In Proc. of Visual Communic. and Image Proc., VCIP '97, San Jose, CA, Feb. 1997

DCT Computation with Minimal Average Number of Operations

Krisda Lengwehasatit and Antonio Ortega

Integrated Media Systems Center
Department of Electrical Engineering-Systems
University of Southern California
Los Angeles, California 90089-2564
Phone: (213) 740-4679, Fax: (213) 740-4679
E-mail: lengweha@sipi.usc.edu and ortega@sipi.usc.edu

ABSTRACT

The Discrete Cosine Transform (DCT) is widely used in all transform-based image and video compression standards due to its well-known decorrelation and energy compaction properties for typical images. Many fast algorithms available for the DCT optimize various parameters such as additions and multiplications but they are input independent and thus require the same number of operations for any inputs. In this paper we study the benefits of input-dependent algorithms for the DCT which are aimed at minimizing the average computation time by taking advantage of the sparseness of the input data. Here, we concentrate on the inverse DCT (IDCT) part since typical input blocks will contain a substantial number of zeros. We show how to construct an IDCT algorithm based on the statistics of the input data, which are used to optimize the algorithm for the average case. We show how, for a given input and a correct model of the complexity of the various operations, we can achieve the fastest average performance.

Keywords: Discrete Cosine Transform, minimum operation, knowledge-based, distortion/complexity trade-off, average complexity, JPEG.

1 INTRODUCTION

The Discrete Cosine Transform (DCT) is by far the most popular transform used for image compression applications. Reasons for its popularity include not only its good performance in terms of energy compaction for typical images but also the availability of several fast algorithms. Aside from the theoretical justifications of the DCT (as approximation to the Karhunen Loeve Transform, KLT, for certain images [1]) our interest stems from the wide utilization in different kinds of image and video coding applications. While we concentrate on the DCT, most of our developments are directly applicable to other orthogonal transforms. The well-known JPEG and MPEG standards use DCT as their transformation to decorrelate input signal (see [2]). Even with the emergence of wavelet transforms, DCT has still retained its position in image compression and there has recently been some active research into ways of reducing the complexity of image manipulation, for example scaling [3], in the DCT domain. These methods heavily rely on the sparseness of the DCT coefficients to improve the efficiency of the manipulations. An example of the sparseness of quantized coefficients can be seen in Fig. 1 where histograms of quantized coefficients in 8x8 block at 2 different image qualities are shown. Note that, as expected, the high

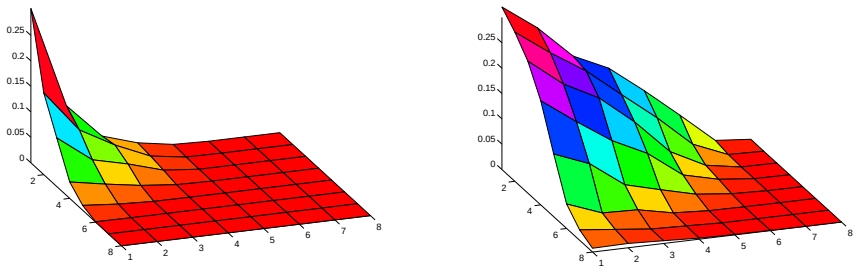

Figure 1: Frequency of nonzero occurrence of 8x8 quantized DCT coefficients of lenna with MSE 60.21 (a) and MSE 14.79 (b).

frequency coefficients are very likely to be zeros. This will also be the case for the difference images encountered in typical video coding scenarios (e.g. P and B frames in MPEG), where the percentage of zero coefficients is likely to be even higher.

One common factor in all the fast algorithms proposed for hardware implementation, see for example [4], is that they aim at reducing the complexity of a generic direct or inverse DCT, *regardless of the input to be transformed*. Therefore complexity is estimated by the number of operations which is the same for every input. In this work we consider as possible operations not only typical additions/multiplications but also other types of computer instructions (for example `if, then, else`). Image coding applications require a 2-D DCT and the number of operations can be further reduced if vertical and horizontal transforms are considered simultaneously. However, for simplicity, in most cases a separable implementation is preferred. There is a theoretical proof of the minimum number of multiplications needed to do 2-D DCT based on 1-D algorithm [5] but again this analysis is input-independent. The goal of this paper is to demonstrate how additional reductions in complexity are possible, in an average sense, if the statistical characteristics of the inputs are considered.

We are motivated by observing that recently general purpose workstations and PCs have increased their speed to a level where performing compression/decompression in software of images and even video, is efficient. Examples of this trend include software-only decoders for the H.263 videoconferencing standard, as well as the wide use of software implementations of the JPEG standard to exchange images over the Internet. This trend is likely to continue as faster hardware becomes available and innovative uses of software, for example usage of JAVA applets, become widespread. In this latter example, the flexibility afforded by using an interpreted language comes at the price of a considerable reduction in performance, so that slowdown factors of about an order of magnitude are not unheard of. In these circumstances optimizing the performance of the algorithms for the specific case of software operation is all the more important.

To the best of our knowledge, only one published work, by Froitzheim and Wolf [6] (FW), formally addresses the problem of minimizing the IDCT complexity in an input dependent manner, by taking advantage of the sparseness of transformed quantized DCT coefficients at the decoder end. However numerous software implementations of IDCT available in the public domain do take into account the sparseness to achieve image decoding speed-up, and it is thus safe to assume that this basic idea is quite widely used. The FW algorithm takes advantage of the sparseness of quantized DCT coefficients by checking for all-zero rows and columns in the block to be decoded, since these sets do not require a transform to be performed. The trade-off comes from the fact that additional logic is required to detect the all-zero rows and columns and so the performance of the worst case decoding is worse than if tests were not performed. However the speed up for the numerous blocks having many zeros more than makes up for the difference and on average, this simple scheme achieves faster decoding for "average" images. Similar ideas have been used in numerous software implementations such as the JPEG implementation by the Independent JPEG Group[7], the MPEG implementation by U.C.Berkeley[8] or vic, the UCB/LBL Internet video

conferencing tool [9]. However, FW and other methods currently being used are somewhat ad-hoc and seem to have been chosen through trial and error for typical images. In this paper, we present *a systematic procedure to obtain the fastest, in the average sense, inverse DCT algorithm for a given image or set of images.*

The paper is organized as follows. Section 2 contains a brief review of the DCT concept. Section 3 provides the intuition of our algorithm along with a detailed description of our probabilistic approach. Implementation of our proposed algorithm and some preliminary results are shown in Section 4 where we also address the issue of modelling the relative complexity of the various kinds of operations in a software environment.

2 DISCRETE COSINE TRANSFORM

The N point DCT $\bar{X}$ of vector input $\bar{x} = [x(0), x(1), ..., x(N-1)]^T$ is defined as $\bar{X} = S\bar{x}$ where S is the transformation matrix with elements $s_{i,j}$

$$s_{i,j} = \frac{c_i}{2} \cdot \cos\frac{(2j+1)i\pi}{2N} \tag{1}$$

$$\text{where} \qquad c_i = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } i = 0 \\ 1 & \text{for } i > 0 \end{cases}$$

for 2-D transform

$$X = S \cdot x \cdot S^T \tag{2}$$

where X and x are now 2-D matrix.

Each basis in the DCT domain represents an equivalent frequency component of the spatial domain data sequence. The last equation above shows the separability of 2-D DCT i.e. we can apply DCT along the row/column separately from each other. After applying the DCT to a typical image, DCT coefficients in low frequency region contain most of the energy. Therefore, DCT has a good energy compaction performance.

As mentioned previously, one can easily obtain $\bar{X}$ from the DFT coefficients of $\bar{x}$ and thus numerous mechanisms to compute the DCT rely on fast algorithms for the DFT (see also [1] for details). Alternative methods, such as the one depicted in Fig. 2 (see [4]), allow further reduction of the DCT computation complexity. Further speed-ups are possible by using algorithms which operate on integers only rather than with floating-point operations, see [10]. These algorithms are useful in particular for situations where DCT is used for compression since the loss in precision in the DCT calculation does not affect the quantized coefficients which are used to represent the images.

3 AVERAGE COMPLEXITY MINIMIZATION

In this work our goal is to minimize the average complexity of the IDCT by taking advantage of sparseness of the quantized transformed coefficients (see Fig. 1). The basic idea is to have different modes operation according to the type of input so that inputs that are "easier" can have their IDCT computed in less time. The first step is thus to classify all the possible inputs. While many different classification methods are possible in this paper we choose a simple one based on the occurrence of zeros in the input vector.

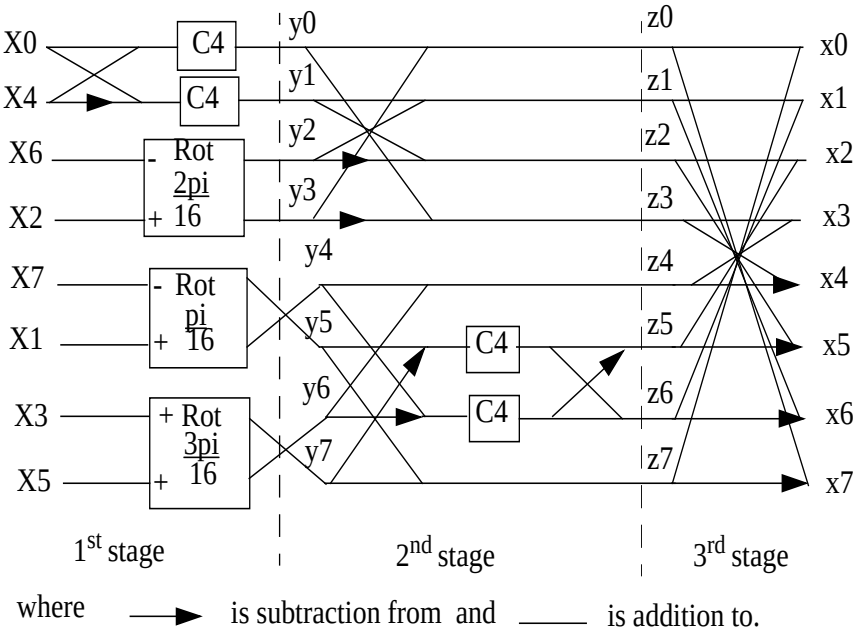

Figure 2: Inverse DCT algorithm by Vetterli-Ligtenberg where Rot(Q) is a rotation operation taking inputs $[x, y]$ and producing outputs $[X, Y]$ has the form of $X = x \cos Q + y \sin Q$ and $Y = -x \sin Q + y \cos Q$.

3.1 Characterization of input

We define an input class to be a set of vector inputs having zero coefficients at certain positions. For example all the DCT inputs where all the coefficients are zero belong to a class and all the inputs which are zero in the first 4 components and non-zero elsewhere belong to a different class. Thus, for a size 8 DCT using a an 8-digit binary number to indicate the position of the zeros (ones indicate a non-zero coefficient) we have 256 different classes of inputs. For these classes we can measure, from typical compressed images, the relative occurrences of each class C_i. Let P_i be the probability of occurrence of C_i, where $i \in \{0, \ldots 2^b - 1\}$ is the integer value corresponding to the position of the zeros for a vector of size b. In this paper we will consider the particular cases where $b = 4$ (16 classes) and $b = 8$ (256 classes).

It is clear that larger number of zeros result in potentially faster IDCT implementations since operations such as multiplication by zero and sum of two zero values need not be performed. For example an all zero input (i.e. Class C_0) would require no operations while a Class C_{255} input would require the maximum number of operations. We first select one baseline algorithm (which gives an exact IDCT for any input) and then it is easy to see that for each input class one can find an exact corresponding IDCT implementation with minimum number of operations. We will refer to this implementation as the *reduced IDCT* version for a given class.

As will be seen later it is more desirable to have a baseline DCT algorithm with a butterfly-like structure as this allows usage of a tree-structured classification. We have thus chosen the Chen-Wang algorithm [10] as our baseline algorithm because of above desired properties and the minimum number of operations, refer to Fig. 2. The number of operations when applying this 1-D IDCT to a 8x8 block of 2-D image along the row and column sequentially is as follows, 176 multiplications (since using of shift can replace multiplication with power of two), 536 additions and 240 bitwise shifting. This algorithm has integer operations and uses fixed point arithmetic while still keeping enough accuracy to be compliant with the JPEG standard.

Obviously only the classes with a significant number of zeros will have algorithms that are much faster than the baseline, input-independent algorithm, and thus it may not be worthwhile (and indeed practical) to have a different algorithm for each class. Also note that we can use a reduced IDCT algorithm without errors in

the inverse transform operation, *only if we know to which class the input belongs.* Thus using input dependent methods requires classifying each of the successive input vectors to determine which algorithm to use. Since the classification itself has a cost our goal will be to define an algorithm where classification is used only when worthwile in terms of the overall complexity. To motivate this further we first consider a simple case.

3.2 Sequential classification

As indicated above, classification is needed to select the algorithm to use for a given input. A simple form of classification is based on a sequential binary test of the inputs where each test determines whether the input belongs or not to a certain class. Once the input class C_i has been determined, one can compute the IDCT using the corresponding reduced IDCT algorithm. The cost of this sequential test can be written as

$$C_{sq} = \sum_{i=0}^{255} (\sum_{j \leq i} C_{test_j} + C_{idct_i}) \cdot P_i \tag{3}$$

where C_{test_j} is the cost for testing input index with the index of j-th algorithm , C_{idct_i} is the cost of i-th algorithm and P_i is the probability that the input belongs to the i-th index. We assume that the probabilities P_i have been measured on a typical set of images. In order to achieve an efficient system we would need to sort the tests so as to minimize the overall cost. It can be easily shown that when $C_{test_j} = C_{test}$ i.e. the cost of all index tests is the same, testing the index according to descending order of P_j gives the minimum total cost.

The major practical shortcoming of this structure is that many "classification tests" have to be performed. Clearly, for classes having a small number of zeros it may not be worthwhile testing, since the reduction in IDCT complexity is minimal and may not compensate the increase in computation cost due to classification. Similarly it may not be worthwhile testing for classes that occur with very low probability. Thus we can modify the algorithm by testing whether the input belongs to each one of the classes upto a certain number of tests after which the input belongs to a class that has not been "tested", and we will thus use the default or baseline algorithm on it. The new cost function would then be:

$$C_{sq} = \sum_{i=0}^{i_{max}-1} (\sum_{j \leq i} C_{test_j} + C_{idct_i}) \cdot P_i + (\sum_{i=0}^{i_{max}-1} C_{test_j} + C_{default}) \cdot \sum_{i=imax}^{255} P_i, \tag{4}$$

Where i_{max} represents the first class which is not "tested for" at the input. Obviously, the above algorithm in general will still be too complex and not useful for practical scenarios because of the excessive overhead for classification which has to be performed sequentially. We use this algorithm to illustrate in simple case the trade-offs involved in the classification. We now consider a tree-structured classification algorithm which is more suitable for practical implementation.

3.3 Tree-structured classification

Consider the baseline full version of the IDCT algorithm we have chosen, which is shown in Fig. 2. In this IDCT algorithm there are three stages of operations, each of which involves 2, 4 and 8 input coefficients respectively. This tree-like structure provides a natural way of classifying the data by using 4 stages of classification going from coarse to fine classification.

The intuition underlying this idea is that the reduced IDCT algorithm for a class with many zeros should be much less complex than for a class with few zeros. Moreover, given a baseline algorithm such as that of Fig. 2 both the number *and* the position of the zeros affect the expected complexity of the reduced IDCT. For example, it is intuitively obvious (and can be verified experimentally) that the largest reduction in complexity achievable

when 4 out of 8 coeffients are zero will correspond to the case when either (X_0, X_4, X_6, X_2) or (X_7, X_1, X_3, X_5) are all zero. Thus, the most efficient way to test whether a set of coefficients is zero will be to consider those sets of coefficients that are used together at one of the stages of the IDCT in Fig. 2. Therefore, at each stage of the input classification, we can deal with certain sets of 8, 4 and 2 coefficients respectively, which correspond to those encountered in the IDCT operation. For completeness it is also possible to add a 4th stage in the classification so that we can determine whether individual coefficients are zero. In this manner we can fully classify the input data within 4 stages and place it into one of the 256 possible classes. The example of pseudo code for testing 2 coefficients at the second stage is as follows,

```
      if(X2==0 and X6==0)
      z0 = z3 = y0
      z1 = z2 = y1
/********* save computation for y2 and y3 **********/
   else if(X0==0 and X4==0)
    z0 = y3, z1 = y2
    z2 = -y2, z3 = -y3
/********* save computation for y0 and y1 **********/
   else        z0 = y0 + y3
      z1 = y1 + y2
      z2 = y1 - y2
      z3 = y0 - y3
/********* performing full version for this part **********/
```

This tree-structured classification is illustrated in Fig 3 where $S_{x_0x_4x_2x_6x_1x_7x_3x_5}$ represents a class of inputs. Each of the subscripts $x_0x_4x_2x_6x_1x_7x_3x_5$ are used to denote the information that is known for this subclass. Subscript x_j provides the available information for the j-th coefficient in the IDCT and can take the following values:

$$x_j = \begin{cases} n & n \in \{1, 2, ...\} \text{ when at least one of coefficients indexed with this number is nonzero} \\ 0 & \text{when } x_j \text{ is known to be zero} \\ d & \text{when } x_j \text{ can be either zero or nonzero (don't care)} \end{cases}$$

As an example of $S_{1111dd00}$ means that at least one of $\{x_0, x_4, x_2, x_6\}$ must be nonzero, x_1 and x_7 are "don't care", i.e. we have no information about them, and x_3 and x_5 are *both* known to be zeros. Thus the tree-structured classification can be represented as in Fig. 3 for the case where the vector size is 4. At the top level of the tree we test to see whether all four coefficients are zero. After the test the input set is partitioned into two classes, S_{0000} and S_{1111} corresponding, respectively, to the case where all 4 coefficients are zero and the case where there is at least one non-zero coefficient among the four. As further tests are performed it is possible to narrow down the position of any potential zeros. For example at the next stage of the classification we can test the leftmost two coefficients, then the rightmost two coefficients. Class S_{1122} then represents the case where both tests indicate that there is at least a nonzero coefficient in each of the two pairs. Note that in the figure we outline the classes that are induced by testing in hierarchical manner, starting with tests on 4 coefficients and continuing with two and then one coefficient. The tree does not indicate in which order the tests should be performed. The goal of our optimization procedure is to find the best classification, that is, to determine (i) the best order in which to apply the tests and (ii) which tests it is worth performing (in the sense of reducing the average complexity).

It can be seen that for the worst case the maximum number of classification steps required to map an input sequence to a certain class corresponds to class S_{1234}. Conversely the minimum number of classification steps with this structure corresponds to the all-zero input case (S_{0000}) which only requires one test since the initial test already provides the answer. All zero inputs occur frequently in typical image decoding scenarios. Thus, since testing is simple, and the payoff is high (no computation has to be performed if an all zero input is detected),

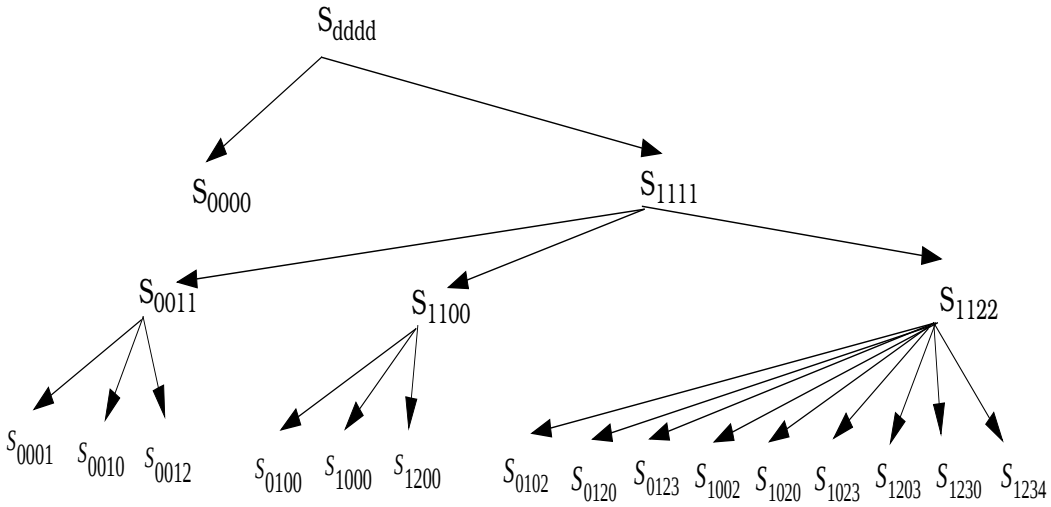

Figure 3: Classification diagram for a given input into 16 possible classes.

all-zero tests are frequently performed in many software implementations of IDCT (for example in [7–9]).

Obviously by using a full tree-structured classification we can separate inputs into the same 256 classes considered before. The advantage of the tree-structured classification is that the process can be stopped at any level of the tree and classification will still be useful since the classification tree is linked to the DCT computation tree. If we decide not to continue classifying the data the appropriate "default" algorithm for that subclass could be used. Our motivation is then to find for a given set of input data the tree for which the average complexity would be minimized. We thus need to find the statistics of the input data to determine whether a certain stage is worth keeping or not since the operations saved from knowing that some coefficients are zeros may not be enough to compensate the operations required to test for zeros. We use training data from typical images at a given quantization level to determine the probabilities P_i. We also need to have cost estimates for the different operations, including the cost of testing for zeros at each stage and the costs of IDCT of each class. The minimization can then be formalized as

$$Ctree_{min} = \min_{p \in \mathcal{P}} \sum_{i \in p} (C_{test_i} + C_{idct_i}) \cdot P'_i, \tag{5}$$

where $\mathcal{P}$ is the set of partitions of the input set of classes induced by all possible classification trees with the given tree structure. P'_i is the probability of class i, one of the subclasses in the partition, and C_{test_i} and C_{idct_i} are, respectively, the classification cost to determine inclusion in a subclass from the partition and the cost of performing the IDCT on inputs belonging to that class. $\mathcal{P}$ in effect represents all the possible trees.

As an example consider the case of Fig. 3. Assume that we only perform tests on blocks of 4 and 2 coefficients, in order to match the characteristics of the DCT algorithm. If no test is performed the partition contains one single class $p_0 = \{S_{dddd}\}$ and the only cost is that of the default IDCT. If we perform the size 4 test the partition is $p_1 = \{S_{0000}, S_{1111}\}$. If in addition the two size 2 tests are performed the resulting partition will be $p_2 = \{S_{0000}, S_{0011}, S_{1100}, S_{1122}\}$. In this case each of the classes in the partition can have a different reduced IDCT algorithm. Assume that we want to decide whether it is worthwhile to use the all-zero test, i.e. to use

partition p_1 instead of the trivial partition p_0. Then, for the given input data we should compare the cost of performing no tests, i.e. $C_{\bar{T}} = C_{idct_{default}}$, and the cost if partition p_1 is used, i.e.

$$C_T = C_{test} + P(S_{0000}) \cdot 0 + P(S_{1111}) \cdot C_{idct_{default}}.$$

Note that the cost of performing the IDCT for class S_{0000} is zero. Also note that since we consider all the classes in the partition we have that $S_{0000} \cup S_{1111} = S_{dddd}$ and $P(S_{0000}) + P(S_{1111}) = 1$. In this simple example we can make a decision whether or not to use the test and thus partition the class of inputs, depending on which of $C_{\bar{T}}$ and C_T is smaller. In terms of the classification tree we can decide whether to prune it or not.

Note how in the above example we have two possible tests of size 2 and thus we also need to determine in which order they should be performed. Refer to Fig. 4. Here we denote A and B, respectively, the tests on the left and right coefficients. C represents the option of not performing any tests. Using this notation we can build a tree which represents all the alternative tests, namely, AC, BC, C, ABC, BAC, and so on for the lower levels. Given that at any given level of the tree we know the probability of reaching that point for the given input data we can then compare the two alternatives (split or merge) and select the best one. This pruning from the full tree back towards the root is optimal. Optimality is obvious from the structure of the tree. Consider the example of Fig. 4 and assume that we have found the best subtrees for the top three branches of the tree, A, B, C. That means we have the best way of computing the data given that our partitions are respectively $\{S_{00dd}, S_{11dd}\}$, $\{S_{dd11}, S_{dd00}\}$, and $\{S_{dddd}\}$. The subtrees correspond to further size 2 and size 1 tests. In order to determine the complexity of any subtree we only need to know the partition at the root of the subtree *and not the way this partition was achieved.* Thus the partition completely summarizes the information needed to estimate the cost of a given subtree. Using dynamic programming arguments it can then be proven that for a given partition one can select the best subtree without resulting in loss in overall optimality. Therefore to obtain the optimal solution one can build the full tree and the prune it back from the leaves up to the root. In the example of Fig. 4, assuming that the best subtrees are obtained for the partitions corresponding to A, B, C, we can then select the one subtree that has minimum cost.

In summary, trees such as the one in Fig. 4 can be used to represent all the possible alternatives (i.e. which tests should be performed and in which order). Once the complete tree has been generated it can be pruned back to a single succession of branches representing the tests that will be sequentially performed. Note however that the set of tests so obtained is only optimal among all tests constrained to operate on blocks of size 4, 2 and 1 (and in that order, e.g. we cannot test first for single coefficients and then perform tests of size 2.) These restrictions, however, should not affect the optimality of our results since in general for the IDCT algorithm under consideration it is preferable to operate that way.

4 RESULTS

In our experiments we implement a software image decoder using the tree-structured classication scheme described in the previous section. The main problem now is obtaining accurate models for the costs of the operations used in zero-testing and IDCT such as addition, multiplication, shift, OR, NOT, IF, etc. While this may be relatively straightforward in hardware implementations or for assembly language level programming, it is more complicated if we use a high level programming language, since the effect of memory accesses, size of the code and so forth are more difficult to characterize. This problem is very difficult to solve since it highly depends on compiler and machine architecture. To obtain more accurate estimates one could resort to an assembly language implementation or to measuring the average time complexity for each of the possible classification trees directly. This latter approach would clearly be too complex and still would not be guarantee an exact cost measure since many different factors affect the runtime performance.

For the sake of praticality, we just use a set of approximated weights for the operations involved. By feeding these approximated parameters into our search algorithms we can get a fairly accurate approximation to the

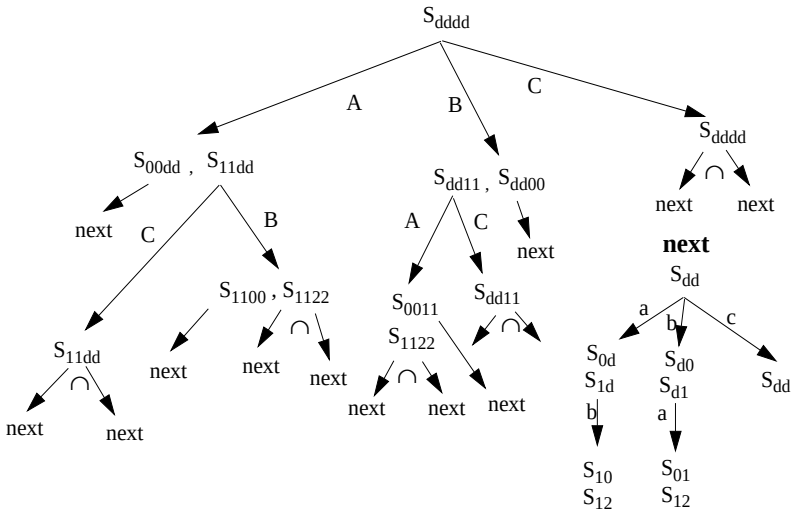

Figure 4: Example of search tree for size-4 input sequence where A and B represent the first and the second zero-test respectively, C means no zero-test. $\cap$ means the next stage classifications must be performed for both left half coefficients and right half coefficients independently. The problem is to find a subtree from the root down to nodes which gives the minimum total cost.

optimal solustion. We use a Sun Sparcstation 5 running Solaris 2.5 with the gcc compiler, we use the following assessment for each operation [11]. Addition, shift and OR are weighted with the same amount, say 1. Multiplication is weighted by 3. One IF statement which consist of comparing to zero and jump. Comparing with zero uses roughly the same time as addition while jump uses at least 3 times more than addition thus the approximate for IF is 5 to 6 times more than addition. One more thing to be considered here is memory access. Whenever one variable is involved in an operation, there is either a read-in or write-out time, however we weight this by 1. We emphasize that since our costs are only approximated we are not guaranteed that the performance of our optimized algorithm will be the best in terms of runtime. As will be seen though the approximate model results in overall better performance than other methods we tested.

The results show the estimated complexity comparison between (i) IDCT based on our proposed method, (ii) the baseline algorithm with all-zero test, (iii) the baseline algorithm with all-zero test for the first 1-D IDCT and ac-zero test for the second 1-D IDCT (since after the first 1-D IDCT, it is more likely for typical images that only the DC coefficient in the 1D vector is non-zero) and (iv) the FW method for various mean squared error adjusted by changing the quantization parameter. We use the example quantization table from the JPEG standard [2]. All the values are normalized by the complexity of the baseline algorithm without all-zero test. Next the actual implementation times are shown for IDCT algorithm part and total decoding time including read-in and write-out input data. Also shown in the total time plot is the mismatch case i.e. we use the IDCT algorithm optimized for a specific image at a certain quality factor for a different image and/or quality factor.

As seen in the figures, our proposed method (optimized tree-structured zero-test algorithm) achieves around 35% reduction of complexity for the total decoding time and upto 50% when only the IDCT part is considered. This supports the claim that IDCT part is the most computation consumption part at the decoder. For the mismatch problem, the degradation gets worse as the statistics of input data gets far apart from the statistics of the training data. However, the assessment for the cost of each operation must be precise enough to maintain the consistency between the estimated complexity and the actual complexity. As seen for both experimental results, even with mismatches, the optimized algorithm is a little bit inferior because of the lack of an exact model for the

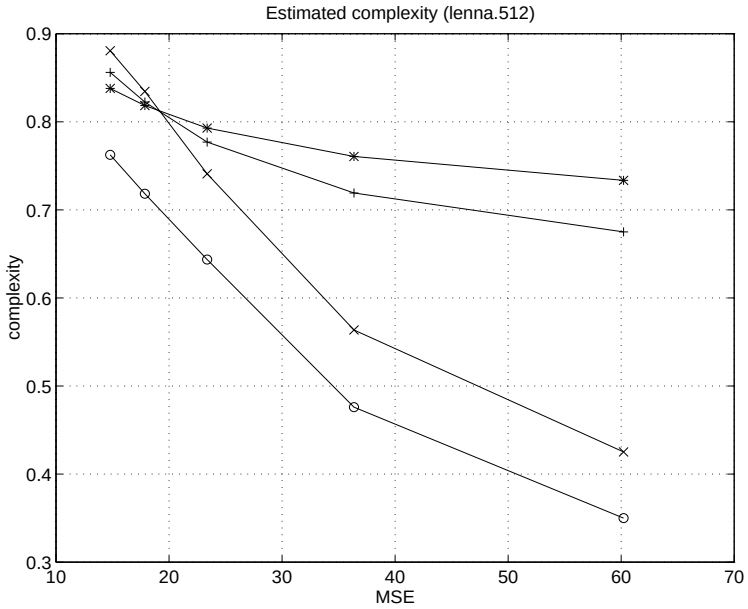

Figure 5: Normalized estimated complexity for lenna image where '+': CW with all-zero test algorithm, 'x': CW with all-zero test for the first 1-D IDCT and ac-zero test for the second 1-D IDCT, '*': FW algorithm, 'o': optimized tree-structured zero-test algorithm.

complexity. However, if our estimation for cost of operations is precise enough, a conclusion drawn from the result when applying optimized IDCT algorithm for "lenna" with MSE 60.21 could be that the degradation for using high-MSE optimized algorithm (overestimate number of zeros) is less than that for low-MSE optimized algorithm (underestimate number of zeros). This may not be true for other types of images. Also it has to be pointed out that our current implementation has concentrated mostly on the issue of structuring the tests, and thus uses a very simple programming style. We are planning to incorporate more efficient programming techniques, as those used in [7–9], in future versions of our system. Our goal at this stage was to implement all the algorithms with a similar style so that a fair comparison is possible.

One application of our method would be the scenario where the encoder can send the information about the optimized tree-structure zero-test IDCT algorithm to the decoder along with the data such that the decoder can do the IDCT operation with the minimum complexity. The problem with this application is that the encoder needs to know details about the architecture and type of programming language used at the decoder end, which may not be available. This would be another instance of a mismatch problem. One way to get around this is to instead perform the optimization for the zero-test structure at the decoder end with the past statistic of input data. This seems to be more practical when input data is large and have pretty constant statistic such as the bitstream of teleconference. For a specific case when the encoder has enough information about decoder end and the issue of limited complexity at the decoder is more important than the bandwidth, the complexity/distortion trade-off will be addressed i.e. within a given computational complexity at the decoder, the encoder adjusts the quantization parameter such that the distortion is minimized while satisfying the complexity constraint. This problem is addressed in an upcoming paper [12].

5 REFERENCES

[1] K. Rao and P. Yip, *Discrete Cosine Transform, Algorithms, Advantages, Applications.* Academic Press,

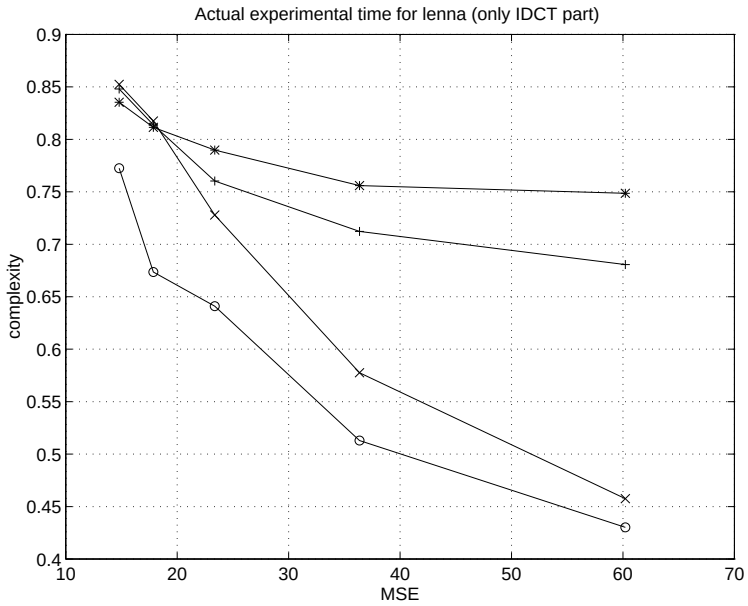

Figure 6: Normalized actual time complexity (only IDCT algorithm part) for lenna image where '+': CW with all-zero test algorithm, 'x': CW with all-zero test for the first 1-D IDCT and ac-zero test for the second 1-D IDCT, '*': FW algorithm, 'o': optimized tree-structured zero-test algorithm.

1990.

[2] W. Pennebaker and J. Mitchell, *JPEG Still Image Data Compression Standard.* Van Nostrand Reinhold, 1994.

[3] N. Merhav and V. Bhaskaran, "A transform domain approach to spatial domain image scaling," in *Proc. of ICASSP'96*, vol. 4, (Atlanta, GA), pp. 2405–2409, May 1996.

[4] A. Ligtenberg and M. Vetterli, "A discrete fourier/cosine transform chip," *IEEE J. on Selected Areas in Comm.*, vol. SAC-4, pp. 49–61, Jan 1986.

[5] P. Duhamel and H. H'Mida, "New 2n DCT algorithms suitable for VLSI implementation," in *Proc. of ICASSP'87*, (Dallas), p. 1805, Apr 1987.

[6] K. Froitzheim and H. Wolf, "A knowledge-based approach to JPEG acceleration," in *Proc. of IS&T/SPIE Symp. on Electr. Imaging Science and technology*, (San Jose), Feb. 1995.

[7] "The independent JPEG's group software JPEG, version 6." `ftp://ftp.uu.net`.

[8] "MPEG video software decoder, v2.2." `http://bmrc.berkeley.edu/projects/mpeg/mpeg_play.html`.

[9] "vic:UCB/LBL video conferencing tool." `ftp://ftp.ee.lbl.gov/conferencing/vic/`.

[10] Z. Wang, "Fast algorithms for the discrete w transform and for the discrete fourier transform," *IEEE Trans. on Signal Proc.*, vol. ASSP-32, pp. 803–816, Aug. 1984.

[11] D.A.Patterson and J.L.Hennessy, *Computer Architecture : a Quantitative Approach 2nd Ed.* Morgan Kaufmann Publishers, 1996.

[12] K.Lengwehasatit and A.Ortega, "Distortion/decoding time tradeoffs in software DCT-based image coding," in *Proc. of ICASSP '97*, 1997. To appear.

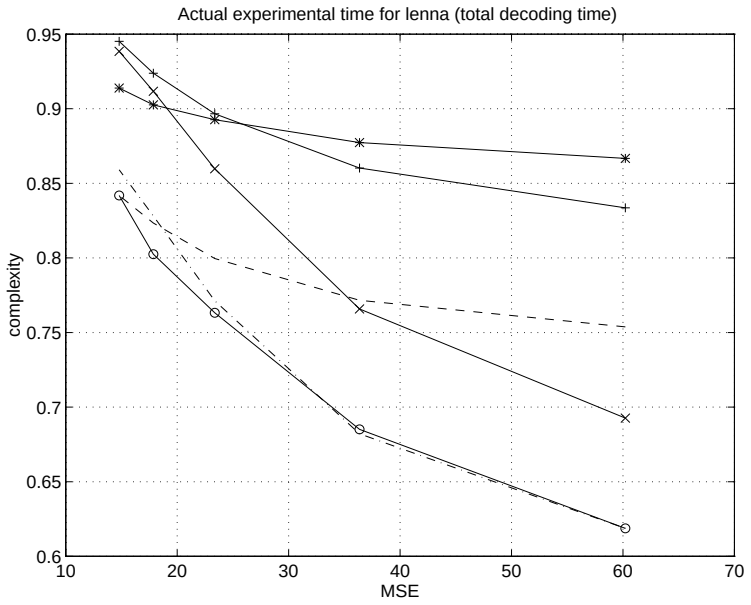

Figure 7: Normalized actual time complexity (total decoding time) for lenna image where '+': CW with all-zero test algorithm, 'x': CW with all-zero test for the first 1-D IDCT and ac-zero test for the second 1-D IDCT, '*': FW algorithm, 'o': optimized tree-structured zero-test algorithm. '–':optimized algorithm for lenna with MSE 14.79, '-.':optimized algorithm for lenna with MSE 60.21.

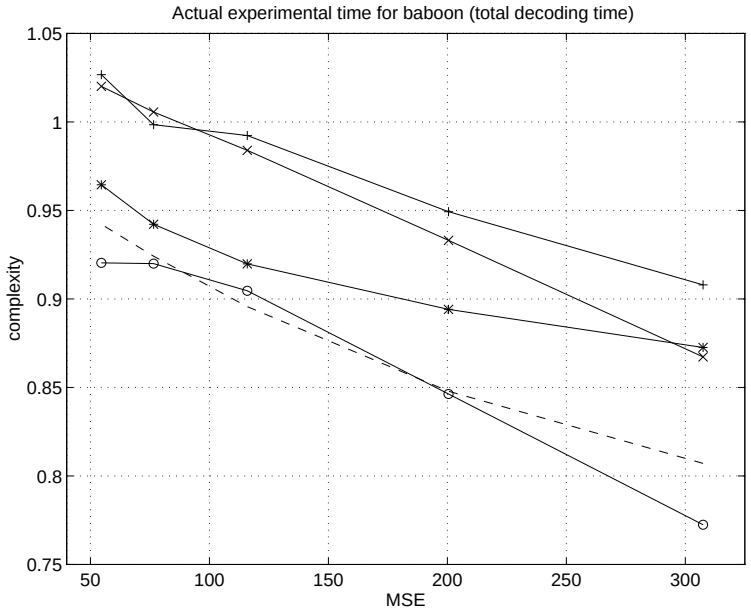

Figure 8: Normalized actual time complexity (total decoding time) for baboon image where '+': CW with all-zero test algorithm, 'x': CW with all-zero test for the first 1-D IDCT and ac-zero test for the second 1-D IDCT, '*': FW algorithm, 'o': optimized tree-structured zero-test algorithm, '–':optimized algorithm for lenna with MSE 14.79