space methods of pattern recognition which combine the role of feature selection and pattern classification. The method of Chien and Fu [10] is suitable for compressing second-order statistical information about patterns. Watanabe's Selfic [11] is ideal for feature selection in nonsupervised pattern recognition, etc. Since in these specific cases such techniques are better than any other procedure it would be inappropriate to attempt to order the methods according to some subjective criterion of feature selection effectiveness. Instead, they should always be discussed in the objective terms of their individual applicability.

IV. Conclusions

This correspondence discussed the relationship of the discriminant vector method of feature selection recently proposed by Foley and Sammon and the method of Kittler and Young. Although both methods determine the feature space coordinate axes by maximizing the Fisher criterion of discriminatory power, the resulting feature spaces are considerably different because of the difference in the constraints imposed on the axes by individual methods. It has been shown that the latter method is, from the point of view of dimensionality reduction, more powerful and also computationally more efficient.

References

[1] D. H. Foley and J. W. Sammon, "An optimal set of discriminant vectors," *IEEE Trans. Comput.*, vol. C-24, pp. 281–289, Mar. 1975.
[2] S. Watanabe, "Karhunen–Loeve expansion and factor analysis," *Trans. 4th Prague Conf. Information Theory*, pp. 635–660, 1965.
[3] K. Fukunaga and W. L. G. Koontz, "Application of the Karhunen–Loeve expansion to feature selection and ordering," *IEEE Trans. Comput.*, vol. C-19, pp. 311–318, Apr. 1970.
[4] J. Kittler, "Mathematical methods of feature selection in pattern recognition," *Int. J. Man-Machine Studies*, vol. 7, pp. 609–637, 1975.
[5] J. Kittler and P. C. Young, "A new approach to feature selection based on the Karhunen–Loeve expansion," *Pattern Recognition*, vol. 5, pp. 335–352, Dec. 1973.
[6] J. Kittler, "A method of feature selection for high dimensional pattern recognition problems," in *Proc. 2nd Int. Conf. on Pattern Recognition*, pp. 48–51, 1974.
[7] ——, "Feature selection methods based on the Karhunen–Loeve expansion—A review," in *Proc. Int. Comput. Symp.*, pp. 379–385, Taipei, 1975.
[8] S. Watanabe, *Knowing and Guessing*. New York: Wiley, 1969.
[9] S. Watanabe and N. Pakvasa, "Subspace method in pattern recognition," in *Proc. 1st Int. Conf. Pattern Recognition*, pp. 25–32, Washington, 1973.
[10] Y. T. Chien and K. S. Fu, "On the generalized Karhunen–Loeve expansion," *IEEE Trans. Inform. Theory*, vol. IT-13, pp. 518–520, July 1967.
[11] S. Watanabe, P. F. Lambert, C. A. Kulikowski, J. L. Buxton, and R. Walker, "Evaluation and selection of variables in pattern recognition," in *Computer and Information Sciences II*. New York: Academic Press, 1967, pp. 91–122.

The Necessity of Feedback in Minimal Monotone Combinational Circuits

RONALD L. RIVEST

***Abstract*—We present a specific n-input $2n$-output positive unate Boolean function which can be realized with $2n$ two-input gates if feedback is used, but which requires $3n$-2 gates if feedback is not used.**

Manuscript received April 2, 1976. This work was supported by the National Science Foundation under Grant GJ-4364X and Contract DCR74-12997-A01.

The author is with the Laboratory of Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139.

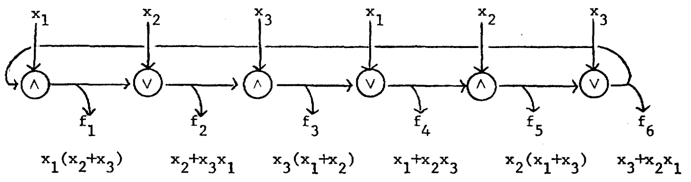

Fig. 1.

***Index Terms*—Boolean functions, feedback, gate complexity, minimal combinational circuits, monotone functions.**

In [2] W. H. Kautz demonstrates that any minimal combinational circuit realizing a specific three-input three-output Boolean function using only NOR gates must contain feedback. (That is, the directed graph corresponding to the minimal circuit is not acyclic.) In [1] David A. Huffman studies circuits containing feedback in detail, concentrating on minimizing the number of inverters used. He proves that any function can be realized with just one inverter, plus AND and OR gates, if feedback is used. At the end of his paper, Huffman raises the question of whether feedback is only useful in producing functions which are not positive unate, or whether feedback can in fact help reduce the number of AND and OR gates required to realize positive unate functions as well. In this correspondence we present a specific positive unate n-input $2n$-output Boolean function which can be realized using just $2n$ two-input gates if feedback is used, but requires at least $3n$-2 gates if feedback is not used. We assume that n is odd for the rest of our discussion.

The optimal circuit for our function for $n = 3$ is given in Fig. 1.

This example is generalized to the following function of n inputs $\{x_1, \cdots, x_n\}$ and $2n$ outputs $\{f_1, \cdots, f_{2n}\}$:

$$f_{2i-1} = x_j \wedge (x_{j-1} \vee (x_{j-2} \wedge (x_{j-3} \vee (\cdots \vee x_{j+1}))\cdots)$$
$$\text{for } 1 \le i \le n \text{ where } j = ((2i-2) \bmod n) + 1;$$
$$f_{2i} = x_j \vee (x_{j-1} \wedge (x_{j-2} \vee (x_{j-3} \wedge (\cdots \wedge x_{j+1}))\cdots)$$
$$\text{for } 1 \le i \le n \text{ where } j = ((2i-1) \bmod n) + 1;$$

where the subscripts of x are computed modulo n, (with x_n used instead of x_0).

The optimal circuit (with feedback) for $\{f_i\}$ requires exactly $2n$ gates; this circuit is merely the generalization of the circuit in Fig. 1. To prove that this circuit is optimal we need only observe that the $2n$ output functions are all different from each other and from the inputs; at least one gate is required to compute each output function.

To prove that any feedback-free network must use at least $3n$-2 gates, it suffices to note that one gate is required to produce each output function, and that in a feedback-free network there must exist at least one gate producing one of the output functions, which does not depend, even indirectly, on the output of any other gate producing an output function. However, since every output function depends on all n inputs, there must be at least n-2 gates (which are not output gates) connecting the inputs to the chosen output gate. Thus, at least $3n$-2 two-input gates are required by any feedback-free combinational realization.

Hence, we have demonstrated that feedback can reduce by a factor of at least $\frac{2}{3}$ the number of two-input gates required to realize positive unate functions, answering Huffman's question, and providing the first proof that feedback can yield reductions in size by a factor $c < 1$, as $n \to \infty$. We have been unable to improve upon the constant $\frac{2}{3}$ by considering nonmonotone functions, nor do we have any reason to suspect that $\frac{2}{3}$ is the best possible improvement factor for monotone functions. Whether feedback is useful in reducing the size of minimal networks for single-output functions is also unknown. It thus remains an open problem to determine the extent to which feedback can yield economical realizations; this determination must await the development of better techniques for proving lower bounds on the gate complexity of feedback-free networks for Boolean functions.

References

[1] D. A. Huffman, "Combinational circuits with feedback," in *Recent Developments in Switching Theory*, A. Mukhopadhyay. New York: Academic Press, 1971.

[2] W. H. Kautz, "The necessity of closed circuit loops in minimal combinational circuits," *IEEE Trans. Comput.*, vol. C-19, pp. 162–166, Feb. 1970.

Comments on "A Computer Algorithm for Transposing Nonsquare Matrices"

DAVID C. VAN VOORHIS

***Abstract*—In the above correspondence[1] a three-step algorithm is presented for transposing large nonsquare matrices stored externally or in main memory. The first step can be generalized, and the last two steps can be replaced with a single step.**

***Index Terms*—Externally stored matrices, large matrices, matrix transposition, nonsquare matrices, partitioned matrices.**

An $M \times N$ array $A[*,*]$ is stored in row-major order as a one-dimensional array $B[*]$, with $B[iN + j] = A[i,j]$ for $i = 0,1,\cdots,M-1$ and $j = 0,1,\cdots,N-1$. The array $B[*]$ may be stored externally or in main memory. In either case the objective is to rearrange the elements of $B[*]$ so that it stores the transpose of $A[*,*]$ in row-major order; i.e., the objective is to achieve the permutation

$$B[jM + i] \leftarrow B[iN + j], \qquad i = 0,1,\cdots,M-1;\ j = 0,1,\cdots,N-1. \quad (1)$$

Furthermore, this permutation should be achieved with a "small" amount of temporary storage.

Alltop[1] describes a three-step transpose algorithm requiring only $2MN/D$ temporary storage locations for an externally stored array and MN/D locations for an array in main memory, where D is a common divisor of M and N. (The original array can be padded with zeros, if necessary, to achieve a larger array whose dimensions have a suitable common divisor.) Although the original description of the algorithm requires D to be a power of 2 when the array is stored externally, we shall see that this restriction can be eliminated if fMN/D temporary storage locations are available, where $f_1 f_2 \cdots f_d$ is any factorization of D and $f = \max_i(f_i)$. Furthermore, we shall see that the last two steps of the three-step algorithm can be replaced with a single step.

The three-step transpose algorithm uses the permutation $\mathcal{T}(R_5,R_4,R_3,R_2,R_1)$ defined as follows for any factorization $R_5R_4R_3R_2R_1$ of MN.

Manuscript received February 2, 1976; revised May 3, 1976.

The author is with the System Communication Division, IBM Corporation, Los Gatos, CA 95030.

[1] W. O. Alltop, *IEEE Trans. Comput.*, vol. C-24, pp. 1038–1040, Oct. 1975.

$$\begin{pmatrix} A_{0,0}[*,*] & A_{0,1}[*,*] & \cdots & A_{0,D-1}[*,*] \\ A_{1,0}[*,*] & A_{1,1}[*,*] & \cdots & A_{1,D-1}[*,*] \\ \vdots & \vdots & & \vdots \\ A_{D-1,0}[*,*] & A_{D-1,1}[*,*] & \cdots & A_{D-1,D-1}[*,*] \end{pmatrix}$$

Fig. 1. The original $M \times N$ array $A[*,*]$ partitioned into D^2 $M' \times N'$ subarrays.

$$\begin{pmatrix} A_{0,0}[*,*] & A_{1,0}[*,*] & \cdots & A_{D-1,0}[*,*] \\ A_{0,1}[*,*] & A_{1,1}[*,*] & \cdots & A_{D-1,1}[*,*] \\ \vdots & \vdots & & \vdots \\ A_{0,D-1}[*,*] & A_{1,D-1}[*,*] & \cdots & A_{D-1,D-1}[*,*] \end{pmatrix}$$

Fig. 2. The $M \times N$ array achieved by Step 1.

$$\begin{pmatrix} A_{0,0}{}^T[*,*] & A_{1,0}{}^T[*,*] & \cdots & A_{D-1,0}{}^T[*,*] \\ A_{0,1}{}^T[*,*] & A_{1,1}{}^T[*,*] & \cdots & A_{D-1,1}{}^T[*,*] \\ \vdots & \vdots & & \vdots \\ A_{0,D-1}{}^T[*,*] & A_{1,D-1}{}^T[*,*] & \cdots & A_{D-1,D-1}{}^T[*,*] \end{pmatrix}$$

Fig. 3. The final $N \times M$ array.

$$\begin{aligned} &B[x_5R_4R_3R_2R_1 + x_2R_4R_3R_1 + x_3R_4R_1 + x_4R_1 + x_1] \\ &\quad \leftarrow B[x_5R_4R_3R_2R_1 + x_4R_3R_2R_1 + x_3R_2R_1 + x_2R_1 + x_1], \\ &x_i = 0,1,\cdots,R_i - 1. \end{aligned}$$

This permutation amounts to treating $B[*]$ as R_5 $R_4 \times R_2$ arrays, whose elements are $R_3 \times R_1$ subarrays, and transposing each $R_4 \times R_2$ array. The three-step algorithm is the following sequence of three such transpose operations, where $M' = M/D$ and $N' = N/D$.

Step 1: $\mathcal{T}(1,D,M',D,N')$.
Step 2: $\mathcal{T}(D,M,1,N',1)$.
Step 3: $\mathcal{T}(N,M',1,D,1)$.

The permutation in Step 1 can be written as

$$B[rMN' + qN + pN' + s] \leftarrow B[pNM' + qN + rN' + s], \qquad (p,q,r,s) \in \mathcal{R}, \quad (2)$$

where $\mathcal{R}$ is the set of 4-tuples defined by

$$\mathcal{R} = \{(p,q,r,s) \mid 0 \leqslant p < D,\ 0 \leqslant q < M',\ 0 \leqslant r < D,\ 0 \leqslant s < N'\}.$$

This permutation treats the original $M \times N$ array $A[*,*]$ as a $D \times D$ array whose elements $A_{p,r}[*,*]$ for $p = 0,1,\cdots,D-1$ and $r = 0,1,\cdots,D-1$ are $M' \times N'$ subarrays, as shown in Fig. 1. The array $B[*]$ initially represents the subarray elements $A_{p,r}[q,s]$ for $(p,q,r,s) \in \mathcal{R}$, with

$$B[pNM' + qN + rN' + s] = A_{p,r}[q,s] = A[pM' + q, rN' + s],$$

and the permutation (2) rearranges the elements of $B[*]$ so that it represents the new $M \times N$ array shown in Fig. 2.

If $B[*]$ is stored in main memory, the transpose (2) can be achieved by simply interchanging the elements of the subarrays $A_{p,r}[*,*]$ and $A_{r,p}[*,*]$, for $p = 0,1,\cdots,D-1$ and $r = 0,1,\cdots,D-1$, which requires only one temporary storage location and one pass over the data. For an externally stored array, Delcaro and Sicuranza's extension [1] of Eklundh's algorithm [2] requires d passes over the data and fMN' temporary storage locations, where $f_1 f_2 \cdots f_d$ is any factorization of D and $f = \max_i(f_i)$. (The original description of the three-step algorithm requires $f_i = 2$ for $i = 1,2,\cdots,d$, so that $D = 2^d$ and $f = 2$.)