Pruning the Fast Discrete Cosine Transform
Zhongde Wang

Abstract—The matrix representation of the simple structured algorithm for the discrete cosine transform (DCT), which was first introduced by Morikawa et al. [1] based on the successive order reduction of the Chebyshev polynomial, and retrieved by a simpler approach by Wang [2], is reviewed. A fast pruning algorithm for the DCT is then developed. It is found that for the commonly used version of the DCT, i.e., the DCT-II, \(K/2 + N(N_1 - K)/2K + 1\) multiplications and \(3K(N - 1 + N_1)/2 + N_1K - N_1(\text{for } N_1 < K)\) or \(3K/2 - N_1 + N_1 + N/N_1\) (for \(N_1 = K\)) additions are required for computing the \(N_1\) lowest frequency DCT-II components where \(K = 2^s < N_1 < 2K < N\).

I. INTRODUCTION

The discrete cosine transform (DCT) has a variety of applications in signal processing. In the last few years, a number of algorithms were developed [1]–[8] for the DCT. Although these algorithms possess the same efficiency in terms of operation counts, their approaches are quite different. Lee’s FCT [3] algorithm possesses very simple structure. However, the use of secants as multiplication coefficients is a disadvantage because larger roundoff error might be caused by the secant multiplication coefficients. Algorithms in [4] and [5] use cosines and sines as multipliers and thus will cause less error than that of Lee’s FCT. However, their structures are more complicated. References [6], [7], and [8] describe the same recursive algorithm which is actually the inverse of Lee’s FCT. In the author’s opinion, it’s error property will not be as good as those algorithms given in [4] and [5], because the errors will be accumulated during the recursive computation. The simple structured FCT algorithm (SFCT), which was first introduced by Morikawa et al. based on the successive order reduction of the Chebyshev polynomial [1], and was simply derived by a different approach by Wang [2], possesses both advantages. Its structure is as simple as Lee’s FCT. On the other hand, all its multiplication coefficients are cosines. The fixed-point error properties of this algorithm have been shown [9] to be even better than that of algorithms given in [4] and [5], and therefore it can be considered to be the best FCT algorithm at present.

Although many FCT algorithms have been developed, they compute all DCT coefficients at the same time. However, in many DCT applications, the most useful information about the signal is kept by the low-frequency DCT components. Therefore, only low-frequency DCT components are necessary to be computed in these applications. However, not even one algorithm has been developed for this purpose.

In this paper, a pruning FCT algorithm, which is based on the SFCT, is derived for this purpose. Since [1] and [2] were written in Japanese and Chinese which most English readers are not familiar with, we shall first show the matrix representation of the SFCT. The pruning DCT algorithm (PCT) is then developed. Since there are many types of the DCT [10], the PCT algorithm can be applied to all of them, although we shall give the PCT algorithm for the DCT-II only. The computational saving is then evaluated.

II. THE DISCRETE COSINE TRANSFORM

The DCT are classified into two categories [10]: the even types and the odd types. There are four different versions in each category. The even DCT-II is the most commonly used one. However, the even DCT-I has also been successfully applied in image processing applications [11]. Since all DCT’s which have been found applications belong to the even types, and the simple structured DCT algorithm was also developed for the even type DCT’s, we will confine attention to even DCT’s only in this paper. Let \(N = 2^N\). The four even type DCT matrices are defined as

\[
C_{N+1}^{I} = \sqrt{\frac{2}{N}} k_m k_n \cos(mn \pi / N),
\]

\(m, n = 0, 1, \ldots, N,\) \hspace{1cm} (1)

\[
C_{N}^{II} = \sqrt{\frac{2}{N}} k_m \cos(m(n + 1/2) \pi / N),
\]

\(m, n = 0, 1, \ldots, N - 1,\) \hspace{1cm} (2)

\[
C_{N}^{III} = \sqrt{\frac{2}{N}} [k_n \cos((m + 1/2)n \pi / N)],
\]

\(m, n = 0, 1, \ldots, N - 1,\) \hspace{1cm} (3)

and

\[
C_{N}^{IV} = \sqrt{\frac{2}{N}} \cos((m + 1/2)(n + 1/2) \pi / N),
\]

\(m, n = 0, 1, \ldots, N - 1,\) \hspace{1cm} (4)

where the subscript of \(C\) represents the order of the matrix, the Roman superscript represents the type of the DCT, and

\[
k_j = \begin{cases} 1/\sqrt{2}, & j = 0 \text{ or } N, \\ 0, & \text{otherwise}. \end{cases}
\]
The DCT-III is actually the inverse of the DCT-II. The DCT-IV itself has not found applications yet and the algorithms for the DCT-I can be derived from algorithms for the DCT-II [12]. For this reason, in what follows, we shall be concerned with the simple structured algorithm for the DCT-II only. Since constants \( \sqrt{2/N} \) and \( k_m \) in (2) do not affect the analysis of the SFCT algorithm, we shall omit them in the following sections.

III. THE SIMPLE STRUCTURED ALGORITHM FOR THE DCT-II

The Hadamard order is involved in the algorithm. Let \( h_K(i) \) represent the sequency (number of sign changes) of the \( i \)th row of a \( K \times K \) Hadamard matrix [13]. If a sequence is ordered according to \( h_K(i) \), we say that the sequence is in a Hadamard order. \( h_K(i) \) can be generated recursively as follows:

\[
h_1(0) = 0, \quad h_{2K}(2i) = h_K(i),
\]

and

\[
h_{2K}(2i) = 2K - 1 - h_K(i).
\]

The SFCT algorithm can be represented by a matrix factorization of the DCT-II matrix. Let \( \mathbf{H}_N \) be a permutation matrix which converts a natural ordered vector into a Hadamard ordered one. Then

\[
C_N^{(0)} = C_N^{(1)} \mathbf{H}_N
\]

\[
= \cos(m(h_K(n) + 1/2)\Pi/N)
\]

\[
m, n = 0, 1, \cdots, N - 1,
\]

is the DCT-II matrix with a Hadamard ordered input where \( n \) represents matrix transposition. Let \( K = 2^k \leq N \) and \( J = N/K \). We partition the first \( J \) rows of \( C_N^{(0)} \) into \( K \) square matrices of order \( J \). The \( k \)th submatrix is given by

\[
C_j^{(k)} = [\cos(m(h_N(iJ + j) + 1/2)\Pi/N)],
\]

\[
m, j = 0, 1, \cdots, J - 1, \quad i = 0, 1, \cdots, K - 1.
\]

Then \( C_N^{(0)} \) can be factorized as

\[
C_N^{(0)} = R_N B_N
\]

\[
\begin{bmatrix}
C_{N/2}^{(0)} & 0 \\
0 & C_{N/2}^{(1)}
\end{bmatrix}
\]

where

\[
B_N = \begin{bmatrix}
I_{N/2} & I_{N/2} \\
I_{N/2} & -I_{N/2}
\end{bmatrix}
\]

and

\[
R_N = \begin{bmatrix}
1 & 0 & 0 \\
I_{N/2-1} & \vdots & I_{N/2-1} \\
-1 & \vdots & -1
\end{bmatrix}
\]

\[
d(1) = \cos(\Pi/4).
\]

Each submatrix can be further factorized in the same manner

\[
C_j^{(k)} = R_j^{(k)} B_j
\]

\[
\begin{bmatrix}
C_{J/2}^{(2k)} & 0 \\
0 & C_{J/2}^{(2k+1)}
\end{bmatrix}
\]

where \( B_j \) is given by (10), and

\[
R_j^{(i)} = \begin{bmatrix}
1 & 0 & 0 \\
I_{J/2-1} & \vdots & I_{J/2-1} \\
-1 & \vdots & -1
\end{bmatrix}
\]

\[
d(K + i) = \cos((h_K(i) + 1/2)\Pi/2K).
\]

\[
i = 0, 1, \cdots, K - 1, \quad K = 2^k \geq 2.
\]

Equations (9) and (13) can be viewed as the decomposition of \( C_j^{(k)} \) into sparse matrices \( B \) and \( C_j^{(k)} \) with lower order. On the other hand, they may also be viewed as composition of \( C_j^{(k)} \) with higher order by \( B, R_j \), and \( C_j^{(k)} \) with lower order. The computational requirements for \( C_j^{(k)} \) can be derived from the above equations.

Each decomposition represented by (9) or (13) will be treated as one stage which consists of two steps. The first step, represented by \( B_j \), requires \( J \) additions only. The second step, represented by \( R_j \) in (9) and (13), requires \( J/2 - 1 \) multiplications together with \( J/2 - 1 \) additions. There are \( K = 2^k \) submatrices of \( B \) and \( B_j \) in the \( k \)th stage. Since all \( C_j^{(k)} \) possess the same structure of decomposition, thus all \( C_j^{(k)} \) with the same order requires the same number of operations. Upon representing the number of multiplications and additions for computing a transform represented by a matrix \( A \) by \( \mu(A) \) and \( \alpha(A) \), respectively, we have, from (9) and (13),

\[
\mu(C_j) = \mu(R_j) + \mu(B_j) + 2\mu(C_{J/2})
\]

\[
= J/2 + 2\mu(C_{J/2}),
\]

and

\[
\alpha(C_j) = \alpha(R_j) + \alpha(B_j) + 2\alpha(C_{J/2})
\]

\[
= 3J/2 - 1 + 2\alpha(C_{J/2}).
\]

where \( C_j \) represents \( C_j^{(i)} \) with any index \( i \). After \( M \) stages of decomposition, one gets the \( N \) smallest submatrices. Each of them contains one element of the first row of \( C_N^{(0)} \) only. It turns out that all of them are unit and no more computation is required. Since one more multiplication is needed to compute \( k_0 \) of (5), the total number of multiplications required to compute an \( N \) point DCT-II is thus given by

\[
\mu(C_N^{(II)}) = MN/2 + 1
\]

while the total number of additions is given by

\[
\alpha(C_N^{(II)}) = 3MN/2 - (2^0 + 2^1 + \cdots + 2^{M-1})
\]

\[
= 3MN/2 - N + 1.
\]

These figures are identical with the operation counts required by other efficient FCT algorithms such as those given
in [30]-[8]. Notice that [3] did not count the multiplication to compute \( k_0 \), and the figure of multiplication given in [3] is one less than that given by (16).

Fig. 1 plots the signal flow graph of a length-16 DCT-II; for now, the solid and broken lines mean the same. The simple structure of this algorithm is obvious. Notice that the input sequence is in a Hadamard order, while the output sequence, which consists of the DCT-II coefficients, is in a natural order.

It has been shown [2] that the multiplication coefficients involved in the simple structured algorithm are all cosines. They can be generated by the following recursive relations:

\[
\begin{align*}
    d(1) &= \sqrt{0.5}, \\
    d(2i) &= [0.5(1 + d(i))]^{1/2}, \\
    d(2i + 1) &= [0.5(1 - d(i))]^{1/2}.
\end{align*}
\]

In general, taking a square root is faster than calling internal cosine function in a computer. Therefore, (18) yields a higher efficiency in generating the multiplication coefficients.

IV. THE PRUNING ALGORITHM FOR THE DCT

Based on the algorithm given in the previous section, we shall derive the pruning algorithm for the DCT-II in this section.

Let us suppose that only the first \( N_1 \) rows of \( C_J^{(i)} \) in (13) have to be composed from \( R_i, B_j, C_J^{(2i)}, \) and \( C_J^{(2i+1)} \).

Let \( R_i(N_1), B_j(N_1) \), and \( C_J^{(2i)}(N_1) \) represent the corresponding matrices with zero elements of any row whose index is greater than or equal to \( N_1 \). In other words, only the elements of the first \( N_1 \) rows are nonzero elements. Then, (13) becomes

\[
C_J^{(i)}(N_1) = R(N_1) C_J^{(2i)}(N_1) B(N_1) J
\]

if \( N_1 < J/2 \); or

\[
C_J^{(i)}(N_1) = R(N_1) C_J^{(2i)}/(N_1) B(N_1) J
\]

if \( N_1 \geq J/2 \). These relations also hold for \( C_J^{(0)} \). Therefore, if \( K = 2^k \leq N \leq 2K \leq N \) and suppose \( N_1 \) low frequency DCT-II components are to be computed, then (19a) can be applied recursively until the condition \( N > J/2 \) is satisfied.

After one more stage of decomposition represented by (19b), the remaining stages of decomposition have to apply (13).

Since the output of the transform represented by \( C_J^{(0)} \) is in a natural order, the first \( N_1 \) rows of it will compute the first \( N_1 \) DCT-II components. Therefore, the (19) actually gives the desired pruning DCT-II algorithm. The signal flow graph for this algorithm is also shown in Fig. 1 where the solid lines indicate the operations to be executed, while broken lines indicate the operations which have been saved by this pruning algorithm. It is obvious from (19) that if \( N_1 \leq J/2 \), \( R(N_1) \) does not need any operations. On the other hand, \( R(N_1) \) requires \( N_1 - J/2 \) multiplications and \( N_1 - J/2 - 1 \) additions if \( J > N_1 > J/2 \). In all above three cases, \( B(N_1) \) requires \( N_1 \) additions. Therefore, we may get the recursive relations regarding the operation counts for (19). They are given by

\[
\begin{align*}
    \mu_{N_1}(C_J) &= 2\mu_{N_1}(C_J/2) & \text{for } N_1 \leq J/2, \\
                 &= 2\mu_{N_1}(C_J/2) + N_1 - J/2 & \text{for } J > N_1 > J/2, \\
                 &= 2\mu_{N_1}(C_J/2) + J & \text{for } N_1 \geq J,
\end{align*}
\]

and

\[
\begin{align*}
    \alpha_{N_1}(C_J) &= 2\alpha_{N_1}(C_J/2) + N_1 & \text{for } N_1 \leq J/2 + 1, \\
                 &= 2\alpha_{N_1}(C_J/2) + 2N_1 - J/2 - 1 & \text{for } J \geq N_1 > J/2 + 1, \\
                 &= 2\alpha_{N_1}(C_J/2) + 3J/2 - 1 & \text{for } N_1 > J.
\end{align*}
\]

From these relations, the total number of multiplications and additions for computing the first \( N_1 \) DCT-II coefficients can
be obtained as

$$\mu_{N_1}(C_{N_1}^H) = (k - 1)N/2 + NN_1/2K + 1,$$

(22)

and

$$\alpha_{N_1}(C_{N_1}^H) = 3N(k - 1 + N_1/K)/2 - N_1 + N/2K$$

for $N_1 \neq K$, \hfill (23a)

or

$$\alpha_{N_1}(C_{N_1}^H) = 3kN/2 - N_1 + N/N_1$$

for $N_1 = K$. \hfill (23b)

It is found that both $\mu_{N_1}$ and $\alpha_{N_1}$ are roughly proportional to log $N_1$. More precisely,

$$\mu_{N_1}/\mu \approx \alpha_{N_1}/\alpha \approx \log N_1 / \log N.$$

Therefore, we may roughly estimate the percentage of saving obtained by the pruning as

$$G \approx (1 - \log N_1 / \log N) \times 100\%.$$  \hfill (23)

Although the pruning algorithm is shown for the DCT-II only, it is straightforward to develop the pruning algorithm for other types of the DCT using the same idea.

REFERENCES


