```
1)    E  ::=  z
2)    E  ::=  T
3)    T  ::=  T  y
4)    T  ::=  y
```

Fig. 1. Grammar G.

	Symbol				
State	y	z	⊥	E	T
1	'4	'1		2	3
2			a		
3	'3		-2		

```
Legend:    'p = read/apply prod'n p
           -p = apply prod'n p
            p = go tc state p
            a = accept
```

Fig. 2. $SLR(1)$ parser table for G.

	Symbol				
State	y	z	⊥	E	T
1	'4	'1		2	2
2	'3		a		

Fig. 3. Modified tables for G.

	Symbol				
State	y	z	⊥	E	T
1	3	2		2	3
2			a		
3	'3		-2		

Fig. 4. Parser for G with two productions eliminated.

fied algorithm is applicable to parser tables produced by *SLR, LALR,* or *LR* generating techniques, whereas Aho and Ullman's algorithm may be applied only to *SLR* and *LR* techniques. Thus the algorithm presented here may be retrofitted into practical parser generators such as Lalonde's *LALR* generator [9], and DeRemer's *SLR* generator [7]. Its implementation in an in-house *LALR*(1) system [11] was straightforward, and through its use, the algorithm has been shown to be effective. For example, in the *XPL* grammar, there is a sequence of unit productions

⟨expression⟩ : : = ⟨logical factor⟩ : : = ... : : = ⟨primary⟩
where ⟨primary⟩ : : = ⟨constant⟩
| ⟨variable⟩
⟨constant⟩ : : = ⟨string⟩
| ⟨number⟩.

The maximum depth of the derivation tree is 9, and involves 11 productions. The algorithm applied to this sequence eliminated 7 of the productions and reduced the depth of the derivation tree to 4. The eliminated productions accounted for 34 percent of the total number of productions applied in the parse of the *XPL* compiler [12] and included those for ⟨primary⟩ and ⟨constant⟩ above. The simplified algorithm would thus make possible a significant savings in the total parse time of the *XPL* compiler.

IV. Conclusion

A simple algorithm has been described for the partial elimination of unit productions. The algorithm may be used to advantage in currently available practical *LR*(1) parser generators with a minimum of installation effort.

References

[1] A. V. Aho and J. D. Ullman, "A technique for speeding up *LR*(k) parsers," *SIAM J. Comput.*, vol. 2, pp. 106–127, June 1973.
[2] T. Anderson, "Syntactic analysis of *LR*(k) languages," Ph.D. dissertation, Univ. Newcastle-upon-Tyne, Newcastle-upon-Tyne, England, 1972.
[3] T. Anderson, J. Eve, and J. J. Horning, "Efficient *LR*(1) parsers," *Acta Inform.*, vol. 2, pp. 12–39, 1973.
[4] D. Pager, "On eliminating unit productions from *LR*(k) parsers," Inform. Sci. Program, Univ. Hawaii, Honolulu, Tech. Rep. PE-238, 1972.
[5] M. L. Joliat, "On the reduced matrix representation of *LR*(k) parser tables," Ph.D. dissertation, Comput. Syst. Res. Group, Univ. Toronto, Toronto, Ont., Canada, Tech. Rep. CSRG-28, Oct. 1973.
[6] F. L. DeRemer, "Simple *LR*(k) grammars," *Commun. Ass. Comput. Mach.*, vol. 14, pp. 453–460, July 1971.
[7] ——, "Practical translators for *LR*(k) languages," Ph.D. dissertation, Massachussets Inst. Technol., Cambridge, Sept. 1969.
[8] D. Pager, "A solution to an open problem by Knuth," *Inform. Contr.*, vol. 17, pp. 463–473, Dec. 1970.
[9] W. R. Lalonde, "An efficient *LALR* parser generator," Comput. Syst. Res. Group, Univ. Toronto, Toronto, Ont., Canada, Tech. Rep. CSRG-2, 1971.
[10] M. L. Joliat, "Practical minimization of *LR*(k) parser tables," in *IFIP Proc.* Amsterdam, The Netherlands: North-Holland, 1974, pp. 376–380.
[11] ——, "The BIGLALR parser generator system," Bell-Northern Res. documentation, Jan. 27, 1975.
[12] W. G. Alexander, "How a programming language is used," Comput. Syst. Res. Group, Univ. Toronto, Toronto, Ont., Canada, Tech. Rep. CSRG-10, Feb. 1972.

A Storage Efficient Way to Implement the Discrete Cosine Transform

ROBERT M. HARALICK

***Abstract*—Ahmed has shown that a discrete cosine transform can be implemented by doing one double length fast Fourier transform (FFT). In this correspondence, we show that the amount of work can be cut to doing two single length FFT's.**

***Index Terms*—Data compression, fast Fourier transform (FFT), fast transform.**

Ahmed *et al.* [1] introduced the discrete cosine transform as one whose basis vectors approximate the eigenvectors of a correlation matrix of a first-order Markov process. [This actually is well known (Gray [2] and Shanmugam, [4]).] If to first-order approximations, images are realizations of first-order Markov properties; then this implies that the discrete cosine transform is a better choice of transforms to use than the discrete Fourier transform (DFT) in image transform coding compression work. He also suggested a fast implementation of the discrete cosine transform using the fast Fourier transform (FFT) on a data vector whose length had been doubled by adding zeros. Schaming [3] defines a related transform using the FFT on a data vector which is doubled by appending its mirror image. In this correspondence we illustrate how to implement the discrete cosine transform by using the FFT on 2 data vectors, each of the original data sequence length. This is not only more economical storagewise, but

Manuscript received July 25, 1975; revised January 7, 1976. This work was supported by the National Science Foundation under Grant ENG74-18981.

The author is with the Remote Sensing Laboratory, University of Kansas, Lawrence, KS 66045.

it is also slightly more efficient computationally. Proofs of the stated lemmas are direct and are, therefore, not illustrated.

Equations (1) and (2) give the discrete cosine transform pair in a slightly different form than given by Ahmed.

$$\hat{c}_n = \sum_{k=0}^{K-1} c_k \cos\left(\frac{\pi n(2k+1)}{2K}\right), \qquad n = 0, \cdots, K-1 \quad (1)$$

$$c_k = \frac{2}{K}\sum_{n=0}^{K-1} \hat{c}_n \cos\left(\frac{\pi n(2k+1)}{2K}\right) - \frac{\hat{c}_o}{K}, \qquad k = 0, \cdots, K-1. \quad (2)$$

In the first lemma, we illustrate how the FFT of an even data sequence relates to the discrete cosine transform. The lemma says that for the discrete cosine transform of the first half, the even data sequence can be obtained by taking the FFT of the full data sequence, all-pass filtering by $e^{j\pi n/2K}$, and scaling by ½. This suggests an implementation of the discrete cosine transform by doubling the K-length data sequence (appending its mirror image) and using the FFT on the $2K$-length data sequence to obtain the discrete cosine transform.

Lemma 1: Let $x_o, \cdots, x_{2K-1}$ be real and satisfy $x_k = x_{2K-k-1}$. Let

$$\hat{x}_n = \sum_{k=0}^{2K-1} x_k e^{2j\pi nk/2K}.$$

Then

$$\frac{\hat{x}_n e^{j\pi n/2K}}{2} = \sum_{k=0}^{K-1} x_k \cos\frac{\pi n(2k+1)}{2K}.$$

Lemma 2 indicates that the FFT of a real even data sequence of length $2K$ can be obtained by taking 2 FFT's of data sequences of length K. Since two FFT's of order k require $2K\log K$ operations and one FFT of order $2K$ requires $(2K)\log(2K)$ operations, there is a computational savings of $2K$ operations and a storage saving of K memory locations using the two shorter FFT's in place of the one longer FFT.

Lemma 2: Let $x_o, \cdots, x_{2K-1}$ be real and satisfy $x_k = x_{2K-1-k}$. Let

$$c_k = x_k, \qquad k = 0, 1, \cdots K-1$$
$$d_k = x_k e^{j\pi k/K}, \qquad k = 0, 1, \cdots, K-1.$$

Define

$$\hat{x}_n = \sum_{k=0}^{2K-1} x_k e^{2j\pi nk/2K}$$
$$\hat{c}_n = \sum_{k=0}^{K-1} c_k e^{2j\pi nk/K}$$
$$\hat{d}_n = \sum_{k=0}^{K-1} d_k e^{2j\pi nk/K}.$$

Then

$$\hat{x}_{2n} = \hat{c}_n + e^{-2j\pi n/K}\hat{c}_n^*$$
$$\hat{x}_{2n+1} = \hat{y}_n + e^{-j\pi(2n+1)/K}\hat{y}_n^*.$$

Putting Lemmas 1 and 2 together we summarize as follows:

$$\text{Re}\{\hat{c}_n e^{j\pi n/K}\} = \sum_{k=0}^{K-1} c_k \cos\frac{\pi(2n)(2k+1)}{2K}$$
$$\text{Re}\{\hat{y}_n e^{j\pi(2n+1)/2K}\} = \sum_{k=0}^{K-1} c_k \cos\frac{\pi(2n+1)(2k+1)}{2K}$$

A similar set of properties hold for the inverse discrete cosine transform. Lemma 3 illustrates how an FFT on a double length sequence can be used to compute the inverse discrete cosine transform of a given data sequence.

Lemma 3: Let $\hat{c}_0, \cdots, \hat{c}_{K-1}$ be real. Let

$$\hat{x}_n = \hat{c}_n e^{-j\pi n/2K}, \qquad n = 0, \cdots, K-1$$
$$= \hat{c}_{2K-n} e^{-j\pi n/2K}, \qquad n = K+1, \cdots, 2K-1$$
$$= 0, \qquad n = K.$$

Define

$$x_k = \frac{1}{2K}\sum_{n=0}^{2K-1} \hat{x}_n e^{-2j\pi nk/2K}.$$

Then

$$2x_k = \frac{2}{K}\sum_{n=0}^{K-1} \hat{c}_n \cos\frac{\pi n(2k+1)}{2K} - \frac{\hat{c}_o}{K}.$$

It also turns out that the doubled data sequence required to FFT has a property which makes it amenable to the FFT by using two FFT's on original length sequences as stated in Lemma 4. This allows a savings of $2K$ operations and a storage savings of K memory locations.

Lemma 4: Let $x_0, \cdots, x_{2K-1}$ satisfy $\hat{x}_n = \hat{x}_{2K-n}$, $n = 1, \cdots, 2K-1$ and $\hat{x}_K = 0$. Let

$$\hat{c}_n = x_n, \qquad n = 0, \cdots, K-1$$
$$\hat{d}_n = x_n e^{-j\pi n/K}, \qquad n = 0, \cdots, K-1.$$

Define

$$c_k = \frac{1}{K}\sum_{n=0}^{K-1} \hat{c}_n e^{-2j\pi nk/K}$$
$$d_k = \frac{1}{K}\sum_{n=0}^{K-1} \hat{d}_k e^{-2j\pi nk/K}$$
$$x_k = \frac{1}{2K}\sum_{n=0}^{2K-1} \hat{x}_n e^{-2j\pi nk/2K}.$$

Then

$$x_{2k} = \text{Re}\{c_k\} - \frac{\hat{c}_o^*}{2K}$$
$$x_{2k+1} = \text{Re}\{d_k\} - \frac{\hat{c}_o^*}{2K}.$$

Putting Lemmas 3 and 4 together for a real data sequence, we obtain

$$2\,\text{Re}\{c_k\} - \frac{\hat{c}_o}{K} = \frac{2}{K}\sum_{n=0}^{K-1} \hat{x}_n \cos\frac{\pi 2n(2k+1)}{2K}$$
$$2\,\text{Re}\{d_k\} - \frac{\hat{c}_o}{K} = \frac{2}{K}\sum_{n=0}^{K-1} \hat{x}_n \cos\frac{\pi(2n+1)(2k+1)}{2K}.$$

In summary, we have shown how to compute the discrete cosine transform or its inverse by using two FFT's on the original length data sequence.

References

[1] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transform," *IEEE Trans. Comput.* (Corresp.), vol. C-23, pp. 90–93, Jan. 1974.

[2] R. W. Gray, "On the asymptotic eigenvalue distribution of Toeplitz matrices," *IEEE Trans. Inform. Theory,* vol. IT-18, pp. 725–730, Nov. 1972.

[3] W. B. Schaming, "Digital image transform encoding," RCA Advanced Technol. Lab., Camden, NJ, Case Rep. PE-622, 1974.

[4] K. Shanmugam, "Comments on discrete cosine transforms," *IEEE Trans. Comput.,* vol. C-24, p. 759, July 1975.