Design of Fast Transforms for High-Resolution Image and Video Coding

Yuriy A. Reznik and Ravi K. Chivukula
Qualcomm Inc, 5775 Morehouse Drive, San Diego, CA, USA 92121

ABSTRACT

We study factorization techniques and performance of Discrete Cosine Transforms of various sizes (including non-dyadic and odd numbers). In our construction we utilize an array of known techniques (such as Heideman's mapping between odd-sized DCT and DFT, Winograd fast DFT algorithms, prime-factoring, etc), and also propose a new decimation strategy for construction of even-sized scaled transforms. We then analyze complexity and coding gain of such transforms with sizes 2-64 and identify ones that show best complexity/performance tradeoffs.

Keywords: discrete cosine transform, DCT, FFT, factorization, multiplicative complexity, scaled transform, video coding standards, H.261, JPEG, MPEG-1, MPEG-2, MPEG-4, H.263, H.264.

1. INTRODUCTION

Discrete Cosine Transform of type II (DCT-II)[1-3] is a fundamental operation performed by the majority of today's image and video compression algorithms. It was first suggested by N. Ahmed, T. Natarajan, and K. R. Rao[1], and subsequent research provided a number of theoretical arguments for its use, such as energy compaction property, asymptotic equivalence of DCT to the Karhunen-Loève transform for signals produced by Markov-1 process with high correlation coefficient, etc. (see, e.g. [3, Chapter 3] for a survey of the related results).

DCT-II of size 8 has served as the transform of choice in H.261, JPEG, MPEG-1, MPEG-2, H.263, and MPEG-4 visual standards[2,9-13]. More recent standards, such as MPEG-4 AVC | H.264[14], VC-1[15], and AVS[16] have adopted integer approximations of DCT-II with transform sizes: 4, 8, and 16. An emerging JPEG-XR image compression standard[17] uses overlapping transforms, which are also based on 4-point DCT-II kernels.

The use of multiple transform sizes in more recent algorithms has been justified by changing image statistics and the need to adapt to them. E.g., given a flat region the use of a large transform would give better compression efficiency. On the other hand, given an image with a sharp edge, and a coarse quantization step size, it might be better to apply several smaller transforms as this would smear it less (the quantization error propagation stays bounded within each transform's block). In this respect, the choice of 4x4 and 16x16 transforms in H.264 standard seemed reasonable for achieving such adaptation.

At the same time, the authors are not aware of any fundamental law or reason suggesting that transforms of dyadic sizes (such as 2, 4, 8, 16) should be any better than transforms of other (e.g. using adjacent odd numbers) sizes. Most likely arguments for the use of dyadic sizes could have been:

- perceived implementation convenience (ease of addressing, alignment with memory banks), and
- known fast algorithms for computation of such transforms.

The second argument, indeed, has its historical roots: first algorithms for computing DCT have relied on FFT[2], which in its original (Cooley-Tukey) form was designed for dyadic transform sizes. However, more advanced research on fast DFT and DCT algorithms (cf. S. Winograd[24], M.T. Heideman[20], C.W. Kok[24], as well as[21]), indicate that some of the non-dyadic-sized transforms can be significantly less complex than dyadic-sized ones. In some cases, e.g, one reported in[21], the amount of computational or power savings can be close to a factor of 2, and we believe such phenomena might bring opportunities for improving designs of future image and video coding algorithms.

Further authors information: (please send correspondence to Yuriy A. Reznik)
Yuriy A. Reznik: yreznik@qualcomm.com, phone: +1 (858) 658 1866, Ravi K. Chivukula: rchivuku@qualcomm.com

In this paper we review some known results, derive few missing factorizations, and analyze complexity / coding gain tradeoffs of DCT-II transforms with sizes in the range of 2-64. Based on this analysis we suggest transform sizes and their progressions that appear to be most promising for use in image and video coding.

Section 2 provides definitions of DCT-II and related transforms and discusses their performance metrics. Section 3 reviews transforms that are currently used in image and video coding standards. It also identifies performance gap left by current transform choices. Section 4 analyses complexity and coding gain of transforms of various sizes, and identifies transforms that are most promising in this respect. Conclusions are presented in Section 6. Appendix A reviews techniques that have been used to derive transform factorizations and their complexity estimates. Appendix B lists few examples of transforms that we have derived in our work.

2. DEFINITIONS, PERFORMANCE AND COMPLEXITY METRICS

Let $\{x_n\}$, $n = 0,1,...,N-1$ represent an input sequence.

The DCT-II and inverse transforms over this sample are defined, correspondingly as[2,3]:

$$y_k = \sqrt{\frac{2}{N}}\lambda(k)\sum_{n=0}^{N-1} x_n \cos\left(\frac{(2n+1)\,k\pi}{2N}\right), \quad k = 0,1,....,N-1,$$

$$x_n = \sqrt{\frac{2}{N}}\sum_{k=0}^{N-1} y_k \lambda(k)\cos\left(\frac{(2n+1)\,k\pi}{2N}\right), \quad n = 0,1,....,N-1,$$

where $\lambda(k) = 1/\sqrt{2}$, if $k = 0$, otherwise 1.

Using vector notation the above mappings can also be rewritten as:

$$y = C^{II} x \text{ and } x = \left(C^{II}\right)^{-1} y = \left(C^{II}\right)^{T} y$$

where C^{II} is a matrix of transform factors.

In order to study performance of transforms it is customary to assume that our input signal x is produced by Markov-1 process with known adjacent element correlation parameter ρ. The covariance matrix of this process R_x therefore is given by:

$$R_x(i,j) = \rho^{|i-j|}, \quad i,j = 0,...,N-1.$$

If we now apply a transform with matrix A to x, then the covariance of the output signal:

$$y = A\,x\ ,$$

becomes:

$$R_y = A\,R_x A^T\ .$$

KLT transform [2] makes this matrix diagonal, which is ideal for subsequent quantization and coding. DCT-II was shown to achieve essentially the same result, but only when $\rho \to 1$ [2,3].

Transform efficiency metric[19] provides a simple tool for measuring degree to which transform decorrelates input signal (i.e. turns its covariance matrix into diagonal one):

$$\eta = 100\frac{\sum_{i=0}^{N-1}|r_{ii}|}{\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}|r_{ij}|}\ \ [\%],$$

where r_{ij} are the elements of R_y. It is easy to see that KLT will always have efficiency of 100%, whereas DCT-II will show lower numbers (decreasing with smaller ρ and/or larger block sizes).

Transform coding gain [4] is another metric, defined as follows:

$$C_g = 10\log_{10}\frac{\frac{1}{N}\sum_{i=0}^{N-1}\sigma_{x_i}^2}{\left(\prod_{i=0}^{N-1}\sigma_{x_i}^2\,\|f_i\|^2\right)^{1/N}}\ \ [\text{dB}],$$

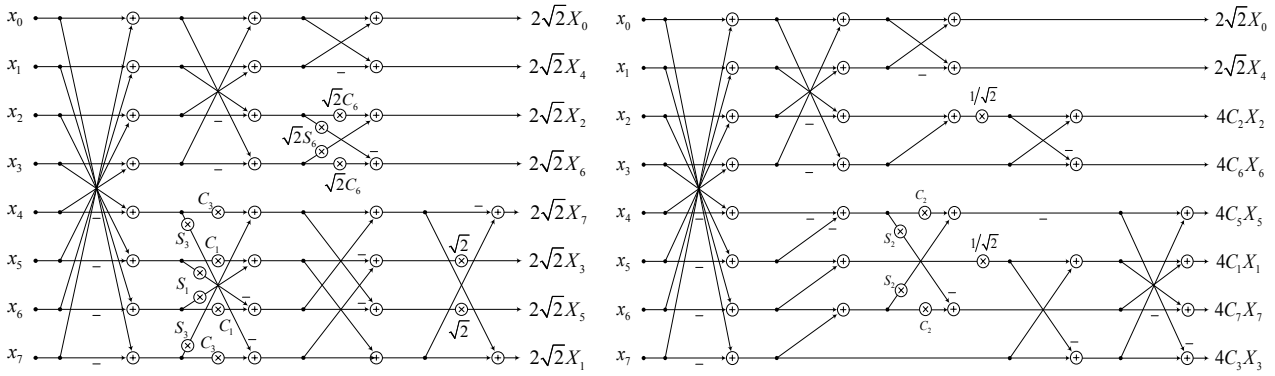


Fig. 1. Popular practical implementations of 8-point DCT-II: factorization by Loeffler, Ligtenberg, and Moschytz[5] (left), and scaled transform factorization by Arai, Agui, and Nakajima[6] (right). Both factorization use multiplications by factors: $C_k = \cos(k\pi/16)$, $S_k = \sin(k\pi/16)$.

where $\sigma_{x_i}^2$ denotes variance of i-th transform coefficient (that is i-th diagonal element in matrix R_y), and $\|f_i\|^2$ denotes the L2-norm of i-th basis function of the transform matrix (which turns to 1 for transforms with orthonormal bases). In essence, this metric says what would be the reduction of MSE due to transform coding (in comparison of PCM vs transform-based encoder using scalar quantizer and optimal bit-allocation strategy) under sufficiently high rates[4].

In our analysis we will use both of these metrics, although coding gain will be of primary interest.

Additionally, in this work we will estimate *complexity* of computing transforms of various sizes.

There will be two types of estimates that we will provide:

- complexity of computing the entire (orthogonal) transform;
- complexity of computing only *scaled transform*.

The transform is said to be *scaled*, when some of its factors are moved all the way to the front of the transform and excluded from computation. I.e., the transform matrix A becomes:

$$A = \Pi D \tilde{A} \ ,$$

where Π is some permutation matrix, D is a diagonal matrix of scale factors, and $\tilde{A}$ is a matrix of the remaining (*scaled*) transform. In systems employing scaled transform designs (such as, e.g. MPEG-4 AVC/H.264 standard) the multiplications by scale factors are usually absorbed into the quantization stage, therefore reducing complexity of the encoder (or decoder).

In order to estimate complexity, we will use best existing or derive new factorizations of transforms of various sizes, and then report the numbers of operations involved, such as:

- the numbers of multiplications (by irrational constant factors, prior to integerization);
- the number of additions (which will include the number of subtractions, as both assumed to be of same complexity); and
- the number of bit-wise shift operations (associated with products by dyadic constants).

We will also report a combined and normalized transform complexity number:

$$\xi(N) = \frac{3m + a + s}{N}$$

where N is a transform size, and m, a, and s denote the numbers of multiplications, additions, and shifts correspondingly. The weight 3 given to multiplications in this formula represents typical costs of their implementations by mapping them into integer arithmetic with subsequent unfolding into sequences of additions and shifts (see [8] and references therein).

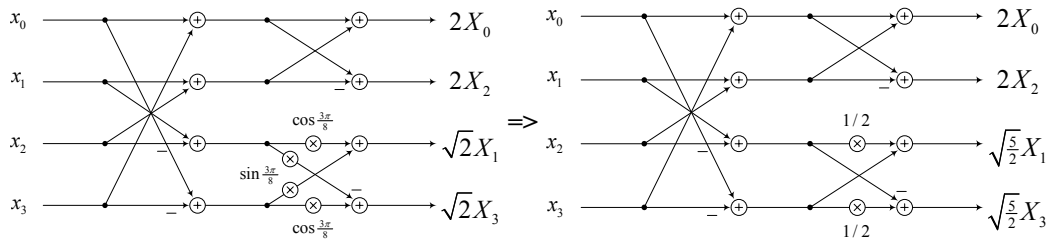


Fig. 2. Relationship between 4-point DCT-II (left) and 4-point "integer transform" of MPEG-4 AVC | H.264 standard.

3. TRANSFORMS CURRENTLY USED FOR VIDEO CODING

We first briefly review main types of transforms currently used in video coding.

3.1 8x8 DCT-II transform

DCT-II of size 8 was the original choice for transforms for H.261, JPEG, MPEG-1,2,4, and other early image and video coding algorithms [9]. Best known factorizations for computing such transforms include (see Fig 1):

- Loeffler, Ligtenberg, and Moschytz (LLM) factorization[5], requiring 11 multiplications and 29 additions; and
- Arai, Agui, and Nakajima (AAN) scaled factorization[6], requiring only 5 multiplications and 29 additions per scaled 8-point transform.

Many efficient fixed-point approximations of 8-point DCT-II have also been derived[9]. Among least complex fixed-point DCT-II realizations is the algorithm of Reznik, Hinds, and Rijavec[7]: it uses only 42 addition and 16 shift operations per scaled 8-point transform.

Recently developed MPEG-C Part 2 standard[8,9] is another fixed-point approximation of DCT-II. This standard offers much higher precision (satisfying all precisions tests of MPEG standards), and it also has a very reasonable complexity: 44 additions and 20 shifts per 8-point transform.

3.2 4x4 transform in MPEG-4 AVC | H.264 standard

MPEG-4 AVC | H.264 standard[14] defines a 4-point "integer transform", which is essentially a conversion of a 4-point DCT-II factorization into a scaled form (see Fig. 2) and then using approximation for the remaining transform factors:

$$\cos\left(\tfrac{3\pi}{8}\right) / \sin\left(\tfrac{3\pi}{8}\right) = 0.41421... \approx 1/2 .$$

In order to make such a transform orthogonal, its scale factors must also be adjusted:

$$\sqrt{2} / \sin\left(\tfrac{3\pi}{8}\right) \approx \sqrt{\tfrac{5}{2}}$$

We note that MPEG-4 AVC | H.264 standard[14] actually defines asymmetrically scaled versions of forward and inverse transforms, but conceptually they are equivalent to the described design.

Complexity-wise 4-point H.264 transform requires only 8 additions and 2 shifts to implement. It attains coding gain of 5.375 dB, and efficiency of 95.2426 %. Compared to an ideal 4-point DCT-II, H.264 transform shows only 0.02dB loss in coding gain, and about 0.5% loss in efficiency.

3.3 8x8 transform in MPEG-4 AVC | H.264 standard

This transform can also be derived by starting with 8-point DCT-II factorization (see Fig. 3) and converting it into a scaled form. Note that design of upper (even portion) of the transform in this case becomes identical to the design of 4-point transform in H.264.

In lower (odd) portion, we use the following approximations:

$$\sin\left(\tfrac{\pi}{16}\right) / \cos\left(\tfrac{\pi}{16}\right) = 0.1989... \approx 1/4, \quad \sqrt{2} = 1.4142... \approx 3/2$$

and also replace scale factors with

$$2\sqrt{2} / \cos\left(\tfrac{\pi}{16}\right) \approx \tfrac{17}{4\sqrt{2}}$$

in order to make such transform orthogonal.

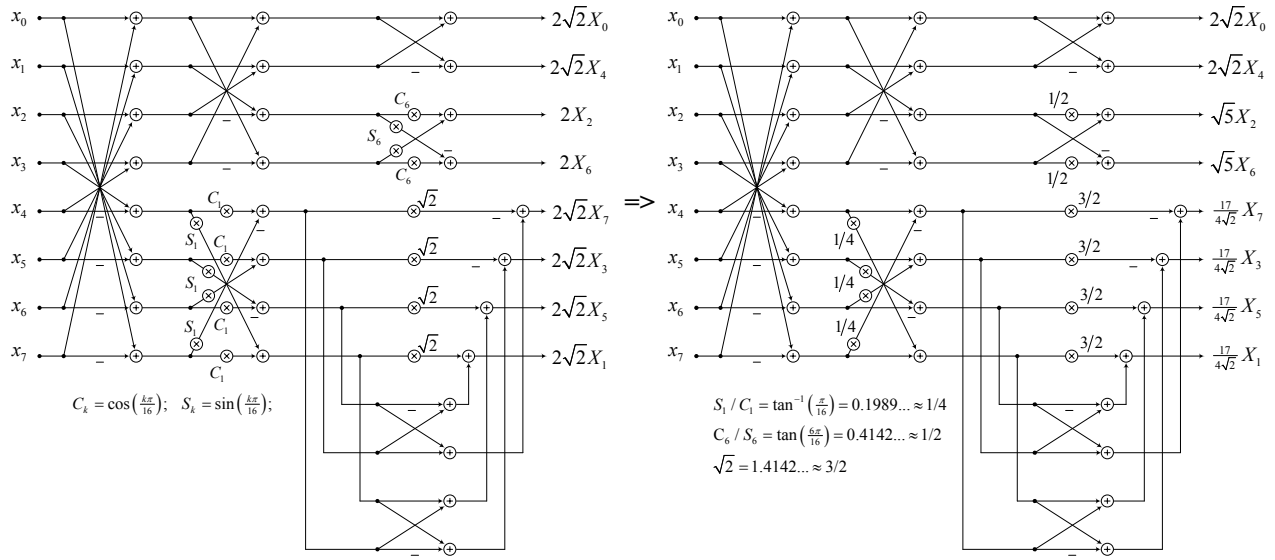


Fig. 3. Relationship between 8-point DCT-II (left) and 8-point "integer transform" of MPEG-4 AVC | H.264 standard.

We note that initial 8-point factorization used in H.264 standard[14] appears to be less efficient than well-known LLM or AAN factorizations (it needs at least 13 multiplications and 31 additions to implement), but it becomes much simpler by conversion to scaled and then fixed point form.

The full complexity of 8-point transform in H.264 is 32 additions and 10 shifts. Its coding gain and efficiency results are summarized in Table 1. As evident, the price to for such simple fixed-point design is close to 0.1dB loss in coding gain and 1.5% loss in efficiency as compared to true (infinite precision) 8-point DCT-II transform.

Table 1. Comparison of 8-point transforms.

Transform	**Coding gain [dB] (p=0.95)**	**Efficiency [%] (p=0.95)**
DCT-II of size 8:	8.8462	93.9912
H.264 transform of size 8:	8.7832	92.4576

3.4 16x16 transform in MPEG-4 AVC | H.264

The 16-point transform (so-called "hierarchical transform") in MPEG-4 AVC | H.264 standard [14] is essentially a cascade of five 4-point transforms, shown in Fig.4. It can also be presented in matrix notation as follows:

$$C_{16}^{H.264} = \left(I_{16} + \left(W_4 - I_4 \right) \otimes \begin{pmatrix} 1 & \\ & O_3 \end{pmatrix} \right) \begin{pmatrix} C_4^{II} & & & 0 \\ & C_4^{II} & & \\ & & C_4^{II} & \\ 0 & & & C_4^{II} \end{pmatrix},$$

where W_4 denotes matrix of a "second stage" transform, I_4 and I_{16} denote identity matrices of sizes 4 and 16 correspondingly, O_3 denotes zero matrix of size 3, and $\otimes$ is a Kroneker product.

It is worth noting, is that at early stage of the development of the H.264 standard[14] its 16-point transform was actually a cascade of nested 4-point DCT-II approximations. The second stage transform was later replaced with Walsh-Hadamard transform. In Table 1 we show coding gains and efficiencies of full 16-point DCT-II and two cascaded transform designs produced in the development of H.264.

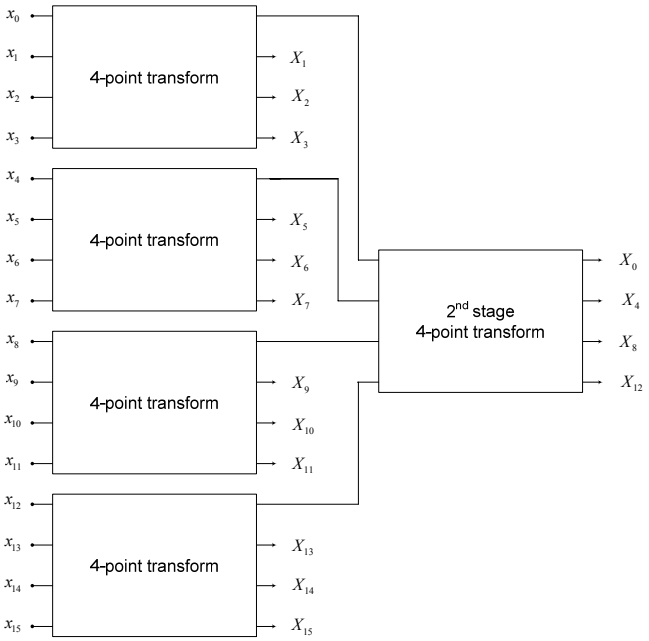


Fig. 4. Hierarchical 16-point transform of MPEG-4 AVC | H.264 standard.

Table 2. Comparison of 16-point transforms.

Transform	**Coding gain [dB] (p=0.95)**	**Efficiency [%] (p=0.95)**
DCT-II of size 16:	9.45547	88.4518
Cascade of DCT-II of size 4:	8.73236	72.3026
MPEG-4 AVC | H.264 16x16 transform	8.5663	67.6576

Based on these estimates, it appears that 16-point transform in H.264 has close to 1dB loss in coding gain (and over 20% loss in efficiency) as compared to full 16-point DCT-II. While some loss of efficiency due to cascade-style design could have been expected (see e.g. [18]), such a major drop in efficiency is actually very surprising.

3.5 Comparison with DCT-II of other sizes

We are now prepared to have a quick initial look at how these known transforms stack against DCT-II transforms of other possible sizes in terms of coding gain (see Fig. 5).

Based on this picture we immediately notice that:

- the choice of 8-point transforms in early video codecs has left a margin of over 1dB of coding gain unutilized;
- the choice of transforms in MPEG-4 AVC | H.264 standard is rather surprising: 4x4 transforms have significantly smaller coding gain, and hierarchical 16x16 transform appears to perform worse than 8-point DCT-II in earlier codecs!
- in order to capture the reamining1dB of gain (beyond what is reachable by 8-point DCT-II) it will be necessary to design transforms of considerably larger (30-40 points) sizes;
- there is absolutely no indication that transforms must be of dyadic sizes – coding gain simply grows monotonically with size.

Our next step is to look at complexity characteristics of transforms of various sizes.

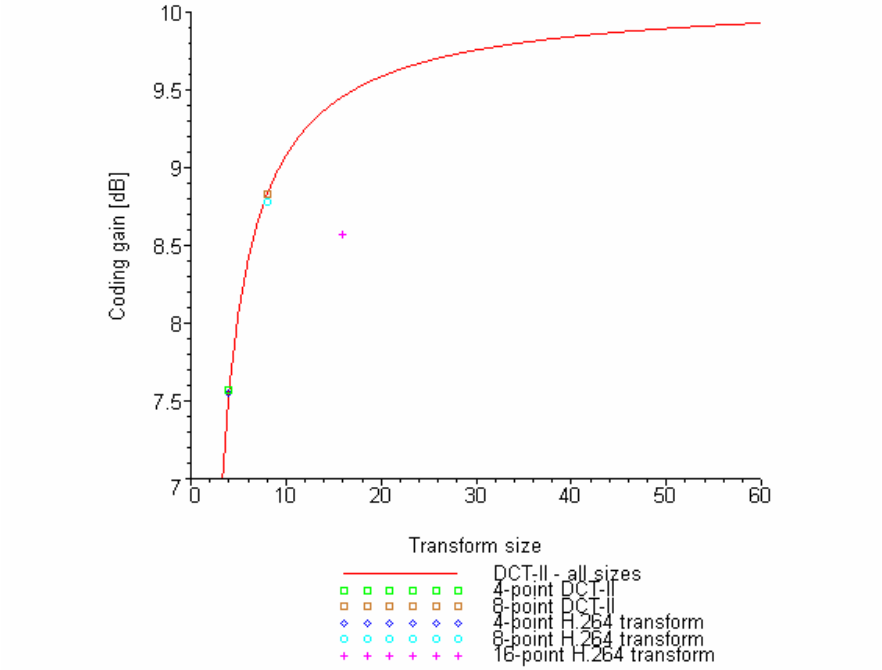


Fig. 5. Coding gain of DCT-II of arbitrary sizes and choices of transforms in current image and video coding standards (in all cases coding gain is measured for p=0.95).

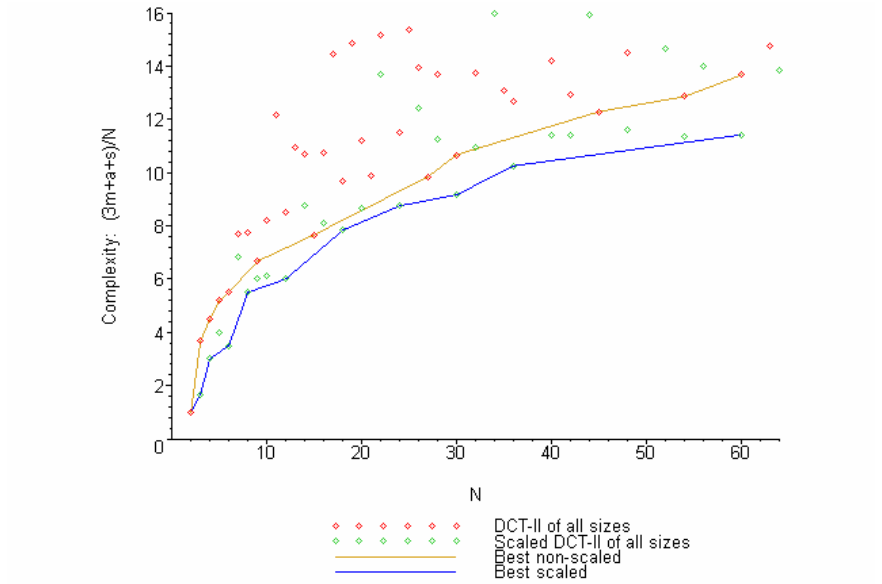


Fig 6. Analysis of complexity of DCT-II factorizations of sizes 2—64.

4. ANALYSIS OF COMPLEXITY AND PERFORMANCE OF TRANSFORMS OF VARIOUS SIZES

We summarize our complexity and coding gain analysis in Table 3.

As indicated earlier, we present are two types of complexity estimates: for complete transforms (with possibly removed one uniform factor for all coefficients) and for scaled ones. In both cases, these complexity estimates are obtained by

constructing factorizations of transforms of different sizes and counting all operations that they require. Since the process of construction of such factorizations is rather laborious, we move all the details (and descriptions of techniques that we've utilized) in Appendix A.

Table 3. Complexity vs Coding Gain of DCT-II transforms of sizes 2-64.

N	Prime Factors	Complexity upper bounds [m – multiplications, a – additions, s – shifts]				Coding Gain [dB]
		Non-scaled	(3m+a+s)/N	Scaled	(3m+a+s)/N	
2	2	2a	(1)	2a	(1)	5.05498
3	3	2m+4a+1s	(3.67)	4a+1s	(1.67)	6.73254
4	2^2	3m+9a	(4.5)	1m+9a	(3)	7.57013
5	5	4m+13a+1s	(5.2)	2m+13a+1s	(4)	8.07242
6	2,3	5m+16a+2s	(5.5)	1m+16a+2s	(3.5)	8.40724
7	7	8m+30a	(7.71)	6m+30a	(6.86)	8.64645
8	2^3	11m+29a	(7.75)	5m+29a	(5.5)	8.82591
9	3^2	8m+34a+2s	(6.67)	6m+34a+2s	(6)	8.96556
10	2,5	13m+40a+3s	(8.2)	6m+40a+3s	(6.1)	9.07734
11	11	20m+74a	(12.2)			9.16886
12	2^2,3	16m+49a+5s	(8.5)	6m+49a+5s	(6)	9.24518
13	13	20m+82a	(10.9)			9.30981
14	2,7	23m+80a+1s	(10.7)	14m+80a+1s	(8.79)	9.36526
15	3,5	14m+70a+3s	(7.67)			9.41335
16	2^4	30m+81a+1s	(10.8)	16m+81a+1s	(8.13)	9.45547
17	17	35m+141a	(14.5)			9.49268
18	2,3^2	25m+94a+5s	(9.67)	14m+94a+5s	(7.83)	9.52578
19	19	38m+168a	(14.8)			9.55542
20	2^2,5	36m+109a+7s	(11.2)	19m+109a+7s	(8.65)	9.58213
21	3,7	26m+130a	(9.9)			9.60633
22	2,11	51m+180a+1s	(15.2)	40m+180a+1s	(13.7)	9.62834
23	23	87m+314a	(25)			9.64847
24	2^3,3	44m+133a+11s	(11.5)	22m+133a+11s	(8.75)	9.66693
25	5^2	66m+186a	(15.4)			9.68394
26	2,13	53m+202a+1s	(13.9)	40m+202a+1s	(12.4)	9.69966
27	3^3	42m+140a	(9.85)			9.71423
28	2^2,7	60m+201a+3s	(13.7)	37m+201a+3s	(11.3)	9.72778
29	29	95m+355a	(22.1)			9.7404
30	2,3,5	43m+184a+7s	(10.7)	28m+184a+7s	(9.17)	9.7522
31	31	80m+358a	(19.3)			9.76325
32	2^5	76m+209a+3s	(13.8)	46m+209a+3s	(10.9)	9.77362
34	2,17	87m+332a+1s	(17.5)	70m+332a+1s	(16)	9.79256
35	5,7	53m+299a	(13.1)			9.80123
36	2^2,3^2	68m+241a+11s	(12.7)	39m+241a+11s	(10.3)	9.80943
37	37	110m+424a	(20.4)			9.8172
38	2,19	95m+392a+1s	(17.8)	76m+392a+1s	(16.3)	9.82457
40	2^3,5	92m+277a+15s	(14.2)	55m+277a+15s	(11.4)	9.83822
42	2,3,7	73m+322a+1s	(12.9)	52m+322a+1s	(11.4)	9.8506
44	2^2,11	124m+425a+3s	(18.2)	91m+425a+3s	(15.9)	9.86188
45	3^2,5	65m+358a	(12.3)			9.86715
46	2,23	197m+696a+1s	(28)	174m+696a+1s	(26.5)	9.87219
48	2^4,3	112m+337a+23s	(14.5)	66m+337a+23s	(11.6)	9.88167
50	2,5^2	157m+446a+1s	(18.4)	132m+446a+1s	(16.9)	9.89041
52	2^2,13	132m+481a+3s	(16.9)	93m+481a+3s	(14.7)	9.8985
54	2,3^3	111m+360a+1s	(12.9)	84m+360a+1s	(11.4)	9.906
56	2^3,7	148m+485a+7s	(16.7)	97m+485a+7s	(14)	9.91298
58	2,29	219m+796a+1s	(25.1)	190m+796a+1s	(23.6)	9.91949
60	2^2,3,5	116m+457a+15s	(13.7)	71m+457a+15s	(11.4)	9.92557
62	2,31	191m+808a+1s	(22.3)	160m+808a+1s	(20.8)	9.93128
63	3^2,7	98m+635a	(14.7)			9.934
64	2^6	184m+513a+7s	(16.8)	122m+513a+7s	(13.8)	9.93664

We present complexity plots for these transforms in Fig. 6.

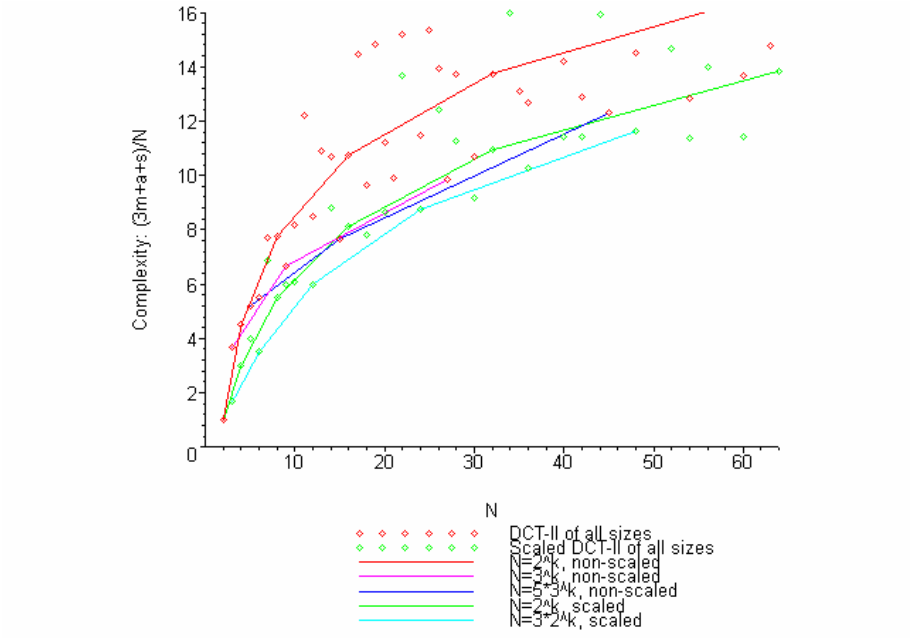


Fig 7. Complexity plots for transforms of suggested progressions of sizes.

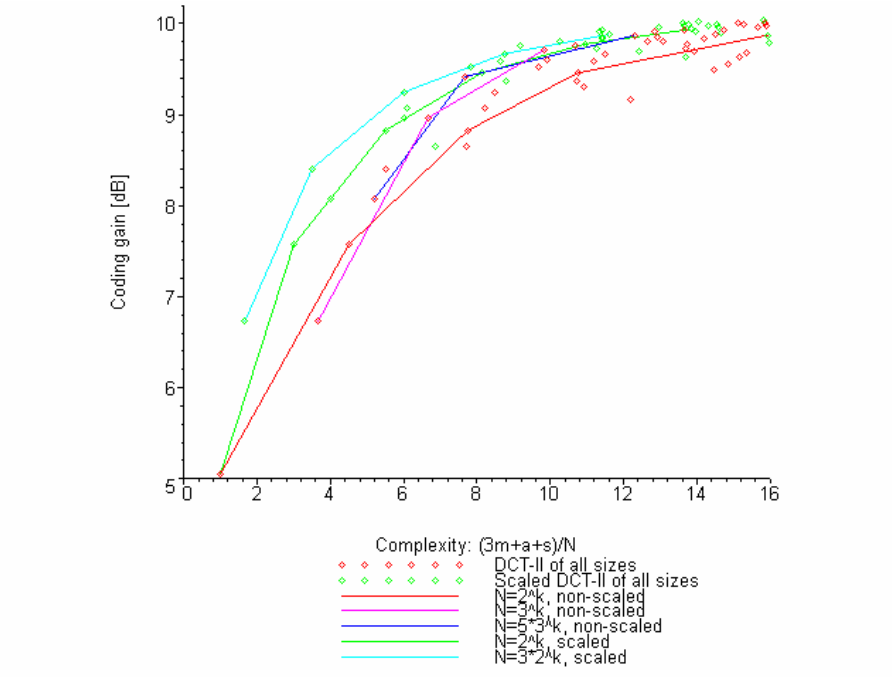


Fig. 8. Complexity vs. coding gain plots of transforms of suggested progressions.

In Fig. 6, we use blue line to show lowest complexity boundary attainable by scaled transforms and yellow line – to show complexity boundary attainable by non-scaled transforms. As expected, the scaled transforms turn out to be considerably less complex.

The transforms that form these boundaries have the following sizes:

- non-scaled: N=2,3,4,5,6,9,15,27,30,45,54,60
- scaled: N=2,3,4,6,8,12,18,24,30,36,60

Using these numbers we can suggest several candidate progressions of transform sizes:

- 5*3^k = 5,15,45,… -- subset of best for non-scaled transforms;
- 3^k = 3,9,27,… -- subset of best for non-scaled transforms;
- 3*2^k =3,6,12,24… -- subset of best scaled transforms, and
- 2^k= 2,4,8,16,… -- standard dyadic progression.

Such progressions are important for enabling image and video codecs to switch between transforms of different sizes, therefore adapting to statistics in images. Current standards, such as MPEG-4 AVC /H.264 are using dyadic progression for their transform sizes. The list above shows few extra choices.

We plot complexities of transforms along these suggested progressions in Fig. 7.

Finally, in Fig 8. we provide complexity vs coding gain analysis for transforms such sizes.

Based on the above plots, it becomes evident that for both scaled and non-scaled design cases, we can find transform sequences yielding much better coding gain vs complexity tradeoffs as compared to transforms with dyadic sizes.

Thus, for scaled transforms, most interest represents the sequence:

$$N=3*2^k = 3, 6, 12, 24, 48.$$

For non-scaled designs, most consistent gains can be observed for transforms with sizes:

$$N=5*3^k = 5, 15, 45.$$

Other sequences, such as

$$N=3^k = 3, 9, 27,$$

or

$$N=15*2^k = 15, 30, 60$$

migh also be of interest in non-scaled design cases.

The largest gap in coding gain at about same complexity can be observed between 8-point and 15-point non-scaled transforms: from 8.646 to 9.413 dB, which is 0.767 dB difference.

For scaled transforms, most dramatic jump is observed between 4- and 6-point transforms: from 7.57 to 8.407 dB, producing 0.837dB difference, although 6-point transforms are about 15% more complex.

Examples of specific transforms corresponding to our complexity estimates can be found in Appendix B. Techniques that we've utilized in deriving them are reviewed in Appendix A.

5. CONCLUSIONS

We have reviewed current transforms used in image and video coding and analyzed complexity – performance tradeoffs achievable by DCT-II transforms of sizes in the range 2—64. Our complexity estimates assumed reasonable costs of mapping into integer arithmetic, and were derived for both scaled and non-scaled IDCT designs.

Our analysis indicates that transforms of certain non-dyadic sizes can offer significantly better performance tradeoffs that ones currently achievable with 4, 8, and 16-point transforms.

REFERENCES

1. N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete Cosine Transform", IEEE Trans. Computers, Vol. X, pp. 90-93, Jan., 1974.
2. K. R. Rao, and P.Yip, "Discrete Cosine Transform: Algorithms, Advantages, Applications", Academic Press, San Diego, 1990.
3. V.Britanak, P.Yip, K.R.Rao, "Discrete Cosine and Sine Transforms: General Properties, Fast Algorithms and Integer Approximations", Academic Press, 2006.
4. H.S.Malvar, "Signal Processing with Lapped Transforms", Artech House, Norwood, MA, 1992.
5. C. Loeffler, A. Ligtenberg, and G. S. Moschytz. "Algorithm-architecture mapping for custom DCT chips." in Proc. Int. Symp. Circuits Syst. (Helsinki, Finland), June 1988, pp. 1953-1956.

6. Y. Arai, T.Agui and M. Nakajima, “A Fast DCT-SQ Scheme for Images”, Transactions of the IEICE E71(11): 1095, November 1988.
7. Y. A. Reznik, A. T. Hinds, N. Rijavec, “Low Complexity Fixed-Point Approximation of Inverse Discrete Cosine Transform”, Proc. IEEE International Conference on Acoustic, Speech, and Signal Proc. (ICASSP), Honolulu, HI, April 15 - 20, 2007.
8. Y. A. Reznik, A.T.Hinds, C.Zhang, L.Yu, and Z.Ni, “Efficient fixed-point approximations of the 8x8 inverse discrete cosine transform”, Proc. SPIE 6696, pp. 669617 1--17 (2007).
9. G.J. Sullivan, “Standardization of IDCT approximation behavior for video compression: the history and the new MPEG-C parts 1 and 2 standards”, Proc. SPIE 6696, pp. 669611 1--22 (2007).
10. ISO/IEC 10918-1 | ITU-T Rec. T.81, Information Technology – Digital Compression and Coding of Continuous-Tone Still Images – Requirements and Guidelines, (known as JPEG), 1992.
11. ISO/IEC 13818-2 | ITU-T Rec. H.262, Information Technology – Generic Coding of Moving Pictures and Associated Audio Information: Video, Version 1: 1995, Corrigendum 2: 1997 (known as MPEG-2 Video – various other amendments and corrigenda).
12. ITU-T Rec. H.263, Video Coding for Low Bit Rate Communication, Version 1: 1995/1996, Version 2: 1998, Version 3: 2000, Version 4: 2001, Version 4: 2005.
13. ISO/IEC 14496-2, Information Technology – Coding of Audio-Visual Objects – Part 2: Visual, Version 1, 2000, Amendment 1: 2000, Amendment 4: 2001.
14. ISO/IEC 14496-10 | ITU-T Rec. H.264, Advanced Video Coding for Generic Audiovisual Services, Version 1: May 2003, Version 2: May 2004, Version 3: March 2005, Version 4: Sept. 2005, Version 5: June 2006, Version 7: April 2007.
15. SMPTE Standard 421M-2006, VC-1 Compressed Video Bitstream Format and Decoding Process, Feb. 2006.
16. People’s Republic of China National Standard (AVS Working Group) GB/T 20090.2-2006, Information Technology – Advanced Coding of Audio and Video, Part 2: Video, March 2006.
17. S. Srinivasan, C. Tu, S. L. Regunathan, G. J. Sullivan, and R. A. Rossi, “HD Photo: A New Image Coding Technology for Digital Photography”, SPIE Applications of Digital Image Processing XXX, Proc. SPIE, Vol. 6696, Aug. 2007.
18. Z. Wu, T. Simono, H. Kitajima, and Y. Ogawa, “Modified Block Transform Coding of Images”, Trans. IECE, 1988, J71A, pp. 481-487 (in Japanese).
19. W.-K., Cham, "Development of integer cosine transforms by the principle of dyadic symmetry," Communications, Speech and Vision, IEE Proceedings I , vol.136, no.4, pp. 276-282, Aug 1989.
20. M.T. Heideman, “Computation of an Odd-Length DCT from a Real-Valued DFT of the Same Length”, IEEE Trans. Signal Proc., vol. 40, no. 1, Jan 1992.
21. Y.Reznik, R.K.Chivukula, “Fast 15x15 transform for image and video coding applications”, Proc Data Compression Conference (DCC), 2009, p 465.
22. M.T. Heideman, and C.S. Burrus, “On the number of multiplications necessary to compute a length-2^n DFT”, IEEE Trans. Acoust., Speech, Signal Process, vol ASSP-34, no 1, pp 91-95, Feb 1986.
23. E. Feig and S. Winograd, "On the multiplicative complexity of discrete cosine transforms (Corresp.)," IEEE Trans. Info. Theory, vol. IT-38, pp. 1387 - 1391, July 1992.
24. S. Winograd, “On Computing the Discrete Fourier Transform”. Proc. National Academy of Sciences, USA, 73, 1006–1006, April 1976.
25. S.C. Chan, and K.L.Ho, Direct methods for computing discrete sinusoidal transforms, Proc. IEE, Vol. 137, Pt. F, No. 6, Dec. 1990, pp. 433--442.
26. C.W. Kok, “Fast Algorithm for Computing Discrete Cosine Transform”, IEEE Trans. Signal Proc., vol.45, no.3, pp.757-760, Mar 1997.
27. E.Dubois, A.Venetsanopoulos , "A new algorithm for the radix-3 FFT," Acoustics, Speech and Signal Processing, IEEE Transactions on , vol.26, no.3, pp. 222-225, Jun 1978,
28. H.V. Sorensen et. al. “Real-Valued Fast Fourier Transform Algorithms”, IEEE Trans. ASSP, vol. 35, no. 6, June 1987.
29. C.S. Burrus, P.W. Eschenbacher, “An In-Place, In-Order Prime Factor FFT Algorithm”, IEEE Trans. ASSP, vol. 29, no. 4, Aug 1981.
30. M.T. Heideman, C.S. Burrus, H.W. Johnson, “Prime Factor FFT Algorithms for Real-Valued Series”, Proc. IEEE ICASSP, San Diego, March 1984.

31. D.P. Kolba, T.W. Parks, “A Prime Factor FFT Algorithm Using High-Speed Convolution”, IEEE Trans. ASSP, vol. 25, Aug 1977.
32. C.S. Burrus, T.W. Parks, “DFT/FFT and Convolution Algorithms – Theory and Implementation”, Wiley, New York, 1985.
33. H.J. Nussbaumer, “Fast Fourier Transform and Convolution Algorithms”, Springer, 1981.
34. H.F. Silverman, “An Introduction to Programming the Winograd Fourier Transform Algorithm (WFTA)”, IEEE Trans. ASSP, vol. 25, no. 2, April 1977.
35. B.D. Tseng, W.C. Miller, “Comments on ‘An Introduction to Programming the Winograd Fourier Transform Algorithm (WFTA)’ ”, IEEE Trans. ASSP, vol. 26, no. 3, June 1978.

APPENDIX A: FACTORIZATION TOOLS AND TECHNIQUES USED IN THIS WORK

A.0 Notation

Let $\{x_n\}$, $n = 0,1,...,N-1$ be an input real-valued sequence.

In what follows, we will be dealing with the following transforms:

- DCT-II and its inverse:

$$X_k^{II} = \sqrt{\frac{2}{N}}\lambda(k)\sum_{n=0}^{N-1} x_n \cos\left(\frac{(2n+1)\,k\pi}{2N}\right), \quad k = 0,1,....,N-1,$$

$$x_n = \sqrt{\frac{2}{N}}\sum_{k=0}^{N-1} X_k^{II}\lambda(k)\cos\left(\frac{(2n+1)\,k\pi}{2N}\right), \quad n = 0,1,....,N-1,$$

  where $\lambda(k) = 1/\sqrt{2}$, if $k = 0$, otherwise 1.

- DCT-III and its inverse:

$$X_n^{III} = \sqrt{\frac{2}{N}}\sum_{k=0}^{N-1} x_k\lambda(k)\cos\left(\frac{(2n+1)\,k\pi}{2N}\right), \quad n = 0,1,....,N-1,$$

$$x_k = \sqrt{\frac{2}{N}}\lambda(k)\sum_{n=0}^{N-1} X_n^{III}\cos\left(\frac{(2n+1)\,k\pi}{2N}\right), \quad k = 0,1,....,N-1,$$

  where $\lambda(k) = 1/\sqrt{2}$, if $k = 0$, otherwise 1.

- DCT-IV and its inverse:

$$X_k^{IV} = \sqrt{\frac{2}{N}}\sum_{n=0}^{N-1} x_n \cos\left(\frac{\pi}{4N}(2n+1)(2k+1)\right), \quad k = 0,1,....,N-1,$$

$$x_k = \sqrt{\frac{2}{N}}\sum_{n=0}^{N-1} X_k^{IV}\cos\left(\frac{\pi}{4N}(2n+1)(2k+1)\right), \quad n = 0,1,....,N-1,$$

For further convenience, we omit normalization factors ($\sqrt{2/N}$ and $\lambda(k)$) and define matrices:

$$C_N^{II}(n,k) = \cos\left(\frac{\pi(2n+1)k}{2N}\right), \quad n,k = 0,...,N-1,$$

$$C_N^{III}(n,k) = \cos\left(\frac{\pi n(2k+1)}{2N}\right), \quad n,k = 0,...,N-1,$$

$$C_N^{IV}(n,k) = \cos\left(\frac{\pi(2n+1)(2k+1)}{4N}\right), \quad n,k = 0,...,N-1$$

representing coefficients of DFT, DCT-II, DCT-III, and DCT-IV transforms correspondingly.

A.1 Connection between DCT-II, DCT-III, and DCT-IV

We immediately notice that DCT-III is simply an inverse (transpose) of DCT-II:

$$C_N^{III} = \left(C_N^{II}\right)^T = \left(C_N^{II}\right)^{-1} \tag{A.1}$$

and that DCT-IV is involutory (self-inverse, self-transpose):

$$C_N^{IV} = \left(C_N^{IV}\right)^T = \left(C_N^{IV}\right)^{-1}. \tag{A.2}$$

Additionally, as it was shown by S.C. Chan, and K.L.Ho[25], as well as C.W.Kok[26], DCT-IV can be reduced to DCT-II by using the following transformation:

$$C_N^{IV} = R_N C_N^{II} D_N \tag{A.3}$$

where R_N is a matrix of recursive subtractions:

$$R_N = \begin{pmatrix} \frac{1}{2} & 0 & 0 & \cdots & 0 \\ -\frac{1}{2} & 1 & 0 & \cdots & 0 \\ \frac{1}{2} & -1 & 1 & \cdots & 0 \\ -\frac{1}{2} & 1 & -1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -\frac{1}{2} & 1 & -1 & \cdots & 1 \end{pmatrix} \tag{A.4}$$

and D_N is a diagonal matrix of factors:

$$D_N = \mathrm{diag}\left(2\cos\left(\tfrac{\pi}{4N}\right), 2\cos\left(\tfrac{3\pi}{4N}\right), \ldots, 2\cos\left(\tfrac{(N+1)\pi}{4N}\right)\right). \tag{A.5}$$

We show flowgraph of this mapping in Figure A.1.

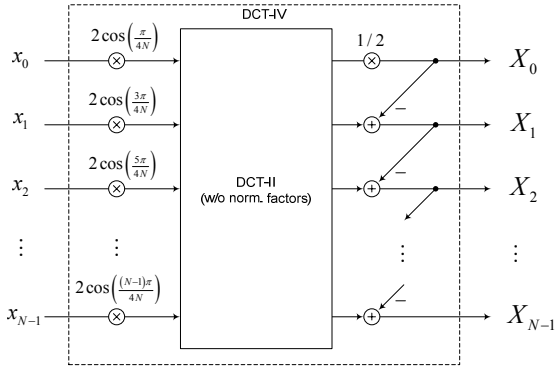


Fig. A.1. DCT-IV via DCT-II computation.

We note that if full DCT-IV and DCT-II are used (instead of core transforms with omitted normalization factors), then a factor $1/2$ in this diagram will need to be replaced by $1/\sqrt{2}$.

From this decomposition it follows that DCT-IV of size N has at most the same complexity as DCT-II plus N multiplications, N-1 additions, and 1 multiplication by factor $1/2$ (or $1/\sqrt{2}$).

A.2 Split of DCT-II into half-sized DCT-II and DCT-IV

It is well known that even-sized DCT-II matrix (with omitted normalization factors) can be factored into a product containing direct sum of smaller DCT-II and DCT-IV matrices as follows[2,3]:

$$C_N^{II} = P_N \begin{pmatrix} C_{N/2}^{II} & 0 \\ 0 & C_{N/2}^{IV} J_{N/2} \end{pmatrix} \begin{pmatrix} I_{N/2} & J_{N/2} \\ J_{N/2} & -I_{N/2} \end{pmatrix}. \tag{A.6}$$

Where P_N is a permutation matrix producing reordering:

$$x'_i = x_{2i}, \;\; x'_{N/2+i} = x_{2i+1}, \quad i = 0,1,...,N/2-1,$$

and where $I_{N/2}$ and $J_{N/2}$ denote $N/2 \times N/2$ identity and order reversal matrices correspondingly.

The decomposition (1) is of fundamental importance, and it was the basis for many pioneering works on fast DCT-II algorithms, including factorization of Chen-Smith-Fralick, algorithms of Wang, and others[2].

A.3 Working with DCT-II of dyadic sizes

This is the most studied class of transforms, for which we simply use existing algorithms[2,3].

The theoretical estimates of minimum necessary number of multiplications by irrational factors (multiplicative complexity) necessary to compute such transforms are given in[22,23]:

$$\mu(2^r) = 2^{r+1} - r - 2;$$

The paper by Feig and Winograd[23] also studies multiplicative complexity of scaled dyadic-sized DCT-II, but they only derive upper bound:

$$\tilde{\mu}(2^r) \le 2^{r+1} - \frac{r(r+3)}{2} - 2;$$

Same paper also includes lower bound (which, actually is valid for any size N):

$$\tilde{\mu}(N) \ge \mu(N) - N;$$

A.4 Decimation of DCT-II of even sizes

A combination of factorization (6) with representation of DCT-IV (3) yields the following recursive process for computing DCT-II or even sizes:

$$C_N^{II} = P_N \begin{pmatrix} C_{N/2}^{II} & 0 \\ 0 & R_{N/2} C_{N/2}^{II} D_{N/2} J_{N/2} \end{pmatrix} \begin{pmatrix} I_{N/2} & J_{N/2} \\ J_{N/2} & -I_{N/2} \end{pmatrix}. \tag{A.7}$$

This algorithm was first derived by C.W. Kok[26], and it is was shown to be more efficient than other known DCT-II decimation techniques in terms of multiplication complexity[26].

We show flow-graph of 10-point DCT-II split using C.W.Kok's algorithm in Figure A.2.

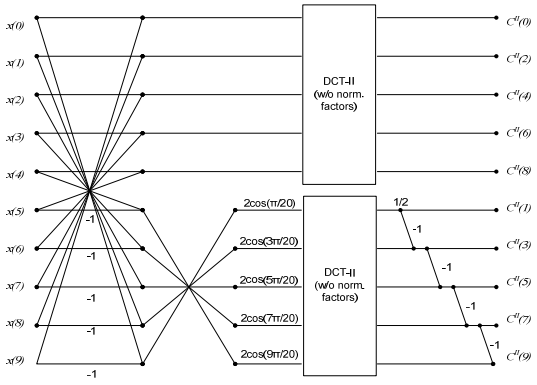


Fig A.2. Decimation of DCT-II using C.W.Kok's algorithm.

It can be easily shown, that given an N-point DCT-II with total complexity of

$$\alpha\, m + \beta\, a + \gamma\, s$$

where m- denotes multiplications, a – additions, and s - shifts, this algorithm produces a transform of size 2N with complexity:

$$(2\alpha + N)m + (2\beta + 3N - 1)a + (2\gamma + 1)s.$$

We note that for transforms of dyadic sizes there are many other decimation techniques[2,3], but this particular one is more general, as it allows decimation of DCT-II with any even (not necessarily dyadic) sizes.

A.5 Factorization of scaled DCT-II of even sizes

By involutary property of DCT-IV we can rewrite (A.3) as:

$$C_N^{IV} = \left(R_N C_N^{II} D_N\right)^T = D_N \left(C_N^{II}\right)^T R_N^T = D_N C_N^{III} R_N^T \tag{A.11}$$

Next, (following same argument as in our derivation of C.W. Kok's algorithm) we substitute the DCT-IV block in factorization (A.6) with formula (A.11).

This leads us to the following alternative decimation technique:

$$C_N^{II} = P_N \begin{pmatrix} C_{N/2}^{II} & 0 \\ 0 & D_{N/2} C_{N/2}^{III} R_{N/2}^T J_{N/2} \end{pmatrix} \begin{pmatrix} I_{N/2} & J_{N/2} \\ J_{N/2} & -I_{N/2} \end{pmatrix}. \tag{A.12}$$

One important distinction of this modification is that all factors represented by matrix $D_{N/2}$ are now moved in the last stage of the transform, and can therefore be separated and possibly merged with some other steps (such as quantization, or scaling) done after transform in the application. As obvious, such separation reduces complexity of the remaining scaled transform.

A.5.1 Decimation of scaled DCT-II

In general, by scaled DCT-II transform we will understand factorization:

$$C_N^{II} = \Pi_N \Delta_N \tilde{C}_N^{II}. \tag{A.13}$$

where Π_N is a matrix defining reordering of coefficients, Δ_N is a diagonal matrix of scale factors, and $\tilde{C}_N^{II}$ is a matrix of the remaining scaled transform.

With these notations in mind, we can reformulate (A.12) as follows:

$$\Pi_N \Delta_N \tilde{C}_N^{II} = P_N \begin{pmatrix} \Pi_{N/2} \Delta_{N/2} \tilde{C}_{N/2}^{II} & 0 \\ 0 & D_{N/2} C_{N/2}^{III} R_{N/2}^T J_{N/2} \end{pmatrix} \begin{pmatrix} I_{N/2} & J_{N/2} \\ J_{N/2} & -I_{N/2} \end{pmatrix}. \tag{A.14}$$

producing the following algorithm for computing scaled DCT-II transforms:

$$\Pi_N = P_N \begin{pmatrix} \Pi_{N/2} & 0 \\ 0 & I_{N/2} \end{pmatrix}, \quad \Delta_N = \begin{pmatrix} \Delta_{N/2} & 0 \\ 0 & D_{N/2} \end{pmatrix}, \tag{A.15}$$

$$\tilde{C}_N^{II} = \begin{pmatrix} \tilde{C}_{N/2}^{II} & 0 \\ 0 & C_{N/2}^{III} R_{N/2}^T J_{N/2} \end{pmatrix} \begin{pmatrix} I_{N/2} & J_{N/2} \\ J_{N/2} & -I_{N/2} \end{pmatrix}. \tag{A.16}$$

This process can be used to derive scaled transforms of sizes: $N = N_1 2^k$, where $N_1 >= 2$.

Now, given the complexity estimates for full and scaled DCT-II modules of size N:

Full DCT-III: $\alpha\, m + \beta\, a + \gamma\, s$

Scaled DCT-II: $\delta\, m + \varepsilon\, a + \varsigma\, s$

where m- denotes multiplications, a – additions, and s - shifts, we can show that complexity of computing the scaled DCT-II of size 2N using algorithm (16) is:

$$(\alpha + \delta) m + (\beta + \varepsilon + 3N - 1) a + (\gamma + \varsigma + 1) s. \tag{A.17}$$

A.6 Derivation of factorizations of DCT-II of odd sizes

In derivation of factorizations of DCT-II of odd sizes we follow several known techniques, such as Heideman mapping of DCT-II to DFT[20], Winograd short length DFT modules[24,35], as well as prime factor algorithms (PFA) and common factor algorithms (CFA) for computing real valued FFTs of factorizable lengths[30-34].

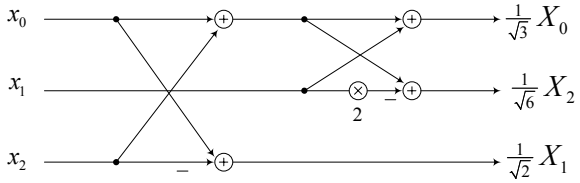
In our work we have tried various permutations of these techniques and picked solutions that are least complex.

APPENDIX B: EXAMPLES OF FAST FACTORIZATIONS

Here we will list a few (particularly interesting from complexity-point of view) examples of transforms that we have produced as part of our analysis.

3-point DCT-II:

Allows the following factorization:

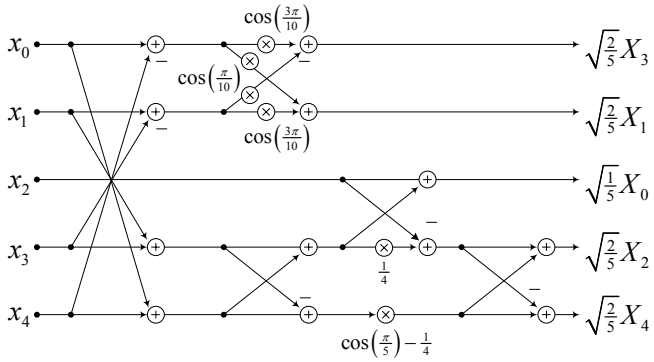


Scaled transform can be implemented by using only 4 additions and 1 shift (multiplication by factor of 2).

This particular factorization is less complex than one previously reported by Heideman [20], which required 5 additions and 1 shift (extra addition due to factor 1.5).

5-point DCT-II:

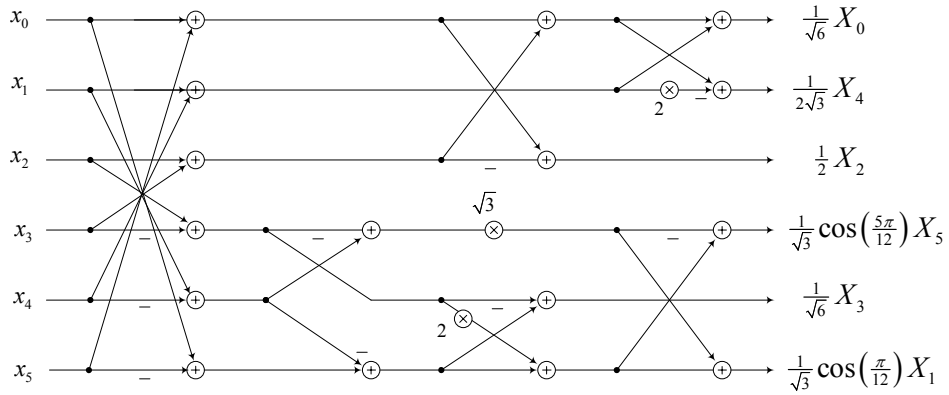
Allows the following factorization:



This flowgraph includes 5 multiplications, 12 additions, and 1 shift. We note that this factorization is less complex than one previously reported by Heideman [20], which required 13 additions, 5 multiplications, and 1 shift (one extra addition due to factor 1.25).

6-point DCT-II:

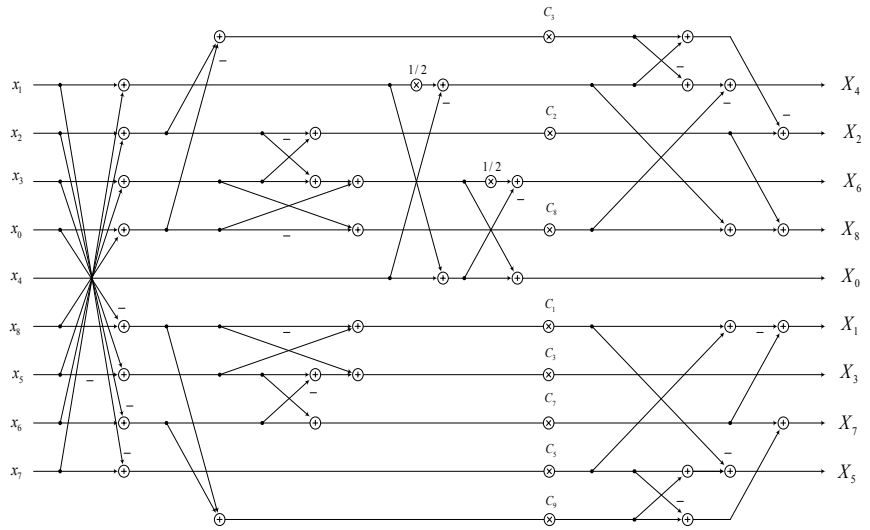
Allows the following scaled factorization:



This transform needs only 1 multiplication 16 additions and 2 shifts.

This and other scaled transforms of sizes N=3*2^k are obtained by using process described in section A.5.1 in the appendix A.

9-point DCT:
Allows the following factorization[20]:



This factorization includes 8 multiplications by factors:

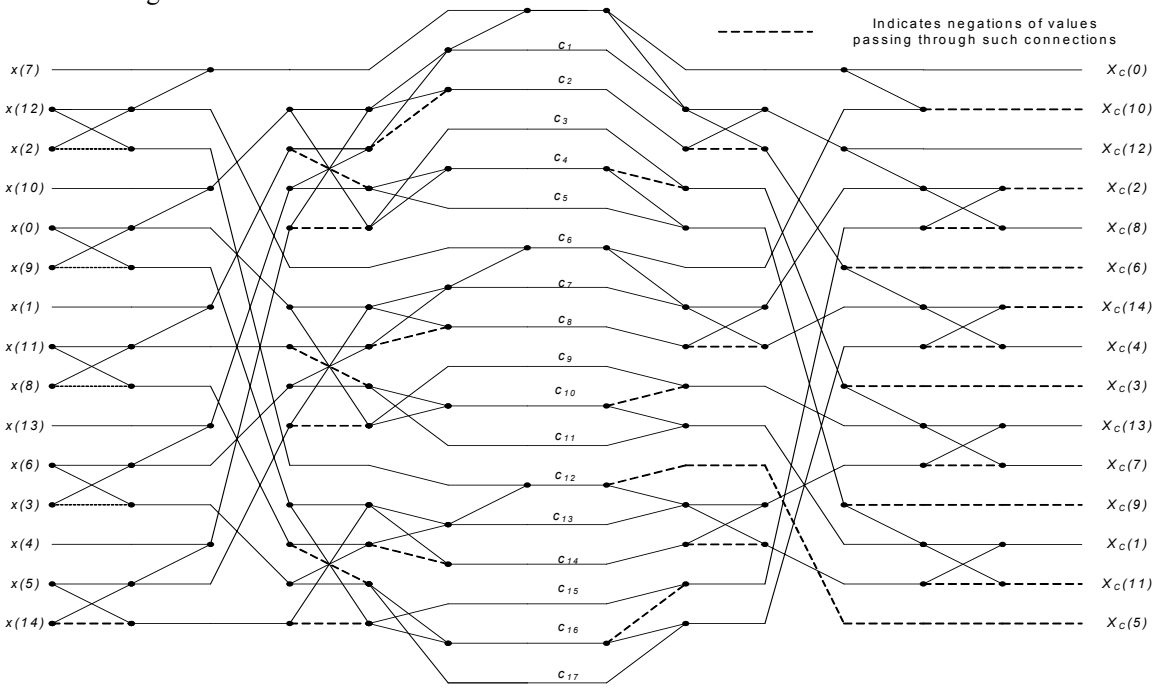
$$C_2 = -\cos\left(\tfrac{2\pi}{9}\right), C_7 = -\sin\left(\tfrac{2\pi}{9}\right);\ C_3 = C_5 = -\sin\left(\tfrac{\pi}{3}\right);$$

$$C_1 = -\sin\left(\tfrac{4\pi}{9}\right),\ C_8 = -\cos\left(\tfrac{4\pi}{9}\right);\ \ C_{21} = -\cos\left(\tfrac{8\pi}{9}\right);\ C_{71} = -\sin\left(\tfrac{8\pi}{9}\right);$$

as well 34 additions and 2 shifts.

We note that this transform is actually less complex (by about 16% using our normalized complexity metric) that 8-point DCT-II.

15-point DCT:
Allows the following non-scaled factorization[21]:



This factorization includes 17 factors ($\alpha = \frac{-2\pi}{5}; \quad \beta = \frac{-2\pi}{3};$):

$$c_1 = \tfrac{1}{2}\left[\cos\alpha + \cos 2\alpha\right] - 1; \quad c_2 = \tfrac{1}{2}\left[\cos\alpha - \cos 2\alpha\right]; \quad c_3 = \sin\alpha + \sin 2\alpha; \quad c_4 = \sin 2\alpha; \quad c_5 = \sin\alpha - \sin 2\alpha;$$

$$c_6 = \cos\beta - 1; \quad c_7 = c_1 c_6; \quad c_8 = c_2 c_6; \quad c_9 = c_3 c_6; \quad c_{10} = c_4 c_6; \quad c_{11} = c_5 c_6;$$

$$c_{12} = \sin\beta; \quad c_{13} = c_1 c_{12}; \quad c_{14} = c_2 c_{12}; \quad c_{15} = -c_3 c_{12}; \quad c_{16} = -c_4 c_{12}; \quad c_{17} = -c_5 c_{12};$$

among which 3 are dyadic rational numbers:

$$c_1 = -\tfrac{5}{4}; \quad c_6 = -\tfrac{3}{2}; \quad c_7 = \tfrac{15}{8};$$

This means, that this factorization requires only 14 essential multiplications and 67 additions.

In terms of normalized average complexity this transform is about as complex, as 8-point DCT-II, while offering extra 0.767 dB in coding gain. Compared to 16-point DCT-II, this transform is almost two-times less complex.