USC–SIPI REPORT #263

Fast Tree-Structured Nearest Neighbor Encoding for Vector Quantization

by

Ioannis Katsavounidis, C.-C. Jay Kuo and Zhen Zhang

June 1994

Signal and Image Processing Institute
UNIVERSITY OF SOUTHERN CALIFORNIA
Department of Electrical Engineering-Systems
3740 McClintock Avenue, Room 404
Los Angeles, CA 90089-2564 U.S.A.

Fast Tree-Structured Nearest Neighbor Encoding for Vector Quantization*

Ioannis Katsavounidis† and C.-C. Jay Kuo † and Z. Zhang‡

June 30, 1994; EDICS: IP 1.1

Abstract

The nearest neighbor encoding problem with an unstructured codebook of an arbitrary size and vector dimension is examined in this research. We propose a new tree-structured nearest neighbor encoding method that significantly reduces the complexity of the full search method without any performance degradation in terms of distortion. Our method consists of efficient algorithms for constructing a binary tree for the codebook and the nearest neighbor encoding by using this tree. Numerical experiments are given to demonstrate the performance of the proposed method.

1 Introduction

Nearest neighbor (NN) encoding [5] is the problem of finding the nearest point for an unknown input point from a set of fixed points, called the codebook in vector quantization (VQ), in a k-dimensional vector space. Its fundamental role in the VQ area is indicated by the fact that NN encoding is a synonym for vector quantizing. Moreover, it constitutes the major computational task of the generalized Lloyd algorithm (GLA), which is the algorithm used in VQ codebook design. It is also used extensively in pattern classification and decision-making problems. A straight-forward solution to this problem is the full search method which involves an exhaustive search of the distances for all available points; in this way, the complexity of encoding one point with full search is proportional to the dimension of the vector space and the size of the codebook.

For the $1-D$ vector space (i.e. the scalar case), we can use a so-called binary search to accelerate the NN encoding due to the linear structure of real numbers. We can thus reduce the complexity of

*This work was supported by the National Science Foundation Presidential Faculty Fellow (PFF) Award ASC-9350309.

†The authors are with the Signal and Image Processing Institute and the Department of Electrical Engineering-Systems, University of Southern California, Los Angeles, CA 90089-2564. E-mail: katsavou@sipi.usc.edu and cckuo@sipi.usc.edu.

‡The author is with the Communication Science Institute and the Department of Electrical Engineering-Systems, University of Southern California, Los Angeles, CA 90089-2565. E-mail: zzhang@chief.usc.edu.

encoding from $O(N)$ to $O(\log N)$, by performing a simple preprocessing step which involves sorting the N scalars that form the codebook, followed by an arrangement of these values in a balanced binary tree. We should note, however, that part of the computational complexity is carried over to the preprocessing stage. Fortunately, sorting N scalars can be achieved in $O(N \log N)$ operations, which justifies the selection of binary search over full (exhaustive) search, even for a relatively small number of input points.

Even though vectors in a higher dimensional space cannot be ordered without lossing their field structure [13], there have been attempts to generalize the notion of binary search in the vector space. They include tree-structured VQ [5], binary hyperplane testing ([2]), $K-d$ trees [1], [5], [8] and the projection method [3]. There have been suggested other methods to achieve an improvement over the computation required by full search by combining a number of ideas centered around the so-called partial distance (or distortion) calculation and triangle inequality (or *anchor points*) based method. For details about these fast NN-encoding algorithms, we refer to [4], [10], [11], [12], [15], [16].

Naturally, it has been the tree structure that received the most attention in such an effort. Most researchers have associated the tree-structured VQ with the notion of sacrificing some performance in distortion to achieve a better speed. Since most tree-structured VQ schemes require a parallel design of both the codebook and the tree to be used for search, the additional constraint imposed on the codebook design leads naturally to performance degradation in terms of higher average distortion per input vector. We may say that the tree-structured VQ does not formally tackle the NN-encoding problem. Instead, it formulates and solves an approximating problem. Another feature of most tree-structured VQ schemes is that the codebook is designed in parallel with the tree structure. Thus, they are not suitable for codebooks of an arbitrary size.

To overcome this, there have been suggested methods such as the projection method or the $K-d$ tree method that operate on an arbitrary input codebook. Most of these methods require the use of Monte-Carlo type techniques for the design of the search tree and therefore a very large amount of computation. Such a computational cost can only be justified for static VQ schemes, where the codebook is fixed. For adaptive VQ, where the codebook is updated dynamically, they are difficult to apply. Furthermore, the storage requirements of these algorithms impose restrictions not only on the codebook size and vector dimension but also on the degree of improvement (speed-up factor) that can be achieved over the execution time of full search. Another interesting method to build a binary search tree with a given unstructured codebook was the binary hyperplane testing proposed by Cheng [2]. It requires a great deal of computation due to the Monte-Carlo approach,

and is proved to be efficient only when $N >> 2^k$, where the corresponding output bit-rate is more than 1 bit per vector component. This clearly prohibits its practical use in most image or video compression problems. To conclude, the major disadvantage of these methods is that the design of the search tree is too expensive computationally together with the fact that they are limited to a relatively small vector dimension, usually in the range between 2 and 8.

The partial distance calculation method appears to be promising, since it adds no memory overhead and requires no extra computation. Even though the performance improvement provided is relatively low in comparison with the other methods discussed above, it is a better choice for adaptive VQ with continuous adaptation of the codebook. This is the reason why its simplest version is adopted in our approach as described in Section 3. Finally, the triangle inequality based methods may be suitable for some specific applications by providing a trade-off between the preprocessing time, memory overhead and NN-encoding time. Nevertheless, their performance improvement is just on the average, and they have only been tested for small vector dimensions.

In this research, we focus on a fast tree-structured nearest neighbor encoder which significantly reduces the complexity of the full search method without any performance degradation in terms of distortion. Both the construction of a binary tree-structured codebook and the nearest neighbor encoding with such a tree-structured codebook are efficiently handled by using our method. The execution time of these two tasks is such that it allows the application of the suggested method to slowly adaptive VQ schemes, i.e. schemes where the adaptation takes place after a number of vectors have been coded. One such case is the GLA, where there is a very large number of training vectors that are vector quantized for every iteration, while the update of the codebook only takes place once after every iteration.

This work is organized as follows. After introducing some basic geometrical properties in Section 2, we present a new method where a binary search tree with respect to an arbitrary codebook can be obtained and the nearest neighbor of an input vector can be located effectively in Section 3. The performance of the proposed method is reported in Section 4.

2 Basic Geometrical Properties

Let C be a subset of the k-dimensional Euclidean space R^k with cardinality $|C| = N$. We denote the elements of C by c_i, $i \in \{1, \ldots, N\}$, and the individual components of the vectors by c_i^j, $j \in \{1, \ldots, k\}$. The nearest neighbor (NN) encoding problem can be stated as follows. For an

arbitrary vector $x \in R^k$, find the vector c_i such that

$$\forall c_j \in C, \quad j \neq i \;\Rightarrow\; d(x, c_j) \geq d(x, c_i).$$

where $d(x, y)$ can be any distance function between two vectors in R^k. For simplicity, we focus on the Euclidean distance in this work. Besides the distance between two points, we are going to use the distance from a point to a set defined as

$$d(x, A) = \inf_{y \in A} d(x, y).$$

As usual, we use $\langle x, y \rangle$ to denote the inner product of two vectors. It can be easily seen that the NN encoding problem has a simple brute force solution which involves the calculation of N distances $d_i = d(x, c_i)$ for $1 \leq i \leq N$ and $(N-1)$ comparisons. We are interested in a more efficient method to solve the NN encoding problem. To provide an insight into the new method, we may start our discussion with the case $N = 2$. Consider $x, c_1, c_2 \in R^k$ with $c_1 \neq c_2$. Without loss of generality, we assume $d(x, c_1) < d(x, c_2)$. Thus, we have

$$\sum_{j=1}^{k} (x^j - c_1^j)^2 < \sum_{j=1}^{k} (x^j - c_2^j)^2.$$

It is easy to simplify the above inequality to get

$$\sum_{j=1}^{k} x^j (c_2^j - c_1^j) + \frac{1}{2} \sum_{j=1}^{k} \left[(c_1^j)^2 - (c_2^j)^2 \right] < 0.$$

By defining

$$w^0 = \frac{1}{2} \sum_{j=1}^{k} \left[(c_1^j)^2 - (c_2^j)^2 \right], \quad \text{and} \quad w^j = c_2^j - c_1^j, \;\; 1 \leq j \leq N,$$

we get the following encoding rule:

$$d(x, c_1) < d(x, c_2) \Longleftrightarrow \langle x, w \rangle + w^0 < 0 \tag{1}$$

where $w = (w^1, \ldots, w^N)^T$. By augmenting vectors x and w with one additional component, we can define vectors of dimension $N + 1$:

$$\tilde{x} = (x^0, x^1, x^2, \ldots, x^N)^T, \;\; \text{where } x^0 = 1, \quad \text{and} \quad \tilde{w} = (w^0, w^1, w^2, \ldots, w^N)^T.$$

Then, the encoding rule (1) can be rewritten as

$$d(x, c_1) < d(x, c_2) \Longleftrightarrow \langle \tilde{x}, \tilde{w} \rangle < 0. \tag{2}$$

Thus, the problem of finding the closest distance between an arbitrary input vector x and two given vectors x_1 and x_2 involves the calculation of just one inner product instead of two. In fact, the $k+1$ dimensional vector $\tilde{w}$ defines a hyperplane in R^k, i.e.

$$H(\tilde{w}) = \{x \in R^k : \langle \tilde{x}, \tilde{w} \rangle = 0\},$$

which partitions the space R^k into two regions, where $d(x, c_1) < d(x, c_2)$ in one and $d(x, c_1) > d(x, c_2)$ in the other. The above derivation is a well known fact and has been used in many decision problems.

Furthermore, the minimum distance between an arbitrary vector x and the hyperplane $H(\tilde{w})$ can be characterized by the following theorem.

Theorem 1 *Let c_1, c_2, $\tilde{x}$ and $\tilde{w}$ be given as above. Then, we have*

$$d(x, H(\tilde{w})) = \frac{|\langle \tilde{x}, \tilde{w} \rangle|}{d(c_1, c_2)}.$$

Proof: To find the minimum distance from a point x to the hyperplane $H(\tilde{w})$ is a constrained optimization problem. That is, we try to minimize

$$d^2(x, y) = \sum_{j=1}^{k} (x^j - y^j)^2$$

under the constraint that $y \in H(\tilde{w})$. Since $y \in H(\tilde{w})$, we have

$$\langle y, H(\tilde{w}) \rangle = \langle y, w \rangle + w^0 = \sum_{j=1}^{k} y^j (c_2^j - c_1^j) + \frac{1}{2} \sum_{j=1}^{k} \left[(c_1^j)^2 - (c_2^j)^2 \right] = 0. \tag{3}$$

By using the method of Lagrange multipliers, we can construct the functional

$$\phi(y) = d^2(x, y) + \lambda \langle y, H(\tilde{w}) \rangle,$$

and require that all partial derivatives of $\phi(y)$ with respect to y_j, $1 \le j \le k$ to be zero. This leads to a set of equations

$$2(y^j - x^j) + \lambda (c_2^j - c_1^j) = 0, \quad j = 1, \ldots, k. \tag{4}$$

We can solve the system of equations (3) and (4) for λ, i.e.

$$\lambda = \frac{\sum_{j=1}^{k} x^j (c_2^j - c_1^j) + \frac{1}{2} \sum_{j=1}^{k} \left[(c_1^j)^2 - (c_2^j)^2 \right]}{\frac{1}{2} \sum_{j=1}^{k} (c_2^j - c_1^j)^2} = \frac{2\langle x, w \rangle}{d^2(c_2, c_1)}. \tag{5}$$

Thus, the distance from x to the hyperplane $H(\tilde{w})$ is given by

$$d^2(x, H(\tilde{w})) = d^2(x, y) = \sum_{j=1}^{k} (x^j - y^j)^2. \tag{6}$$

By using (4) in (6), we obtain

$$d^2(x, H(\tilde{w}) = \frac{\lambda^2}{4} \sum_{j=1}^{k} (c_2^j - c_1^j)^2 = \frac{\lambda^2}{4} d^2(c_2, c_1) = \frac{\langle x, w \rangle^2}{d^2(c_2, c_1)},$$

where the last equality is due to (5). Thus, the theorem is proved. □.

Motivated by Theorem 1, we define the *signed distance* from a point x to a hyperplane $H(\tilde{w})$ as

$$d_s(x, H(\tilde{w})) = \frac{\langle x, w \rangle}{d(c_1, c_2)}.$$

Note that the signed distance has the same magnitude as that of the regular distance, and takes the minus (or plus) sign if x is closer to c_1 (or c_2) as indicated in (1).

Next, we present another theorem that will be needed in the tree search algorithm described in Section 3.2.

Theorem 2 *Let $x, y \in R^k$ and $H(\tilde{w})$ defined as above. Then, we have*

$$d(x, y) \geq |d_s(x, H(\tilde{w})) - d_s(y, H(\tilde{w}))|.$$

Proof: By definition, we have

$$d_s(x, H(\tilde{w})) = \frac{|\langle x, w \rangle|}{d(c_1, c_2)} \quad \text{and} \quad d_s(y, H(\tilde{w})) = \frac{|\langle y, w \rangle|}{d(c_1, c_2)},$$

so that

$$d_s(x, H(\tilde{w})) - d_s(y, H(\tilde{w})) = \frac{|\langle x, w \rangle| - |\langle y, w \rangle|}{d(c_1, c_2)} = \frac{\sum_{j=1}^{k} x^j w^j - \sum_{j=1}^{k} y^j w^j}{d(c_1, c_2)}.$$

By squaring the above expression, we obtain

$$[d_s(x, H(\tilde{w})) - d_s(y, H(\tilde{w})]^2 = \frac{[\sum_{j=1}^{k} (x^j - y^j) w^j]^2}{d^2(c_1, c_2)} = \frac{[\sum_{j=1}^{k} (x^j - y^j)(c_2^j - c_1^j)]^2}{d^2(c_1, c_2)}. \tag{7}$$

By applying Schwartz's inequality to (7), we get

$$[d_s(x, H(\tilde{w})) - d_s(y, H(\tilde{w}))]^2 \leq \frac{\sum_{j=1}^{k} (x^j - y^j)^2 \sum_{j=1}^{k} (c_2^j - c_1^j)^2}{d^2(c_1, c_2)} = \frac{d^2(x, y) d^2(c_1, c_2)}{d^2(c_1, c_2)} = d^2(x, y),$$

and the proof is completed. □.

3 Fast Nearest Neighbor Encoding

3.1 Binary Tree-Structured Codebook Design

The procedure of designing a codebook of size N from a set of training vectors by the generalized Lloyd algorithm (GLA), also referred to as the k-means algorithm after MacQueen [9] or the LBG algorithm after [7], can be stated as follows.

1. Given a codebook $C_m = \{y_i; i = 1, \ldots, N\}$ obtained from the mth iteration, find the optimal partitioning of the space R^k. That is, use the nearest neighbor condition to form the nearest neighbor cells $R_i = \{x : d(x, y_i) < d(x, y_j); j \neq i\}$.
2. Use the centroid condition to update the codebook $C_{m+1} = \{\text{centroid}(R_i); i = 1, \ldots, N\}$, which is the optimal reproduction codebook based on the cells found on (1).

The GLA is an iterative optimization procedure based on the method of alternating projections and, therefore, leads to a local minimum. Most researchers have applied the GLA in codebook design. In this work, we use it as the main tool to build a binary tree-structured codebook from a given unstructured codebook. It is important to note that Step 1 in GLA is nothing but NN-encoding of all the training vectors with respect to the existing codebook. It is for this reason that GLA is probably the perfect testbed experiment for all fast NN-encoding methods. It is also clear from Step 2 that the codebook is changing at every iteration. Thus, in order to apply a fast NN-encoding algorithm to the GLA, the preprocessing time (i.e. the execution time needed by the algorithm to set up all necessary data structures for a given codebook) should be small and definitely not larger than the execution time of full search.

Our tree construction algorithm is stated below.

Algorithm 1: Tree Construction Algorithm

1. (Initialization) Store the entire codebook at the root of a binary tree.
2. With the codebook stored at the node, run the GLA to obtain the two centroids c_1 and c_2 that best represent it, determine the hyperplane $H(\tilde{w})$ that separates the two Voronoi cells and calculate the signed distances between all the codewords and the hyperplane $d_s(v, H(\tilde{w}))$. Create two child nodes and store the indices of the codewords that have negative distance from the hyperplane at the left node and those with positive distance at the right node.
3. We repeat the process in Step 2 until each leaf contains only one codeword.

It is evident that the above algorithm leads to a binary tree that may be balanced or unbalanced, depending on the input codebook. Our experiments have shown that even though the resulting tree is almost always unbalanced, it does not have a high degree of non-uniformity. A balanced binary tree of N items has a depth of $\log_2 N + 1$ while the averaged depth with our algorithm is approximately $\log_2 N + 3$. Thus, time required to search this binary tree is still low. Since the tree is dependent on the input codebook, the computational complexity is random and we can only consider the averaged performance. Moreover, each step involves a call to the GLA which is an iterative optimization procedure with unknown number of iterations so that the problem of determining the complexity becomes even more difficult. Fortunately, the GLA for the case of $N = 2$, requires significantly less iterations to converge than any other case. With conjunction to the maximum distance initialization technique [6], we observed that no more than 5 iterations are necessary for convergence. Thus, the computational complexity of the algorithm can be estimated as $O(kN \log_2 N)$.

3.2 Nearest Neighbor Encoding with Tree-structured Codebook

The *partial distance search* method [5] provides an effective means to find the closest codeword for a given input vector x in a codebook. With this method, we proceed sequentially through the codebook by keeping the index of the nearest neighbor up to that point together with its distortion (in terms of distance squared) from the input vector. To test the next codeword c_n, we compute the partial distance $D_r = \sum_{j=1}^{r} (x^j - c_n^j)^2$ with $r = 1$. If $D_r \geq D$, we reject this vector and go to the next code vector. Otherwise, we increase the value of r by one and perform the test again. This process continuous until we reach $r = k$. Then, we set $D = D_k$ and move to the next codeword. It has been reported that the partial search algorithm can typically reduce the search time of full search by a factor of 4 without any additional cost.

We are now ready to present our nearest neighbor encoding based on the tree-structured codebook constructed in Section 3.1. For a given input vector x, we do the following.

Algorithm 2: Tree Search Algorithm

1. *Perform a greedy binary search from the root to the leaf.*
   Compute the signed distance of the input vector x to the hyperplane $H(\tilde{w})$ associated with the node. If the signed distance $d_s(x, H(\tilde{w})) \leq 0$, we move to the left node, otherwise, to the right. The same process repeats until a leaf with only one codeword c_i is reached. The candidate codeword c_i is reported to the parent node. We move from the leaf node to its

parent node.

2. *Search the best candidate codeword from the leaf to the root.*

   At a given node, we search all codewords associated with the node that have a signed distance from the hyperplane with an opposite sign to that of $d_s(x, H(\tilde{w}))$. Apply the partial distance calculation method to the set of codewords that satisfy $|d_s(x, H(\tilde{w})) - d_s(c, H(\tilde{w}))| < d(x, c_i)$. By doing so, we can find the best candidate codeword with respect to this node. Then, we move to its parent node and repeat the process until the root is reached.

The validity of the above algorithm just presented is obvious. The nearest neighbor is chosen via two elimination processes. First, since we have $d(x, c) \geq |d_s(x, H(\tilde{w})) - d_s(c, H(\tilde{w}))|$ from Theorem 2, if a codeword c satisfying the relation $|d_s(x, H(\tilde{w})) - d_s(c, H(\tilde{w}))| \geq d(x, c_i)$, we can eliminate c as a candidate for the nearest neighbor codeword. The second elimination process comes from the partial distance search method that is self evident. Note that the encoding algorithm can be implemented very efficiently by having the signed distances of the codewords pre-calculated, ordered and stored at each node when the binary tree is constructed in Algorithm 1.

4 Experimental Results and Discussion

4.1 Experimental Results

To test our method, we used the "basic" VQ scheme, i.e. the simplest VQ scheme applied to image blocks where each component of the vectors is the intensity of an image pixel. It is however important to point out that our method can be applied to any VQ scheme such as predictive VQ, adaptive VQ, transform VQ, etc.

We first used the GLA to obtain a number of codebooks of various dimensions (k) and sizes (N). The training vectors were obtained from the "Baboon", "Lenna" and "Boat" images, taken from the USC image database, which are all grey-level images of size 512×512 (in pixels) with 8 bits per pixel (256 grey levels). We partitioned these images into blocks of dimension 2×2, 4×4 and 8×8, and generated training vectors of dimensions $k = 4$, 16 and 64, correspondingly. In the codebook design, we adopted the maximum distance method [6] for its initialization, performed a number of GLA iterations, and stopped the iteration when there was that no significant reduction in the MSE (Mean Square Error) values. We chose codebook sizes of the form $N = 2^i$ with integer i in the experiments, although neither the initialization technique nor the GLA imposes this restriction. The resulting codebook sizes are up to 2048, 8192 and 32768 for the cases of 8×8, 4×4 and 2×2, respectively.

After obtaining various codebooks as stated above, we ran the tree construction algorithm (i.e. Algorithm 1) to build a binary tree for each codebook. The execution time, which is the actual CPU time on a Hewlett-Packard Apollo Series 400 Workstation, versus the codebook size for various vector dimensions is plotted in Fig. 1. One can verify the $O(kN \log_2 N)$ behavior as claimed in Section 3.1. Moreover, it is important to point out that the execution time for the largest experiment we ran, i.e. the case of block size 2×2 and codebook size $N = 32768 = 2^{15}$, required approximately 5 minute CPU time to build the binary search tree. This is significantly less than the time required by existing algorithms that build a binary tree from an arbitrary codebook using Monte-Carlo techniques by (at least) an order of magnitude.

Next, we test the tree-search algorithm (i.e. Algorithm 2) by encoding three different test images: "Couple", "Elaine" and "Heather" of size 512×512. In what follows, we maintain the following notation:

- u: The input (original) image;
- $\tilde{u}$: The output (quantized) image;
- p_i (empirical distribution) $= \frac{\text{no. of vectors classified to codeword } c_i}{\text{total no. of input vectors}}$.

For every experiment, we reported the following quantities:

1. The resulting distortion which is calculated as
$$MSE = \frac{1}{512^2} \sum_{i=1}^{512} \sum_{j=1}^{512} [u(i,j) - \tilde{u}(i,j)]^2,$$
2. The entropy of the encoded image (in bits per pixel) based on the empirical distribution of the code vectors,
$$H = -\frac{1}{k} \sum_{i=1}^{N} p_i log_2(p_i)$$
where N is the codebook size and k is the vector dimension,
3. The CPU execution time of the full search and the new tree search algorithms in terms of seconds.

The execution times obtained for the images "Couple", "Elaine" and "Heather" are given, respectively, in Figs. 2, 3 and 4 as functions of the codebook size N parameterized by three different vector dimensions. Based on these execution times, we list the speed-up factor, which is defined as

Table 1: Speed-up factors for the three images.

N		64	128	256	512	1024	2048	4096	8192	16384	32768
Couple	2×2	3.09	5.02	8.40	12.82	23.83	35.54	58.64	78.09	95.59	130.47
	4×4	4.28	6.67	10.41	14.54	21.35	26.77	31.05	35.56		
	8×8	4.66	5.86	7.64	10.20	13.06	15.64				
Elaine	2×2	3.18	5.03	8.31	13.49	23.77	36.58	60.48	83.75	108.59	169.78
	4×4	5.00	7.82	12.46	17.78	27.38	34.51	39.68	43.56		
	8×8	6.04	8.25	12.30	16.86	23.53	29.57				
Heather	2×2	3.55	5.86	10.11	16.08	30.23	46.89	77.32	115.48	135.74	207.22
	4×4	5.80	9.04	14.82	23.68	39.43	57.40	82.03	77.79		
	8×8	7.65	10.55	15.82	26.59	34.57	48.63				

the ratio of the CPU time with full search and the proposed tree search algorithm, in Table 1 for various vector dimensions and codebook sizes. We see a similar general behavior from this table.

By analyzing the experimental results more carefully, we observe that the execution time of the new method tends to depend more on the entropy H of the output than on the codebook size N. Figs. 5-7 present the encoding time as a function of the entropy (expressed in bits per pixel) for the image "Elaine" with 2×2, 4×4 and 8×8 blocks by using our algorithm and full search. The resemblance between these three figures is apparent. It is even more interesting that the corresponding plots for the other two images (not shown here) are also similar. The shape of the graphs motivates us to model the logarithm of the execution time linearly with respect to the entropy. By performing a simple linear regression, we can model the execution time as a function of entropy as

$$T = C\alpha^{Hk}$$

where the parameters C and α are listed in Table 2. One can see clearly from the table what is the gain provided by our new algorithm. While the encoding complexity of full search grows as a function $O(\alpha_1^{Hk})$, the proposed algorithm grows as $O(\alpha_2^{Hk})$, where the ratio of α_1 and α_2 is in the range $1.8 \leq \frac{\alpha_1}{\alpha_2} \leq 3$.

4.2 Discussion

By plotting the execution time versus the entropy, we get a much more accurate picture of the performance of our algorithm. The entropy figure is theoretically and practically achievable by proper loseless source coding. It is important to note that the codebook size N is an internal design parameter of the vector quantizer, since it is not present at the communication channel

Table 2: Parameters C and α for the three images.

	Full search						New method					
	Couple		Elaine		Heather		Couple		Elaine		Heather	
dim.	C	α	C	α	C	α	C	α	C	α	C	α
2×2	0.43	2.64	0.44	2.58	0.93	2.97	2.17	1.46	2.18	1.41	2.83	1.47
4×4	0.28	3.26	0.28	3.20	0.36	6.24	0.78	1.82	0.58	1.96	1.04	2.01
8×8	0.27	3.44	0.17	3.82	0.30	7.57	0.48	2.04	0.40	2.05	0.65	2.40

where only the indices of the codewords are transmitted. It is however related to the entropy and the distortion obtained for a quantized image. Generally speaking, the larger the codebook size is, the larger the entropy and the smaller the distortion.

It is known that different images behave differently when one applies a VQ scheme with the same codebook. There is the notion of *match* between the codebook and an input image, which can also be viewed as a measure of similarity between the images used to generate the training sequence for the GLA and the image used for encoding. This can be clearly seen on Fig. 8, where we plotted the entropy of the output image (measured in bits per pixel) versus the codebook size for the three different block sizes. Each plot shows the entropy obtained for the three different images and their relationship to the theoretical vector quantizer resolution defined as $r = \frac{\log_2 N}{k}$. For example, it appears that the images "Elaine" and "Couple" are better matched to the codebooks than the image "Heather". Thus, on one hand, it may look strange that the complexity of the full search algorithm depends entirely on the codebook size for a fixed vector dimension as observed in Figs. 2-4. On the other hand, our algorithm follows much closer the entropy of the encoded sequence. The quantized "Elaine" or "Couple" images have higher entropy values than the quantized "Heather" image. It is for this reason that the image "Heather" appears to achieve a higher speed-up factor than "Elaine" or "Couple". We can say that data that are closer to be uniformly distributed with respect to the codebook and therefore have more entropy require more time to encode than other types of data with less entropy with the same encoding parameters N and k.

To further illustrate the relationship of the execution time with the entropy, we choose the "Elaine" image and plot the so-called *empirical rate-distortion function* in Fig. 9 by putting together all the points (H, MSE) obtained for different codebook sizes and vector dimensions. Rigorously speaking, it is the convex hull of these points that defines the empirical rate-distortion function. This plot provides very useful information about the source and the relationship between vector dimension and codebook size. It gives a strong supporting evidence to the known fact that very large

codebook sizes together with very large vector dimensions are necessary to achieve an (H,MSE) point arbitrarily close to the theoretical rate-distortion function of a source, which may be however impractical to achieve in most cases. For example, we can achieve approximately $MSE = 50$ with vector dimension $k = 16$ and codebook size $N = 8192$ while we cannot do better than $MSE = 300$ with vector dimension $k = 4$ and codebook size $N = 4$, even though both configurations result in an effective bit rate of about $H = 0.5$ bits/pixel. Consequently, the design of codebooks of larger sizes and dimensions under a fixed entropy requirement is of practical importance. Since our proposed method performs increasingly faster than full search as the codebook size increases as shown in Table 1, the new method will provide a clear advantage in satisfying such a need.

5 Conclusions and Extensions

In this work, we proposed a tree construction algorithm and a tree search algorithm to achieve fast nearest neighbor encoding. The superior performance of the proposed method was demonstrated by a set of numerical experiments.

We would like to mention some related research work to be carried out in the future. The computational bottleneck of our encoding algorithm is the partial distance search elimination of the codewords. To reduce the complexity of this part, we may incorporate other ideas such as anchor points and k-d trees. One very important application of this new fast NN encoding algorithm is the speed-up of the generalized Lloyd algorithm, since the computational bottleneck of the GLA is the nearest neighbor encoding. In the GLA context, since at every iteration the codebook is updated, we have to perform both the tree creation algorithm (to structure the codebook) and the tree search algorithm to classify the training vectors. Thus, we have to consider the sum of the execution time for both tree creation and search. As shown earlier, both algorithms are much faster than full search. Thus, the incorporation of the new NN encoding in the GLA will significantly accelerate the running time of GLA. Also, we believe that there is much more into the relationship between entropy, distortion, computational complexity and memory capacity than what this work is able to address. The first such relationship was the rate-distortion function studied by Shannon [14] who actually assumed infinite complexity and memory capacity. A more complete analysis, that provides the *operational rate-distortion* function, i.e. the achievable rate-distortion pair under some constraints of bounded computational complexity and/or memory capacity should be of great interest.

References

[1] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Comm. of the ACM*, pp. 209–226, Sept. 1975.

[2] D. Y. Cheng and A. Gersho, "A fast codebook search algorithm for nearest-neighbor pattern matching," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, vol. 1, (Tokyo), pp. 265–268, Apr. 1986.

[3] D. Y. Cheng, A. Gersho, B. Ramamurthi, and Y. Shoham, "Fast search algorithms for vector quantization and pattern matching," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, (San Diego), pp. 9.11.1–9.11.4, Mar. 1984.

[4] J. H. Friedman, F. Baskett, and L. Shustek, "An algorithm for finding nearest neighbors," *IEEE Trans. on Computers*, Vol. C-24(10), pp. 1000–1006, Oct. 1975.

[5] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Publishers, 1992.

[6] I. Katsavounidis, C.-C. J. Kuo, and Z. Zhang, "A new initialization technique providing accelerated convergence of the generalized Lloyd algorithm." submitted to *IEEE Signal Processing Letters*, Apr. 1994.

[7] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. on Communications*, Vol. COM-28, pp. 84–95, Jan. 1980.

[8] A. Lowry, S. Hossain, and W. Millar, "Binary search trees for vector quantization," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, (Dallas), pp. 2205–2208, 1987.

[9] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. of the Fifth Berkeley Symposium on Math. Stat. and Prob.*, vol. 1, pp. 281–297, 1967.

[10] K. Motoishi and T. Misumi, "On a fast vector quantization algorithm," in *Proceedings of the VIIth Symp. on Inform. Theory and its Applications*, 1984.

[11] M. Orchard, "A fast nearest neighbor search algorithm," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, (Toronto, Canada), pp. 2297–2300, May 1991.

[12] V. Ramasubramanian and K. Paliwal, "An efficient algorithm for fast nearest-neighbor search based on a spherical distance coordinate formulation," in *Proc. European Signal Processing Conf.*, (Barcelona), Sept. 1990.

[13] W. Rudin, *Principles of Mathematical Analysis*, International Series In Pure and Applied Mathematics, McGraw-Hill, third ed., 1976.

[14] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, No. 27, pp. 379–423,623–656, 1948.

[15] M. R. Soleymani and S. D. Morgera, "An efficient nearest neighbor search method," *IEEE Trans. on Communications*, Vol. COM-35, pp. 677–679, 1987.

[16] E. Vidal, "An algorithm for finding nearest neighbors in (approximately) constant average time complexity," *Pattern Recognition Letters*, Vol. 4, pp. 145–147, 1986.

Figure Captions

Figure 1: The plot of tree creation time versus codebook size.

Figure 2: Performance comparison for the "Couple" image.

Figure 3: Performance comparison for the "Elaine" image.

Figure 4: Performance comparison for the "Heather" image.

Figure 5: Performance comparison for the "Elaine" image with block size 2×2.

Figure 6: Performance comparison for the "Elaine" image with block size 4×4.

Figure 7: Performance comparison for the "Elaine" image with block size 8×8.

Figure 8: Entropy versus codebook size for the various vector dimensions (dash-dotted line: theoretical VQ resolution, solid line: "Elaine", dotted line: "Couple", dashed line: "Heather").

Figure 9: Empirical rate-distortion function for the "Elaine" image.

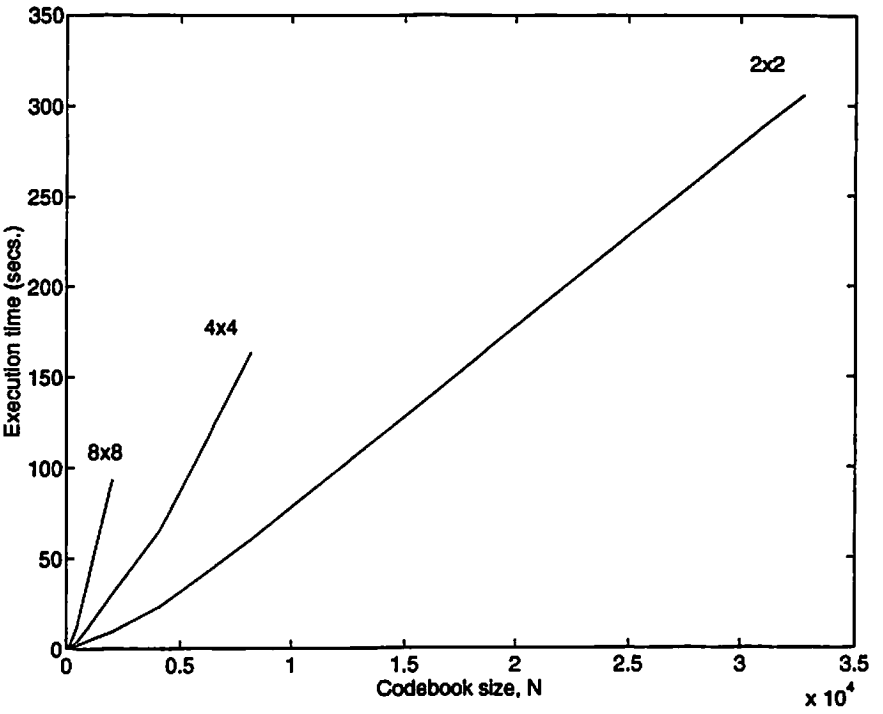

Figure 1: The plot of tree creation time versus codebook size.

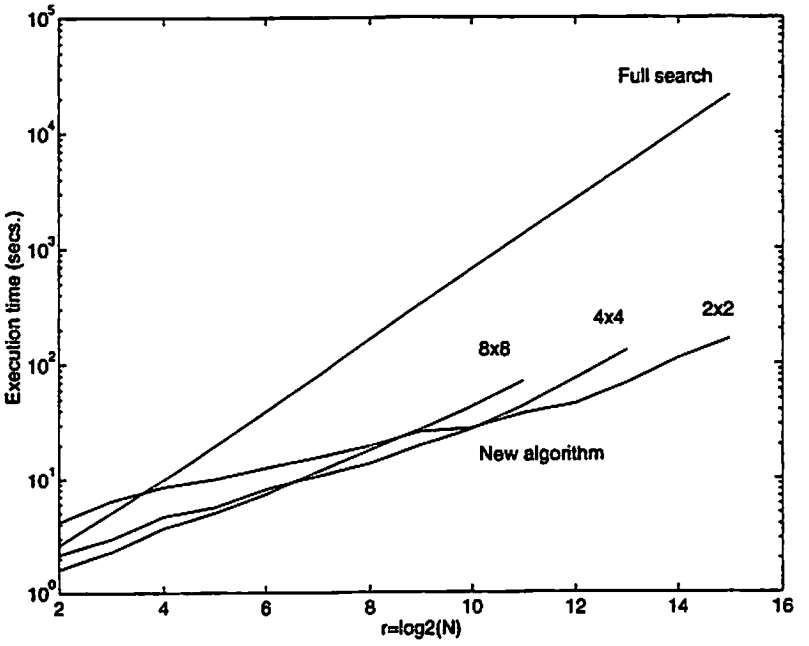

Figure 2: Performance comparison for the "Couple" image.

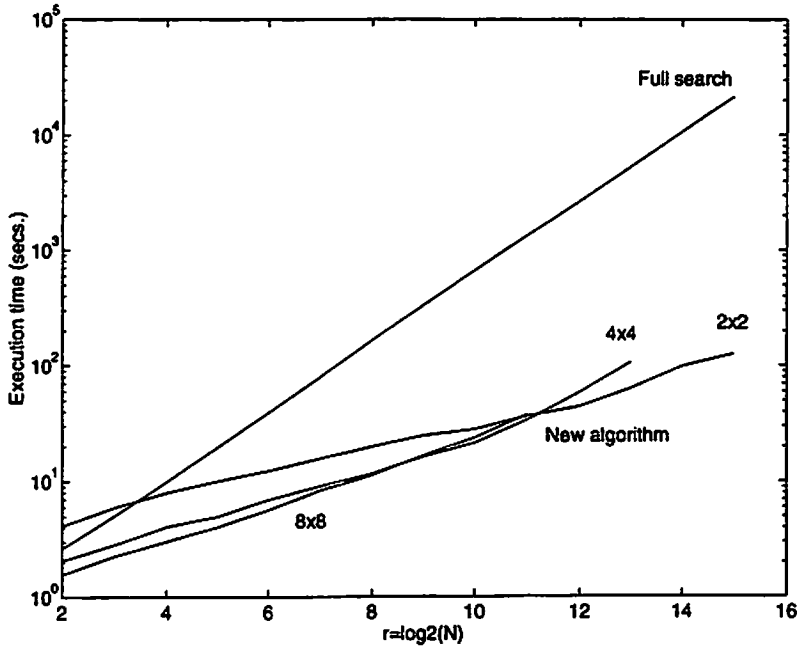

Figure 3: Performance comparison for the "Elaine" image.

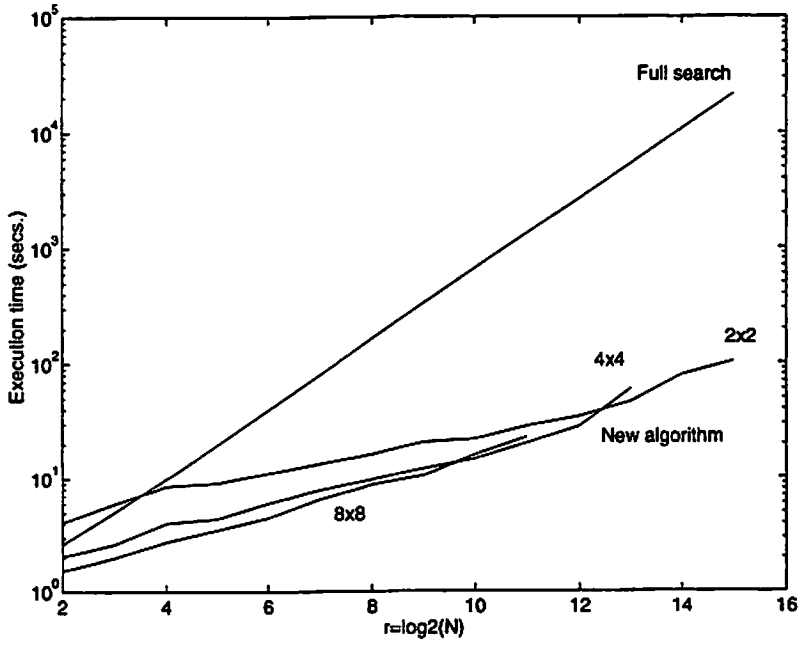

Figure 4: Performance comparison for the "Heather" image.

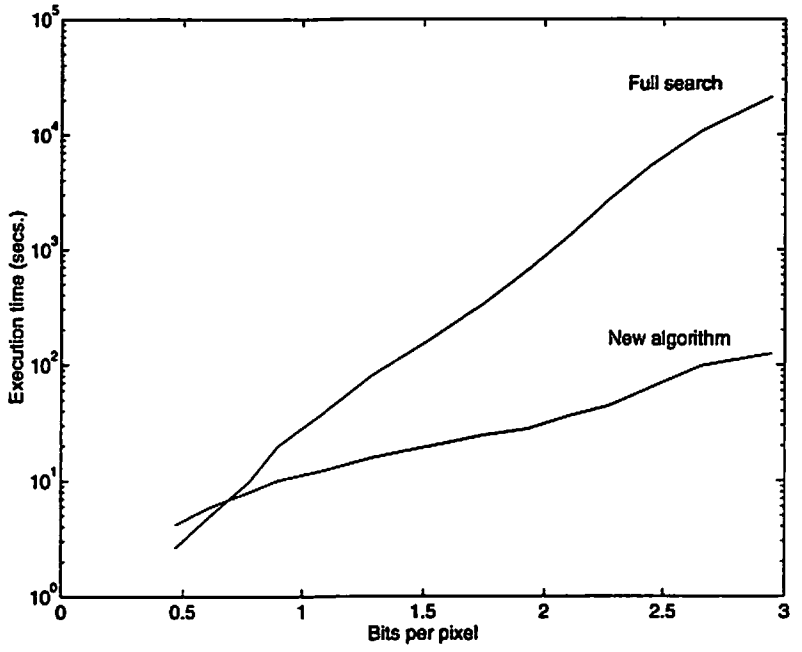

Figure 5: Performance comparison for the "Elaine" image with block size 2×2.

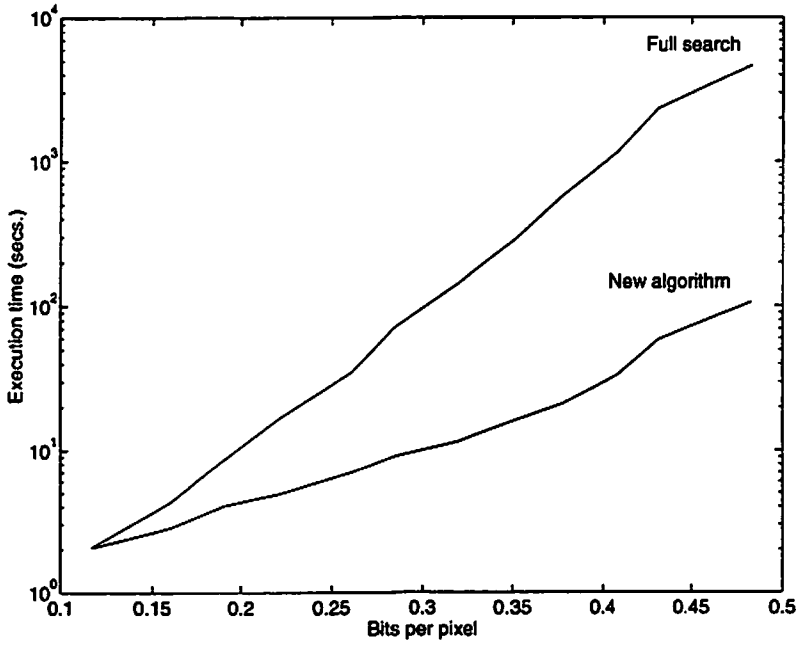

Figure 6: Performance comparison for the "Elaine" image with block size 4×4.

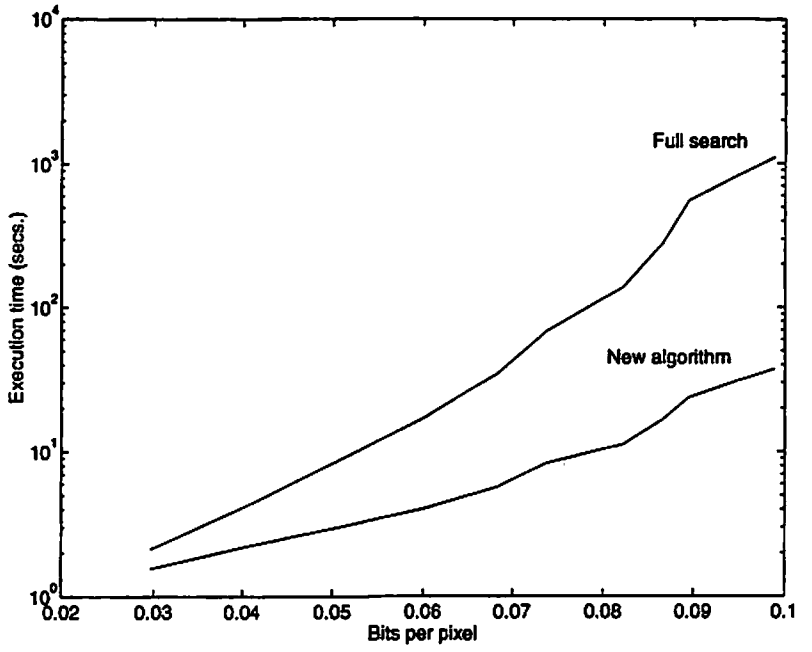

Figure 7: Performance comparison for the "Elaine" image with block size 8×8.

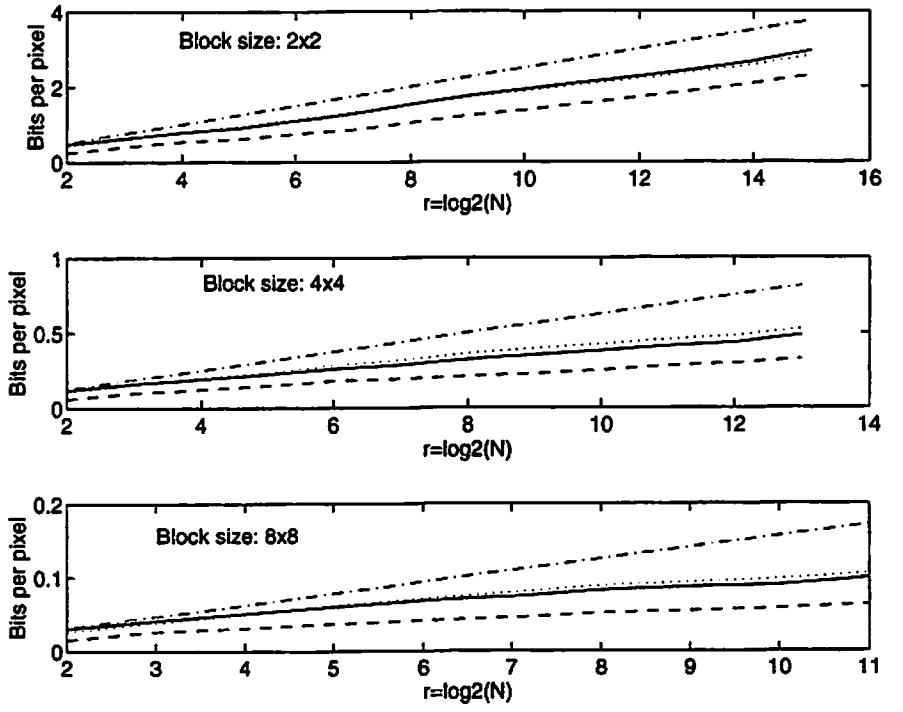

Figure 8: Entropy versus codebook size for the various vector dimensions (dash-dotted line: theoretical VQ resolution, solid line: "Elaine", dotted line: "Couple", dashed line: "Heather").

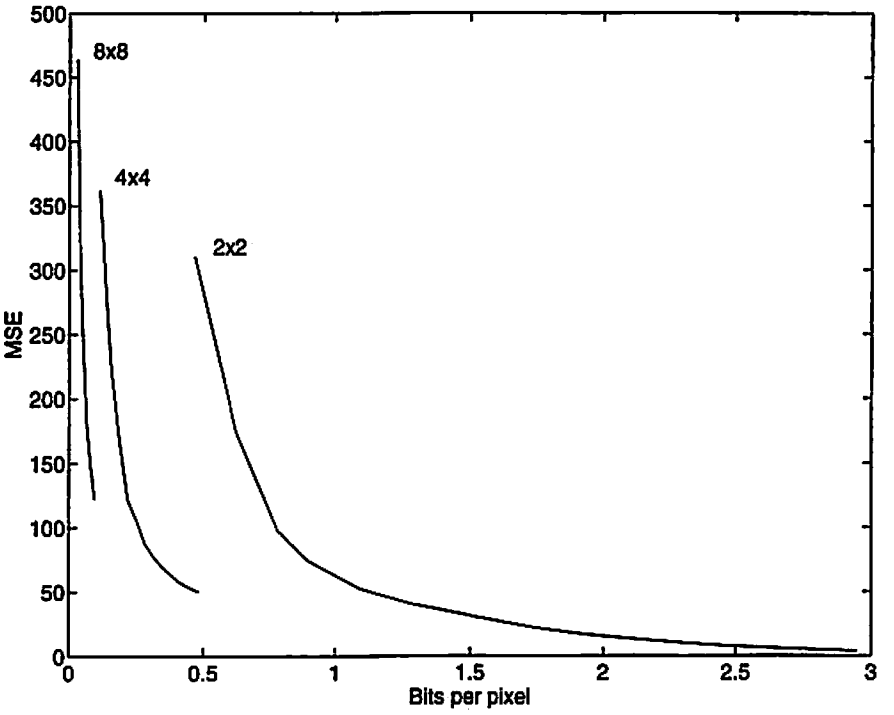

Figure 9: Empirical rate-distortion function for the "Elaine" image.