 : X, P = Q_1 : Q_K
        (C̄*_{P,x}, s̄*_{P,x}) ← best_split_v2(P, x);
  for x = 1 : X {
    C̄̄*_{1:1,x} ← C̄*_{Q_1,x};
    optimal_previous_state_{1:1,x} ← 0;
  }
  // Forward sweep
  for i = 2 : K
    for x = 1 : X {
      C̄̄*_{1:i,x} ← min_{x'} (c̄̄(Q_i, x, Q_{i-1}, x') + C̄*_{Q_i,x} + C̄̄*_{1:i-1,x'});
      optimal_previous_state_{1:i,x} ← arg min_{x'} (c̄̄(Q_i, x, Q_{i-1}, x')
                                          + C̄*_{Q_i,x} + C̄̄*_{1:i-1,x'});
    }
  //Backtracking
  x* = arg min_x C̄̄*_{1:K,x};
  for i = K : −1 : 1 {
    t*_i ← best_tree_v2(Q_i, x*);
    x* ← optimal_previous_state_{1:i,x*};
  }
  return t*_1, ..., t*_K;
}
```

Fig. 8. Pseudocode for the dynamic programming over blocks, Section IV-B.

## V. Example 3: Multitree Image Coding Algorithm.

We fuse our rectangular tiling algorithm with several aspects of the compression strategy in [37], to obtain an image coder which finds the optimal tiling, and encodes every tile. The input is partitioned into blocks $Q_1, \ldots, Q_K$, in the raster order. Within each block, we find the optimal tree $t^*_k$ and encode it as follows:

- one bit per node is used to indicate whether the node is an internal node or a leaf;

- for each node with a state $x \in \{1, \ldots, X\}$, we use $\lceil \log_2 X \rceil$ bits to encode the state $x$;
- $\lceil \log_2 \text{SPLITS}_P \rceil$ bits are used to encode the split location for every internal node $P$, where $\text{SPLITS}_P$ is the total number of possible split locations for the node $P$.

To find the optimal tree, we optimize with respect to the rate-distortion cost [37] $D + \lambda R$, where $R$ is the number of bits it takes to encode the image, $D$ is the total distortion, and $\lambda$ is a parameter. We assume that the distortion $D$ is additive over the tiles and over the blocks. In our experiments, we use the sum of squared differences as our distortion criterion. For each tile, we follow a JPEG-like procedure which finds the DCT coefficients, quantizes them, and entropy-codes the AC coefficients and differential DC coefficients. The DC coefficients are differentially coded in the following manner:

- the root DC coefficient for the first block $Q_1$ is encoded;
- the difference between the root DC coefficients for the $k$-th block and the $(k-1)$-st block is encoded, for $k = 2, \ldots, K$;
- for every leaf node $P$ of every tree $t_k^*$, the difference between the DC coefficient for $P$ and the root DC coefficient is encoded.

Following [37], we assume that one of several quantizers can be used for each tile, and optimize our choice of the quantizer for each tile concurrently with the search for the optimal tiling. The state $x_P$ corresponds to the quantizer used for the tile $P$. In addition, we allow the choice of the same set of quantizers to encode the root DC coefficient.

Because of the differential coding of the DC coefficients, the bit rate within each block can be shown to have the form of Eq. (8), and the overall bit rate is additive over pairs of consecutive blocks and is therefore of the form (10). This, combined with the additivity of the distortion, means that the overall cost $D + \lambda R$ is of the form (10). This means that, in order to optimize it, we can use the algorithm of Section IV-B and Fig. 8.

In order to minimize the distortion subject to a fixed rate, or to minimize the rate subject to a fixed distortion, our optimization algorithm can be used within an iterative procedure similar to that of [37].

### A. Compression Experiments.

We compare our multitree-JPEG compression algorithm with standard JPEG and with the quadtree-based algorithm of [37].[4] We test the algorithms on four images: a $512 \times 512$ image "barbara", and three $256 \times 256$ images "goldhill," "lenna," and "cameraman". The corresponding sets of rate-distortion curves are shown in Fig. 9. In each figure, the rate in bits per pixel is plotted against the peak signal-to-noise ratio (PSNR). For each quadtree and multitree experiment, a target distortion was fixed, and the rate was minimized. Note that our multitree algorithm (solid) outperforms the standard JPEG (dash) by about 2-4 dB and the quadtree algorithm (dashdot) by about 1-2 dB at a fixed bit rate. Equivalently, the multitree algorithm represents compression savings of about 25-40% over the standard JPEG and 10-20% over the quadtree algorithm, for a fixed PSNR.

In these experiments, we take the block size to be $16 \times 16$ and we take the smallest cell size to be $4 \times 4$—i.e., we allow rectangular tiles with sides 4, 8, 12, and 16. This means that, for each $16 \times 16$ block, we search over 68480 distinct tilings—this is in contrast to the quadtree method which only allows 17 distinct tilings, and the standard JPEG which only considers one tiling. While the number of possible tilings for our method is drastically larger, the number of distinct subrectangles of each block—which is what determines the computational complexity of our algorithm—is only 100, compared to 21 for the quadtree method and 4 for the standard JPEG. Thus, we are able to search over a much larger set with only a modest increase in the computational burden. It is shown in Appendix that the increase in the allowed number of tilings is exponential as compared to the quadtree algorithm whereas the increase in the computational burden is only polynomial.

The results for the "barbara" image at PSNR = 36.4 dB are given in Fig. 10: the JPEG, quadtree, and multitree compression algorithms achieve 1.31, 1.00, and 0.83 bits per pixel, respectively. Note that the images look basically the same; however, the multitree algorithm gives compression savings of 37% over JPEG and 17% over the quadtree algorithm.

Fig. 11 illustrates the results for the same image at the bit rate 0.49 bits per pixel. (In this experiment, the bit rate was fixed at 0.49, and the distortions for the quadtree and multitree methods were minimized.) At this bit rate, the JPEG, quadtree, and multitree algorithms achieve PSNR's for the overall image of 28.3 dB, 30.5 dB, and 31.9 dB, respectively. A patch from the image and its three compressed versions is shown in Fig. 11. In addition to a higher signal-to-noise ratio, it is clear from the figure that the multitree algorithm results in both less blocky renditions of homogeneous areas of the image, sharper edges, and less ringing and blockiness in the textured areas and around the edges.

In these experiments, our implementation of JPEG is a baseline implementation which uses Huffman coding of the coefficients. To make the comparisons fair, we use similar Huffman coding strategies for the quadtree and multitree algorithms.

Further experiments show that, if we replace Huffman coding with arithmetic coding, then our multitree coder becomes competitive when compared to the state-of-the-art embedded wavelet coders such as JPEG2000 [45] and SPIHT [39] which both employ arithmetic coding.[5] Fig. 12 shows the rate-distortion curves for JPEG2000, SPIHT, and our multitree coder with arithmetic coding. The right column of the figure displays the bit rates as percentages of the multitree bit rate.

[4]The rate-distortion curves we obtain for the JPEG and quadtree algorithms are different from those given in [37] since we use a somewhat different implementation—for example, we use a different set of quantization matrices. However, the relative improvement of the quadtree algorithm over JPEG that we observe is similar to what is reported in [37].

[5]The improvement in performance is due to the fact that the baseline JPEG Huffman encoder, whose variant we use, is relatively poor. It has been observed that the QM arithmetic coder specified in the JPEG standard typically produces 5-10% compression savings, as compared to the baseline Huffman coder, see, for example, `sylvana.net/jpeg-ari/uncompressed/READ.txt`.

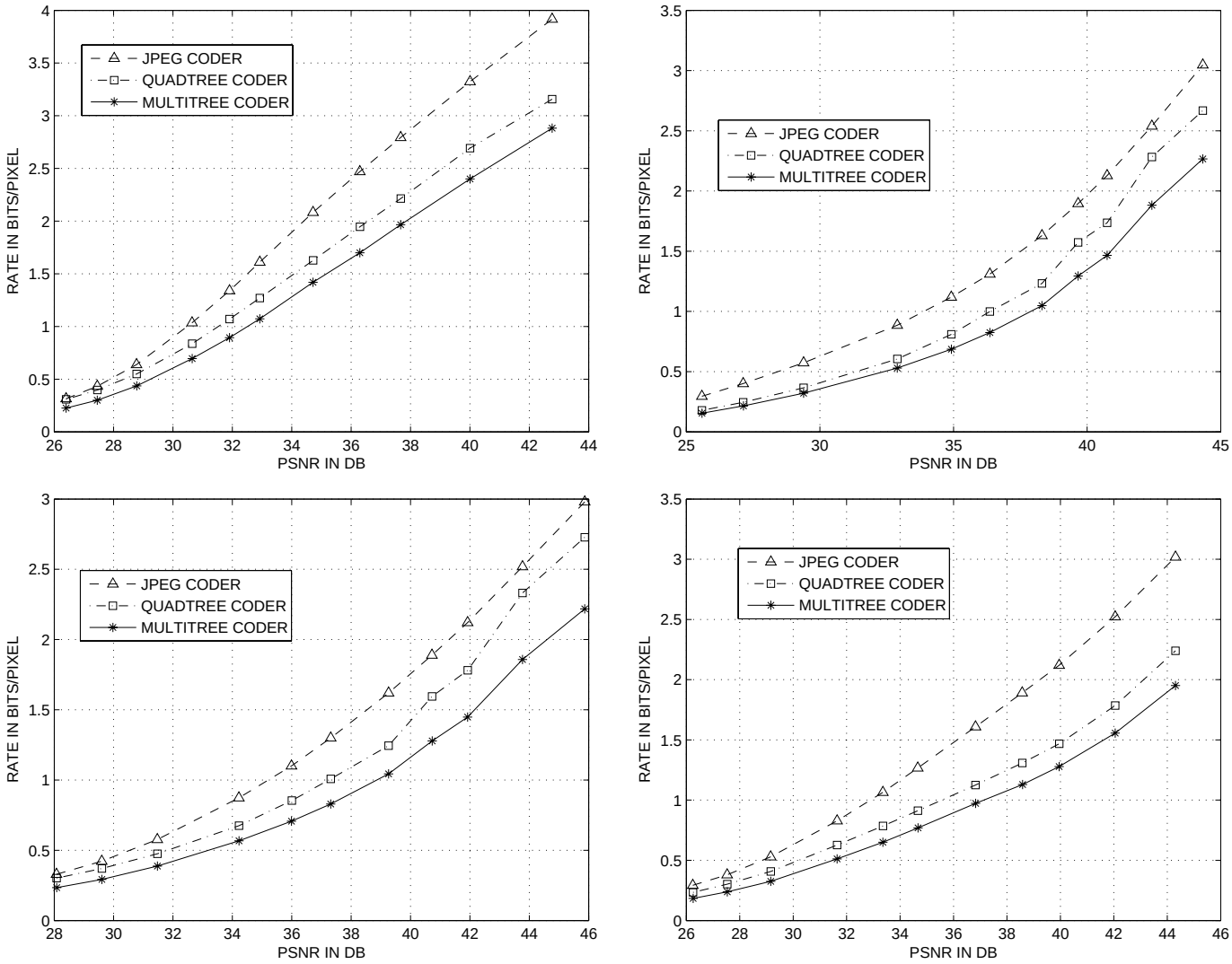



Fig. 9. Rate-distortion curves for "goldhill"(top left), "barbara" (top right), "lenna" (bottom left), and "cameraman" (bottom right).

For "goldhill" (top row) and "cameraman" (bottom row), our algorithm clearly outperforms both JPEG2000 and SPIHT. It also does better than SPIHT for "barbara" (second row) and better than JPEG2000 for "lenna" (third row). At low and high rates for "barbara," our algorithm is outperformed by JPEG2000; and at low rates for "lenna," it is outperformed by SPIHT. While the rate-distortion curves of the three algorithms are similar, our algorithm is potentially more amenable to implementations with much lower memory complexity, since, unlike JPEG2000 and SPIHT, it is based on small image blocks.

## VI. Multitree Dictionaries.

We now generalize our algorithms of Sections II, III, and IV-A and show that they are all instances of one general algorithm which is applicable to a wide variety of scenarios.

Tree models such as those of Sections II, III, and IV-A are conveniently described using the formalism of grammars. We define a *grammar* $G = (A, S)$ to be a pair of the following sets:

- a set $A$ of *symbols*,[6] and
- a set $S$ of *allowed splits*, also called *productions*, of the form $a \to \alpha$ where $a \in A$, and $\alpha$ is a finite sequence of elements of $A$.

For example, in Section IV-A, the symbols are pairs $(P, x)$ where $P$ is a rectangular region and $x \in \{1, \ldots, X\}$, and the productions are all of the form $(P, x) \to (P', x')\ (P'', x'')$ where $P'$ and $P''$ are two rectangles which partition $P$.

By starting with a single element of $A$, we can generate various sequences of elements of $A$ via recursive splitting—i.e., recursive application of productions. This process can be visualized as a tree where each production $a \to \alpha$ is depicted as a node labeled $a$ whose children are labeled with the elements of $\alpha$, left to right. We let $T(G)$ be the set of all

[6]This is somewhat different from standard treatments of grammars [28] which distinguish between the *start symbol* which can only appear at the root, the *nonterminal symbols* which can only appear at the nonroot internal nodes, and *terminal symbols* which can only appear at the leaves. We, on the other hand, assume that any symbol in $A$ can appear at the root or any internal nodes or leaf nodes.

(a) Original image

(b) JPEG, 1.31 bpp

(c) Quadtree compression, 1.00 bpp

(d) Multitree compression, 0.83 bpp

Fig. 10. Compression results for the "barbara" image at PSNR = 36.4 dB: (a) original image, (b) JPEG (rate = 1.31 bits per pixel), (c) quadtree compression (rate = 1.00 bits per pixel), and (d) multitree compression (rate = 0.83 bits per pixel).

trees that can be produced[7] by the grammar $G$.

Note that in the previous sections, the splitting process was binary and led to binary trees. Here, we allow splits into an arbitrary finite number of symbols.

We let a *multitree dictionary* $T_a(G)$ be the set of all trees in $T(G)$ whose root is labeled $a$. We say that a grammar $G = (A, S)$ is *finite-depth* if, for every $a \in A$, $T_a(G)$ is a finite set. This can be insured by only allowing a finite set of symbols to be descendants of $a$, and not allowing $a$ to be its own descendant.

Suppose that each symbol $u \in A$ is assigned a cost $c(u)$, and that each production $u \to \alpha \in S$ is assigned a cost $\bar{c}(u \to \alpha)$. Suppose further that the cost $\text{COST}(t)$ of any tree $t \in T_a(G)$ is the sum of the individual costs of all the productions comprising $t$, plus the sum of the costs of all its leaves:

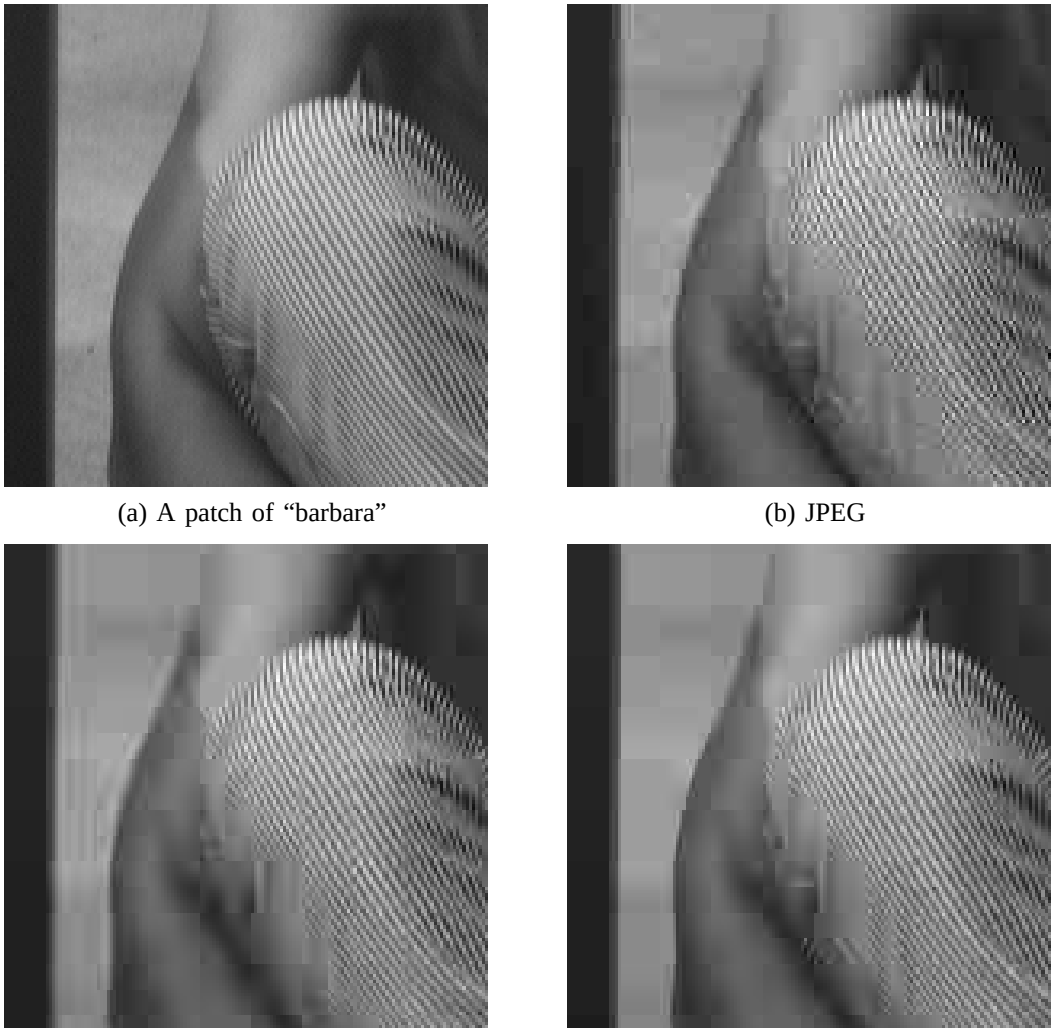
$$\text{COST}(t) = \sum_{u \to \alpha \in t} \bar{c}(u \to \alpha) + \sum_{u \in \text{yield}(t)} c(u). \qquad (12)$$

We would like to find the best tree in the dictionary $T_a(G)$ i.e., the tree $t_a^*$ whose cost is the smallest:

$$t_a^* = \arg \min_{t \in T_a(G)} \text{COST}(t).$$

We denote the corresponding cost by $C_a^*$, i.e., $C_a^* = C(t_a^*)$. We let $S_a$ be the set of all allowed splits of a fixed symbol $a$. To illustrate our fast recursive algorithm for best tree search, we first suppose that $S_a = \{a \to b_1\ b_2\}$. Then there is a

[7]We assume that each branch of our recursive tree generation process can stop after any number of recursions. This is different from standard treatments of grammars [28] where the stopping is handled via distinguishing between nonterminal symbols which must have children, and terminal symbols which never have children.



(a) A patch of "barbara" (b) JPEG

(c) Quadtree (d) Multitree

Fig. 11. Compression results for the "barbara" image at the bit rate of 0.49 bits per pixel: (a) a patch of the original image, (b) JPEG (PSNR for the overall image = 28.3 dB), (c) quadtree compression (PSNR = 30.5 dB), and (d) multitree compression (PSNR = 31.9 dB).

single tree in $T_a(G)$ which consists of one node labeled $a$ with $\text{COST}([a]) = c(a)$. For any other tree $t \in T_a(G)$, its left subtree $t_{left}$ is in $T_{b_1}(G)$, and its right subtree $t_{right}$ is in $T_{b_2}(G)$. Therefore, since the cost is additive,

$$\text{COST}(t) = \bar{c}(a \to b_1\ b_2) + \text{COST}(t_{left}) + \text{COST}(t_{right}).$$

Consequently, the optimal tree is:

$$t_a^* = \begin{cases} \left[\ \begin{array}{ccc} & a & \\ t_{b_1}^* & & t_{b_2}^* \end{array}\ \right] & \text{if } \bar{c}(a \to b_1\ b_2) + C_{b_1}^* + C_{b_2}^* < c(a) \\ [a] & \text{otherwise.} \end{cases}$$

In other words, we find the best trees $t_{b_1}^*$ and $t_{b_2}^*$ in the dictionaries $T_{b_1}(G)$ and $T_{b_2}(G)$, respectively, and compare their total cost plus the cost of the root production $a \to b_1\ b_2$, with the cost of the tree $[a]$.

We have a similar recursion in the general case. We let $R(a)$ be the set of the right-hand sides of all the elements of $S_a$. Then the possible candidates for $t_a^*$ are

$$\left[\ \begin{array}{ccc} & a & \\ t_{b_1}^* & \cdots & t_{b_{|\alpha|}}^* \end{array}\ \right] \text{ with cost } \bar{c}(a \to \alpha) + \sum_{i=1}^{|\alpha|} C_{b_i}^*,$$

for any $\alpha = (b_1\, b_2\, \ldots\, b_{|\alpha|}) \in R(a)$, and $[a]$, with cost $c(a)$.

To find the globally optimal $t_a^*$, we recursively search over these possibilities. The recursion terminates when $S_a = \varnothing$: in this case, $t_a^* = [a]$. The termination is guaranteed to happen in a finite number of steps for a finite-depth grammar. To avoid repetitive calculation, we store the optimal costs and corresponding productions in a global data structure called TABLE, as illustrated in the pseudocode of Fig. 13(a). Once this recursive call is done, the best tree can be generated from TABLE using the pseudocode in Fig. 13(b).

The most significant computational burden is in computing and storing the best costs and productions. To analyze this procedure, we let $A(a)$ be the union of $\{a\}$ and the set of

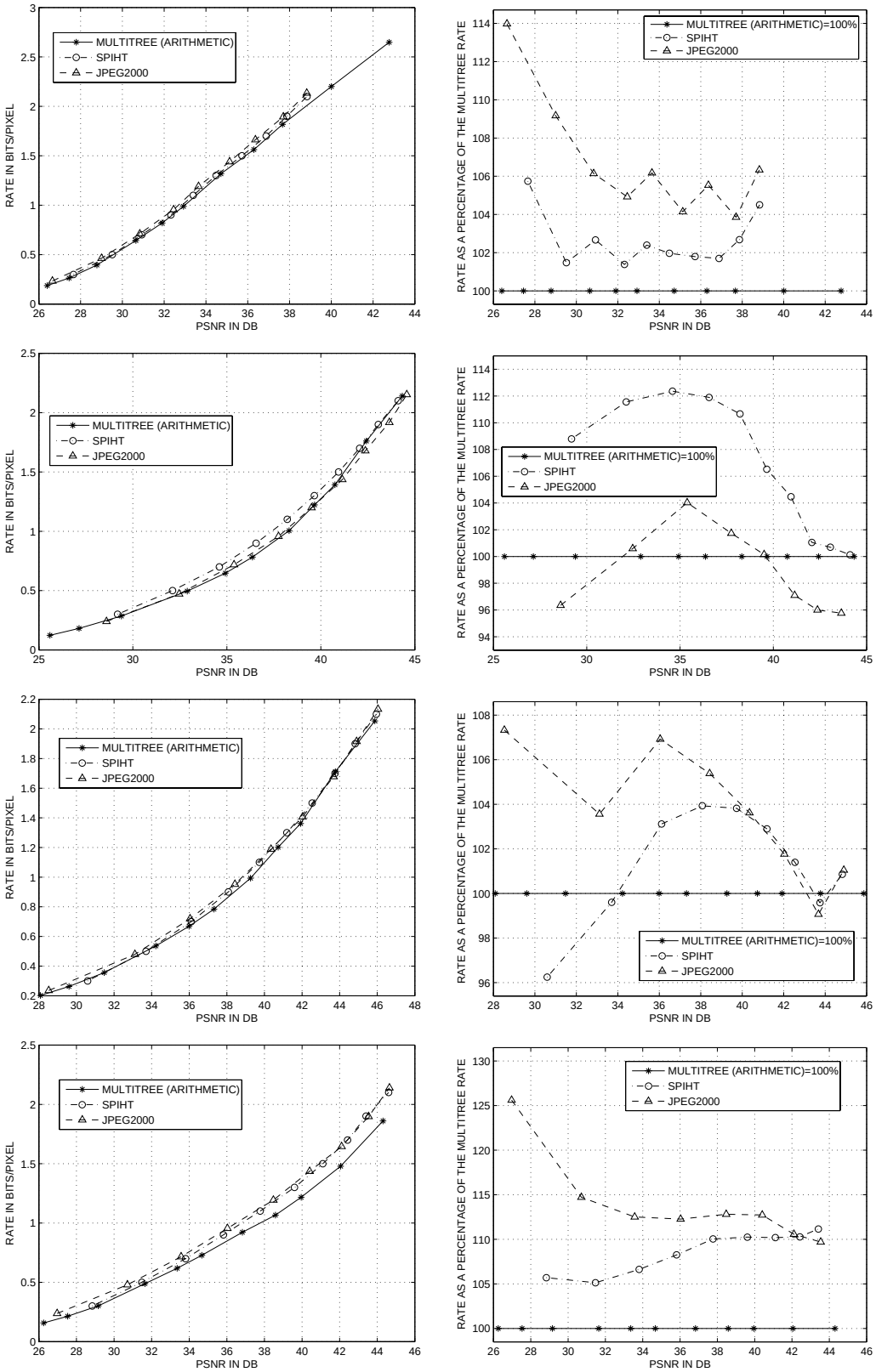



Fig. 12. Rate-distortion curves for “goldhill” (top row), “barbara” (second row), “lenna” (third row), and “cameraman” (bottom row). The right column shows bit rates as percentages of the bit rate for the multitree algorithm with arithmetic coding of the coefficients.

```
(C*_a, s*_a) ← best_split(a) {
    if C*_a has been computed
        get C*_a and s*_a from the global data structure TABLE;
    else {
        s*_a ← ∅;            //Initialize s*_a
        C*_a ← c(a);         //Initialize C*_a
        for α ∈ R(a) {
            for b ∈ α
                (C*_b, s*_b) ← best_split(b);
            if c̄(a → α) + Σ_{b∈α} C*_b < C*_a {
                s*_a ← α;
                C*_a ← c̄(a → α) + Σ_{b∈α} C*_b;
            }
        }
        record C*_a and s*_a in the global data structure TABLE;
    }
    return C*_a and s*_a;
}
```

(a) Recursive calculation of best splits and costs.

```
t*_a ← best_tree(a) {
    get s*_a from the global data structure TABLE;
    if s*_a is the empty set
        t*_a ← [a];
    else {
        i ← 0;
        for b ∈ s*_a {
            i ← i + 1;
            b_i ← b;
            t*_{b_i} ← best_tree(b_i);
        }
        t*_a ← [ a
                 t*_{b_1}  ...  t*_{b_i} ];
    }
    return t*_a;
}
```

(b) Recursive generation of best tree.

Fig. 13. Pseudocode for the recursive calculation of the best splits and best costs, and for the recursive generation of the globally optimal tree.

all symbols which can be descendants of $a$. We let $S_{A(a)}$ be the set of all allowed splits of elements of $A(a)$. For each symbol $b \in A(a)$, there is exactly one recursive call to the subroutine best_split of Fig. 13(a). During this call, the costs of all possible splits of $b$ are compared. The number of such comparisons is $|S_b|$. Therefore, the overall time complexity of the algorithm is $O(|S_{A(a)}|)$. In applications where only the yield of a tree is of interest, such as our rectangular tiling example of Section II, there is some redundancy associated with searching over multiple trees which have the same yield. In some instances, such as in [16], [17], this redundancy is very significant and may be eliminated, leading to a lower time complexity.

The overall space complexity is $O(|A(a)|)$ since we need to store two numbers—the best cost and the best split—for each symbol in $A(a)$. The key to controlling the time and space complexity is therefore keeping the sizes of the sets $S_{A(a)}$ and $A(a)$ low. In addition, as we have remarked before, the computation of the costs $\bar{c}(a \to \alpha)$ and $c(a)$ could actually dominate the time complexity of the overall algorithm, and therefore another important guideline to a successful application of our algorithm is to use tractable cost functions.

We note that the dynamic programming algorithm of Section IV-B is easily generalized to the problem of finding the optimal tree in each of a sequence of multitree dictionaries, provided that the overall cost has additive structure, as in Eq. (10).

## VII. Relationships with Prior Work.

It can be easily shown that standard wavelet packet and dyadic local cosine dictionaries [8], [9], as well as anisotropic 2-D wavelet packet dictionaries [2], [11], are all multitree dictionaries. It is also easy to see that a specialization of our algorithm of Fig. 13 to the wavelet packets and dyadic local cosines is essentially a restatement of the best basis algorithm of [8], [9], its specialization to anisotropic wavelet and cosine packets is a restatement as the anisotropic best basis algorithm of [2], [11], and its specialization to dyadic tiling is a restatement of the dyadic CART algorithm of [11].

Our algorithm can also be used for a variety of other dictionaries, such as, for example, any dictionary of block or lapped bases in two or more dimensions. It is interesting to point out that arbitrary block and lapped dictionaries in 1-D can be efficiently searched without exploiting their tree structure, but rather using standard dynamic programming techniques, as was shown in [16], [17].

It was pointed out in [11] that there is a close relationship between the best basis algorithm of [8], [9] and pruning methods used in the design of classification and regression trees [4]. These methods have also been used for vector quantization and other applications [6]. These and other methods such as, for example, [3], [18], [22], [32], [41], [42], [44], [47], [49]–[51], seek to optimally tile a multidimensional domain with dyadic hyperrectangles. Our multitree algorithm can potentially be applied to these problems, allowing one to lift the requirement that the split locations be dyadic.

We now point out a close relationship between our algorithm and procedures for estimating the maximum a posteriori probability parse of a string [1], [21], [28] or an image [34]–[36], [43]. In these problems, $-\bar{c}(u \to \alpha)$ of Eq. (12) stands for the log-probability of the production $u \to \alpha$, and the probability of a tree $t$ is defined as the product of the probabilities of all the productions in $t$. The objective of these estimation tasks is to find the most probable tree, i.e., to minimize with respect to $t$ the negative-log-probability of the tree $t$, $\sum_{u \to \alpha \in t} \bar{c}(u \to \alpha)$. But this is exactly what our algorithm of Fig. 13 does. Thus, the estimation algorithms of [1], [21], [28], [34]–[36], [43] represent special cases of our search algorithm for the best tree in a multitree dictionary.

## VIII. Conclusions.

We presented a general framework of multitree dictionaries and provided a recursive algorithm for finding the best representation in a multitree dictionary. We illustrated our framework and algorithm within the contexts of optimal rectangular

and wedgelet tilings and image compression, and designed a new block image coder. The key property that enables our algorithm to be fast for any additive or multiplicative cost is the fact that, while the number of possible trees can be enormous, the number of possible symbols at tree nodes is typically manageable. By storing the optimal cost and the optimal set of children for each symbol in a global data structure, the algorithm only needs to make one recursive call per symbol.

In the future we plan to further explore the flexibility of our framework and design various other multitree dictionaries which allow a fast selection of the best representation in applications such as time-frequency analysis, approximation, embedded image compression, video compression, vector quantization, and classification.

## APPENDIX

## THE TOTAL NUMBER OF TILINGS AND TIME COMPLEXITY FOR MULTITREE VS QUADTREE DICTIONARIES.

Suppose we have a $2^L \times 2^L$ square image, with $N = 2^{2L}$ pixels. It is shown in Section II that for such an image, our algorithm finds the optimal rectangular tiling in a multitree dictionary with arbitrary split locations, in $O(N^{2.5})$ time. The search algorithm of [8], [9] which finds the best quadtree tiling, is linear in $N$. The optimal multitree search in this case is therefore a factor of $O(N^{1.5})$ slower than the optimal quadtree search. On the other hand, we now show that the ratio of the number of multitree tilings and the number of quadtree tilings is exponential in $N$.

**Proposition 1.** *Let $\mu_N$ and $\kappa_N$ be the total number of multitree and quadtree rectangular tilings, respectively, for a $2^L \times 2^L$ square image with $N = 2^{2L}$ pixels. Assume that the multitree tilings allow arbitrary splits of a rectangle into two other rectangles, and quadtree tilings may only partition each square into four congruent squares. Then, for $N \geq 64$, we have:*

$$\frac{\mu_N}{\kappa_N} > 1.5^N. \tag{13}$$

*Proof.* A careful enumeration for $L = 2$ (i.e., a $4 \times 4$ image) yields $\mu_{16} = 68480$. For $L = 3$, note that an $8 \times 8$ square can be partitioned into four $4 \times 4$ squares, and thus one way of tiling an $8 \times 8$ square is to tile each $4 \times 4$ square independently. Therefore, $\mu_{64} > \mu_{16}^4$. Proceeding by induction on $L$, we see that, for $N = 2^{2L}$ with $L \geq 3$,

$$\mu_N > \mu_{16}^{4^{L-2}} = 68480^{4^L/16} = \left(68480^{1/16}\right)^{4^L} > 2^{4^L} = 2^N.$$

On the other hand, it is shown in Proposition 8.5 of [25] that

$$\kappa_N < 2^{\frac{49}{48}4^{L-1}},$$

which is smaller than $1.2^N$. Therefore,

$$\frac{\mu_N}{\kappa_N} > \left(\frac{2}{1.2}\right)^N > 1.5^N,$$

proving our assertion. □

## REFERENCES


[1] J. Baker. Trainable grammars for speech recognition. In D. Klatt and J. Wolf, editors, *Speech Communications Papers for the 97th Meeting of the Acoustical Society of America*, pages 547–550. June 1979.

[2] N. N. Bennett. Fast algorithm for best anisotropic walsh bases and relatives. *J. of Appl. and Comput. Harmonic Analysis*, 8(1):86–103, January 2000.

[3] G. Blanchard, C. Schäfer, and Y. Rozenholc. Oracle bounds and exact algorithm for dyadic classification trees. In *Proc. 17th. Conf. on Learning Theory*, volume 3120 of *Springer Lecture Notes in Artificial Intelligence*, pages 378–392, 2004.

[4] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Chapman & Hall, New York, 1984.

[5] H. Cheng and C. A. Bouman. Document compression using rate-distortion optimized segmentation. *Journal of Electronic Imaging*, 10(2):460–474, April 2001.

[6] P. A. Chou, T. Lookabaugh, and R. M. Gray. Optimal pruning with applications to tree-structured source coding and modeling. *IEEE Trans. Inf. Th.*, 35(2):299–315, March 1989.

[7] I. Cohen, S. Raz, and D. Malah. Orthonormal shift-invariant adaptive local trigonometric decomposition. *Sig. Proc.*, 57(1):43–64, February 1997.

[8] R. R. Coifman, Y. Meyer, and M. V. Wickerhauser. Wavelet analysis and signal processing. In M. B. Ruskai *et al.*, editor, *Wavelets and Their Applications*, pages 153–178. Jones and Bartlett, Boston, 1992.

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

[10] R.R. Coifman and Y. Meyer. Remarques sur l'analyse de Fourier à fenêtre. *C. R. Acad. Sci. Paris Sér. I Math.*, 312(3):259–261, 1991.

[11] D. L. Donoho. CART and best-ortho-basis: A connection. *Ann. Stat.*, 25(5):1870–1911, October 1997.

[12] D. L. Donoho. Wedgelets: Nearly minimax estimation of edges. *Ann. Statist.*, 27(3):859–897, June 1999.

[13] D. L. Donoho and I. M. Johnstone. Ideal denoising in an orthonormal basis chosen from a library of bases. *Comptes Rendus Acad. Sci., Ser. I*, 319:1317–1322, 1994.

[14] 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. Sig. Proc.*, 41(12):3341–3359, December 1993.

[15] C. Herley, Z. Xiong, K. Ramchandran, and M. T. Orchard. Joint space-frequency segmentation using balanced wavelet packet tree for least-cost image representation. *IEEE Trans. Im. Proc.*, 6(9):1213–1230, September 1997.

[16] Y. Huang, I. Pollak, C. A. Bouman, and M. N. Do. Best basis search in lapped dictionaries. Technical Report TR-ECE-04-08, School of ECE, Purdue University, West Lafayette, IN 47907, December 2004. To appear in IEEE Trans. Sig. Proc. `www.ece.purdue.edu/~ipollak/local_cosines.pdf`.

[17] Y. Huang, I. Pollak, C. A. Bouman, and M. N. Do. New algorithms for best local cosine basis search. In *Proc. ICASSP-2004*, Montreal, Quebec, May 17-21 2004.

[18] R. M. Figueras i Ventura, L. Granai, and P. Vandergheynst. R-D analysis of adaptive edge representations. In *Proc. IEEE Workshop MMSP*, pages 130–133, December 2002.

[19] H. Krim and J.-C. Pesquet. On the statistics of best bases criteria. In A. Antoniadis, editor, *Wavelets and statistics*, Lecture Notes in Statistics, pages 193–207. Springer-Verlag, 1995.

[20] H. Krim, D. Tucker, S. Mallat, and D. Donoho. On denoising and best signal representation. *IEEE Trans. Inf. Th.*, 45(7):2225–2238, November 1999.

[21] K. Lari and S. Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4(1):35–56, January 1990.

[22] R. Leonardi and M. Kunt. Adaptive split-and-merge for image analysis and coding. In *Proc. SPIE*, volume 594, pages 2–9, December 1985.

[23] M. Lindberg. *Two-Dimensional Adaptive Haar-Walsh Tilings*. Licentiat Thesis in Applied Mathematics, Åbo Akademi University, Åbo, Finland, October 1999.

[24] M. Lindberg and L. F. Villemoes. Image compression with adaptive Haar-Walsh tilings. In *Wavelet Applications in Signal and Image Processing VIII, Proc. SPIE 4119*, 2000.

[25] S. G. Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, second edition, 1999.

[26] H. Malvar. Lapped transforms for efficient transform/subband coding. *IEEE Trans. ASSP*, 38(6):969–978, June 1990.

[27] H. Malvar. *Signal Processing with Lapped Transforms*. Artech House, 1992.
[28] C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.
[29] P. Moulin. Signal estimation using adapted tree-structured bases and the MDL principle. In *Proc. IEEE-SP Int. Symp. TFTS*, pages 141–143, Paris, June 1996.
[30] U. Ndili. A coding theoretic approach to image segmentation. Master's thesis, Rice University, Houston, Texas, April 2001.
[31] U. Ndili, R. D. Nowak, and M. A. T. Figueiredo. Coding theoretic approach to image segmentation. In *Proc. ICIP-2001*, Thessaloniki, Greece, October 2001.
[32] E. Le Pennec and S. G. Mallat. Sparse geometric image representations with bandelets. *IEEE Trans. Im. Proc.*, 14(4):423–438, April 2005.
[33] J.-C. Pesquet, H. Krim, D. Leporini, and E. Hamman. Bayesian approach to best basis selection. In *Proc. ICASSP-96*, pages 2634–2638, Atlanta, USA, May 1996.
[34] I. Pollak, J. M. Siskind, M. P. Harper, and C. A. Bouman. Modeling and estimation of spatial random trees with application to image classification. In *Proc. ICASSP*, Hong Kong, April 2003.
[35] I. Pollak, J. M. Siskind, M. P. Harper, and C. A. Bouman. Parameter estimation for spatial random trees using the EM algorithm. In *Proc. ICIP*, Barcelona, September 2003.
[36] I. Pollak, J. M. Siskind, M. P. Harper, and C. A. Bouman. Spatial random trees and the center-surround algorithm. Technical Report TR-ECE-03-03, Purdue University, School of ECE, January 2003. `www.ece.purdue.edu/~ipollak/it03.pdf`.
[37] K. Ramchandran and M. Vetterli. Best wavelet packet bases in a rate-distortion sense. *IEEE Trans. Im. Proc.*, 2(2):160–175, Apr. 1993.
[38] J. H. Rothweiler. Polyphase quadrature filters—a new subband coding technique. In *Proc. ICASSP-83*, pages 1280–1283, Boston, MA, March 1983.
[39] A. Said and W. A. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Trans. Circ. Syst. Vid. Tech.*, 6(3):243–250, June 1996.
[40] G. M. Schuster and A. K. Katsaggelos. A video compression scheme with optimal bit allocation between displacement vector field and displaced frame difference. In *Proc. ICASSP-96*, pages 1967–1970, Atlanta, GA, May 1996.
[41] C. Scott and R. D. Nowak. Minimax-optimal classification with dyadic decision trees. Technical Report TREE0403, Rice University, 2004. `www.stat.rice.edu/~cscott/pubs.html`.
[42] R. Shukla, P. L. Dragotti, M. N. Do, and M. Vetterli. Rate-distortion optimized tree structured compression algorithms for piecewise smooth images. *IEEE Trans. Im. Proc.*, 14(3):March, 343–359 2005.
[43] J. M. Siskind, J. Sherman, I. Pollak, M. P. Harper, and C. A. Bouman. Spatial random tree grammars for modeling hierarchal structure in images. Preprint, May 2004. `www.ece.purdue.edu/~ipollak/draft2004_5_25.pdf`.
[44] G. J. Sullivan and R. L. Baker. Efficient quadtree coding of images and video. *IEEE Trans. Im. Proc.*, 3(3):327–331, May 1994.
[45] D. Taubman. High performance scalable image compression with EBCOT. *IEEE Trans. Im. Proc.*, 9(7):1158–1170, July 2000.
[46] C. M. Thiele and L. F. Villemoes. A fast algorithm for adapted time-frequency tilings. *J. of Appl. and Comput. Harmonic Analysis*, 3:91–99, 1996.
[47] J. Vaisey and A. Gersho. Image compression with variable block size segmentation. *IEEE Trans. Sig. Proc.*, 40(8):2040–2060, August 1992.
[48] L. F. Villemoes. Adapted bases of time-frequency local cosines. Preprint, June 1999, `www.math.kth.se/old-home-pages/larsv/publ.html`.
[49] M. B. Wakin, J. K. Romberg, H. Choi, and R. G. Baraniuk. Rate-distortion optimized image compression using wedgelets. In *Proceedings of ICIP-2002*, Rochester, New York, September 2002.
[50] M. Wien. Variable block-size transforms for H.264/AVC. *IEEE Trans. Ckts. Syst. Vid. Tech.*, 13(7):604–613, July 2003.
[51] R. M. Willett and R. D. Nowak. Platelets: a multiscale approach for recovering edges and surfaces in photon-limited medical imaging. *IEEE Trans. Medical Imaging*, 22(3):332–350, March 2003.
[52] Z. Xiong, K. Ramchandran, C. Herley, and M. T. Orchard. Flexible tree-structured signal expansions using time-varying wavelet packets. *IEEE Trans. Sig. Proc.*, 45(2):333–345, February 1997.
[53] D. Xu and M. N. Do. Anisotropic 2-D wavelet packets and rectangular tiling: theory and algorithms. In *Proc. SPIE Conf. on Wavelet Appl. in Sig. and Im. Proc. X*, San Diego, Aug. 2003.

PLACE PHOTO HERE

**Dr. Yan Huang** received the B.Eng. degree in electronic engineering from Tsinghua University, Beijing, China, in 2000, and Ph.D. in electrical engineering from Purdue University in 2004. She is currently a post-doctoral researcher at the School of Electrical and Computer Engineering, Purdue University. Her research interests are in image and signal processing.

PLACE PHOTO HERE

**Dr. Ilya Pollak** received the B.S. and M.Eng. degrees in 1995 and Ph.D. in 1999, all from M.I.T., all in electrical engineering. In 1999-2000, he was a post-doctoral researcher at the Division of Applied Mathematics, Brown University. Since 2000, he has been Assistant Professor of Electrical and Computer Engineering at Purdue University. He has held short-term visiting positions at the Institut National de Recherche en Informatique et en Automatique in Sophia Antipolis, France, and at Tampere University of Technology, Finland. In 2001, he received a CAREER award from the National Science Foundation. He is the Chair of the Signal Processing Chapter of the Central Indiana Section of the IEEE. His research interests are in image and signal processing, specifically, hierarchical statistical models, fast estimation algorithms, nonlinear scale-spaces, and adaptive representations.

PLACE PHOTO HERE

**Dr. Minh N. Do** was born in Thanh Hoa, Vietnam, in 1974. He received the B.Eng. degree in computer engineering from the University of Canberra, Australia, in 1997, and the Dr.Sci. degree in communication systems from the Swiss Federal Institute of Technology Lausanne (EPFL), Switzerland, in 2001.

Since 2002, he has been an Assistant Professor with the Department of Electrical and Computer Engineering and a Research Assistant Professor with the Coordinated Science Laboratory and the Beckman Institute, University of Illinois at Urbana-Champaign. His research interests include wavelets, image and multidimensional signal processing, multiscale geometric analysis, and visual information representation.

Dr. Do received a Silver Medal from the 32nd International Mathematical Olympiad in 1991, a University Medal from the University of Canberra in 1997, the best doctoral thesis award from the Swiss Federal Institute of Technology Lausanne in 2001, and a CAREER award from the National Science Foundation in 2003.

PLACE PHOTO HERE

**Dr. Charles A. Bouman** received a B.S.E.E. degree from the University of Pennsylvania in 1981 and a MS degree from the University of California at Berkeley in 1982. From 1982 to 1985, he was a full staff member at MIT Lincoln Laboratory, and in 1989 he received a Ph.D. in electrical engineering from Princeton University. In 1989, he joined the faculty of Purdue University where he holds the rank of Professor with a primary appointment in the School of Electrical and Computer Engineering and a secondary appointment in the School of Biomedical Engineering.

Professor Bouman's research focuses on the use of statistical image models, multiscale techniques, and fast algorithms in applications including medical and electronic imaging. Professor Bouman is a Fellow of the IEEE, a Fellow of the American Institute for Medical and Biological Engineering (AIMBE), a Fellow of the society for Imaging Science and Technology (IS&T), a member of the SPIE professional societies, a recipient of IS&T's Raymond C. Bowman Award for outstanding contributions to digital imaging education and research, and a University Faculty Scholar of Purdue University. He is currently the general Co-Chair of the SPIE/IS&T Symposium on Electronic Imaging, secretary of the IEEE Biomedical Image and Signal Processing Technical Committee, and a member of the Steering Committee for the IEEE Transactions on Medical Imaging. He has been an associate editor for the IEEE Transactions on Image Processing and the IEEE Transactions on Pattern Analysis and Machine Intelligence. He has also been the Awards Chair for the ICIP 1998 organizing committee, Co-Chair of the SPIE/IS&T conferences on Visual Communications and Image Processing 2000 (VCIP), and a member of the IEEE Image and Multidimensional Signal Processing Technical Committee, a Vice President of Publications and a member of the Board of Directors for the IS&T Society, and he is the founder and co-chair of the SPIE/IS&T conference on Computational Imaging.