XXI. COGNITIVE INFORMATION PROCESSING*

Academic and Research Staff

Prof. M. Eden
Prof. T. S. Huang
Prof. F. F. Lee
Prof. C. Levinthal
Prof. S. J. Mason
Prof. K. Ramalingasarma
Prof. W. F. Schreiber
Prof. D. E. Troxel
Dr. W. L. Black
Dr. K. R. Ingham
Dr. P. A. Kolers
Dr. O. J. Tretiak
C. L. Fontaine
P. H. Grassmann†
E. R. Jensen
J. E. Green

Graduate Students

J. Allen
G. B. Anderson
T. P. Barnwell III
B. A. Blesser
P. C. Economopoulos
A. Gabrielian
C. Grauling
R. E. Greenwood
E. G. Guttmann
R. V. Harris III
D. W. Hartman
H. P. Hartmann
A. B. Hayes
L. J. Hirschfeld
G. R. Kalan
A. N. Kramer
W-H. Lee
C-N. Lu
J. I. Makhoul
G. P. Marston III
O. R. Mitchell, Jr.
S. L. Pendergast
D. R. Pepperberg
G. F. Pfister
R. S. Pindyck
W. R. Pope, Jr.
D. S. Prerau
G. M. Robbins
K. B. Seifert
C. L. Seitz
D. Sheena
D. R. Spencer
W. W. Stallings
P. L. Stamm
R. M. Strong
Alice A. Walenda
G. A. Walpert
J. W. Wilcox
J. W. Woods
I. T. Young

A. MINIMIZATION OF BOOLEAN EXPRESSIONS BY COMPUTER PROGRAM

One of the primary goals of a logic designer is to minimize the amount of hardware which is necessary to realize a digital system. Essentially all digital systems require the realization of combinational nets, and this study[1] is directed toward the minimization of that realization. It is recognized, of course, that the final realization of digital systems must be in terms of NAND or NOR gates. Unfortunately, no mechanical procedures are available for the minimization of such circuits.

Mechanical procedures are available, however, which will result in a minimum sum of products (MSP) or a minimum product of sums (MPS). While it is easy to concoct simple examples when the MSP is not along the most direct route to a minimum solution, e.g., the standard solution for an EXCLUSIVE OR function, it is probable that for a large number of cases the MSP is a reasonable first step in the realization of a minimized NAND or NOR form.

---

*This work was supported principally by the National Institutes of Health (Grants 1 PO1 GM-14940-02 and 1 PO1 GM-15006-01), and in part by the Joint Services Electronics Programs (U.S. Army, U.S. Navy, and U.S. Air Force) under Contract DA 28-043-AMC-02536(E).

†On leave from Siemens A.G., Erlangen, Germany.

The initial purpose of this project was to implement the Quine-McCluskey[2] method for the minimization of a Boolean function on a PDP-9 computer. Unfortunately, this method requires a full expansion of the input expression which would either require an unreasonably large block of memory or severely limit the number of possible variables in the expression to be minimized. Therefore, an alternative method that makes use of the basic principles of switching theory enumerated in Fig. XXI-1 was implemented. Of course, this method also requires a certain amount of core memory for a given input

(A1)	x + x = x	(idempotency)
(A2)	x*x = x	(idempotency)
(A3)	x*x' = 0	(complement)
(A4)	x + x' = 1	(complement)
(A5)	x + xy = x	(absorption)
(A6)	(x+y) + z = x + (y+z) = x + y + z	(associative)
(A7)	(xy)z = x(yz) = xyz	(associative)
(A8)	x(y+z) = xy + xz	(distributive)
(A9)	xy + x'z = xy + x'z + yz	(consensus)
(A10)	f(x,y,...,z,+,*)' = f(x',y',...,z',*,+)	(DeMorgan's theorem)

Fig. XXI-1. Basic principles of switching theory.

expression. But it has the advantage of requiring memory that is somewhat proportional to the complexity of the input expression rather than being simply the number of variables included in the input expression. In other words, the same available core memory can accommodate expressions with many variables but not too many terms as easily as it can expressions with few variables but many terms. While the program that realizes this alternative method was written in the MACRO-9 language for the PDP-9 computer, the algorithm can be applied to any language that is capable of string processing.

1. Method

The flow chart for the algorithm is indicated in Fig. XXI-2. The primary storage for the input expression and those generated in the process of the algorithm are two push-down stores and a small internal buffer area. The input expression consists of a string of variables and operators which is terminated by an exclamation point. The only

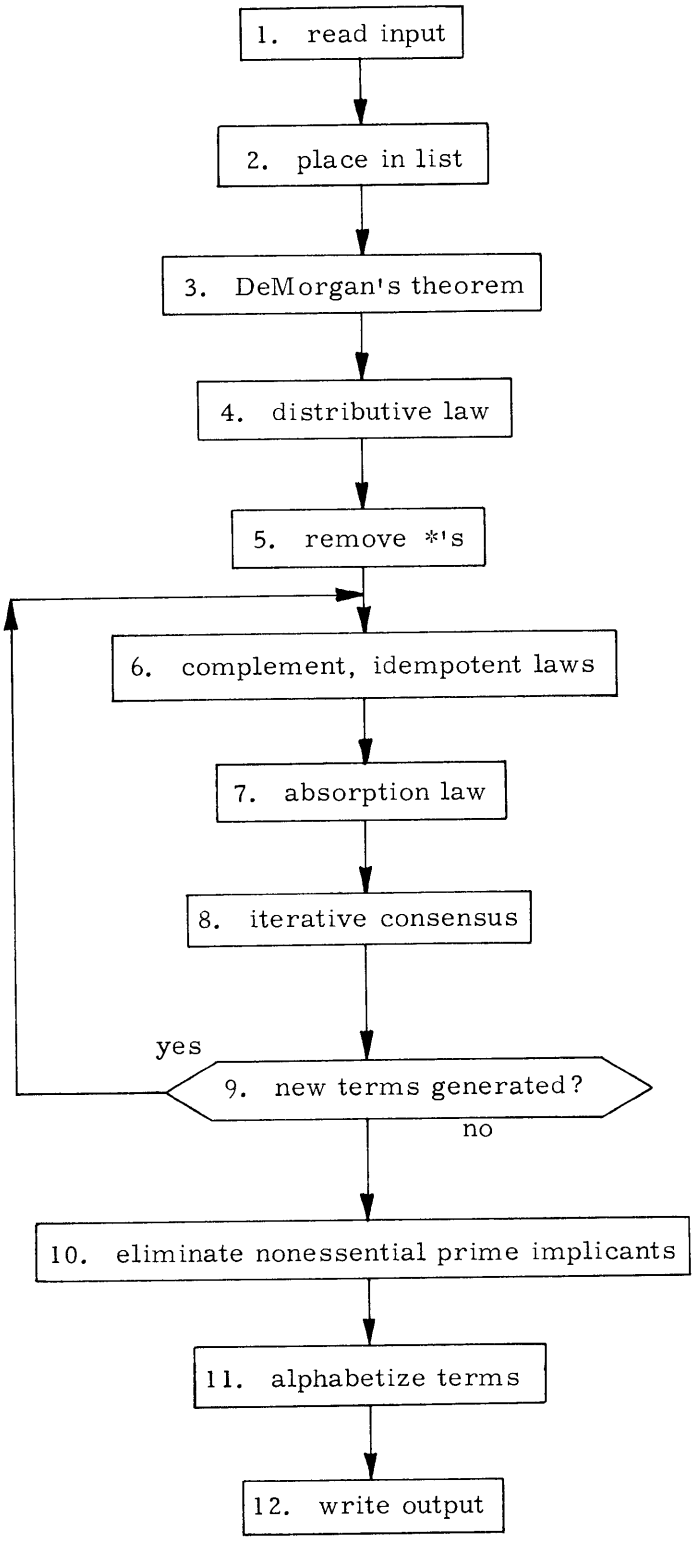

Fig. XXI-2. Algorithm for minimization.

operators that are recognized are addition (+), multiplication (*), left parenthesis (( ), right parenthesis ( )), and prime ('). All other characters recognized by the computer are valid variables. The elements 0 and 1 should, however, be avoided as variables, because of their properties in the 2-valued Boolean algebra. Of course, the input expression should include all such parentheses as are necessary to avoid ambiguities. Also, the multiplication operator must be included for all cases except simple juxtaposition of variables. The first operation is to read the input Boolean expression and load it into the push-down store, one character per list word. The second list is needed not only to facilitate eliminating and searching but also because certain phases of the procedure require end-to-beginning scanning rather than beginning-to-end scanning.

2. DeMorgan's Theorem

As indicated in Fig. XXI-2, the first step after placing the input in the initial list is to apply DeMorgan's theorem. Because of the rule that requires all necessary parentheses to be included in the input, it is sufficient to locate only those primes immediately following a closed parenthesis. The justification for the second list is now clear. The string is processed from end to beginning, with DeMorgan's theorem being followed whenever a prime is followed by a closed parenthesis. Two subroutines are used in the implementation of this part of the algorithm. One examines the string from end to beginning, searching for the pattern described above; the other performs the necessary operations (complement all variables, replace + by *, and replace * by +). Because of the possibility of nested parentheses, it is necessary that each of the two subroutines be able to call itself, as well as the other. Once DeMorgan's theorem has been executed on a single pass through the string, the expression contains primes associated with single variables only. Parentheses are still present, however.

3. Distributive Law

The purpose of this section of the program is to eliminate all parentheses. Equations (A6) and (A7) suggest that there are certain conditions under which this elimination is easy. By checking the symbols immediately preceding and following parenthesized expressions, these trivial cases can be solved. A problem arises when a * is a neighbor of a parenthesis. When considering parentheses, it is necessary to work from the innermost set outward. To accomplish this, the string is scanned from beginning to end in search of the first closed parenthesis. The boundaries of this expression are then inspected. If this parenthesized term is surrounded by +'s, or another set of parentheses, the original parenthesis can be erased. When these cases do not exist, however, a multiplication must be involved. When the appropriate * is found, its surrounding terms can be cross-multiplied. When the cross multiplication is completed,

the scanning resumes at the closed parenthesis of the new term. Eventually the multiplication must cease and the trivial parentheses contained in the final expression are eliminated as mentioned. The function is now in a sum-of-products form. It contains only variables, ' 's associated with the variables, $\not$'s, and *'s. As the *'s are not now necessary to prevent ambiguities, they are removed.

4. Complement and Idempotent Laws

In effect, the procedure described above has been aimed not at minimization but at transforming input information into a sum of products to facilitate subsequent minimization. The basic properties represented by Eqs. (A2) and (A3) are implemented by subjecting the string to another pass in which each term is scanned for the multiple occurrence of any variable. If a variable occurs twice in the same form, only the second occurrence is retained. But if the second occurrence is the complement of the initial variable, the entire term is discarded.

5. Absorption Law

The expression is further minimized by application of the absorption law (Eq. (A5) with Eq. (A1) as a special case). A pairwise comparison is made between all terms in the sum of products. The criterion for elimination is a simple one: If every variable of one term appears in the same form (primed or unprimed) in the other term, the second term is redundant. It is important to note, however, that, contrary to possible first thoughts, the term with the largest sum of variables is the one that is discarded. A comparison is made among all pairs of terms, thereby eliminating redundancy in the process. The special case shown in (A1) is reduced also, since a term appearing more than once is considered to be contained in itself.

6. Iterative Consensus

The consensus operation[3] is the most subtle of the basic principles. The consensus of two product terms is the largest element contained in the sum of the two terms, but not wholly contained in either. Basically, the consensus of two terms exists only when one and only one variable exists in one form in the first term and as the complemented form in the other term. In this case, the consensus term is defined as the product of the two terms after the double-valued variable has been deleted. Once again, the procedure is accomplished by a pairwise comparison technique that involves searching two auxiliary lists each of which contains one term. Nothing is discarded, however. The consensus terms are added to the string, and form a longer string. This does not seem reasonable in a minimization technique, but in order to minimize the function the largest groupings of terms must be represented. The process iterates until no new terms are created. In the meantime, the string is passed through the basic

redundancy process before each iteration. When no new terms have been created and there are no redundancies of the trivial type, the expressions consist of only the largest groupings of the basic terms; that is, the expression is the sum of all of the prime implicants.

7. Elimination of the Nonessential Prime Implicants

This part of the process is indeed the most valuable part of the minimization. In the Quine-McCluskey method, the nonessential prime implicants are easily spotted because the original inputs have been expressed in the fully expanded form. Every basic input term is checked against the prime implicant, and if an input is only included in one prime implicant, that prime implicant is essential. All essential prime implicants can be found by continuing this operation with the remaining inputs and prime implicants. The first step in the alternative method involved finding the set of all consensus terms of the prime implicant. This set is then reduced by the methods used earlier in the program. From the definition of the consensus operation, it can be seen that a consensus term cannot be an essential prime implicant. It must also be realized that any prime implicant that is not equal to or included in a consensus term must be essential. The procedure is now to separate all of these essential prime implicants. Now any other prime implicant that is equal to or included in the consensus of 2 equal terms is definitely redundant and may be discarded. The final step is to take consensus between essential and nonessential prime implicants. Any nonessential prime implicants that are equal to or included in these consensus terms may be eliminated. As this step is carried out and all eliminations are made, the remaining prime implicants form a minimum expression. This expression may not be unique, however. There may be more than one solution to the problem; nevertheless, each one has the same number of terms. If this is the case, the order in which the consensus was taken between the essential and nonessential prime implicants will determine which of the minimum answers is presented.

Now that the minimization is complete, the expression is alphabetized and written out.

L. J. Hirschfeld, D. E. Troxel

References

1. L. J. Hirschfeld, "Boolean Minimization by Computer Program," S.M. Thesis, M.I.T., June 1968.

2. E. J. McCluskey, Introduction to the Theory of Switching Circuits (McGraw-Hill Publishing Company, New York, 1965), pp. 140-157.

3. Ibid., pp. 165-174.

B. BIT-PLANE ENCODING

Some computer analysis has been performed to evaluate the processing of multilevel digitized pictures by bit-plane encoding.[1] The pictures are initially described by an $N \times M$ matrix of intensity values, from nominal black to nominal white. The intensity is quantized to a linear scale of 2^6 values. Each element of the intensity matrix, having 1 of 64 possible values, may be uniquely represented by 6 bits. By using a specific encoding of the 64 levels – for example, a natural binary code – the 6 bits may be ordered in terms of significance. From one $N \times M$ picture of 2^6 possible intensity values per element, six $N \times M$ bit-plane pictures can be made, each having 2 possible intensity values per element.

We have investigated three aspects of bit-plane encoding:

1. The potential advantage of using the unit-distance, reflected Gray code instead of the natural binary code.

2. The performance of several binary compression schemes when applied to the bit-planes.

3. The possibility of filtering the bit-planes to enhance the performance of the compression schemes with negligible degradation to the original picture.

Three pictures were used in this investigation covering a wide range of subject complexity. Each picture was transformed into 6 bit-planes, using first the natural binary code, and then the Gray code. Each of the bit-planes was encoded with the use of Differential Pulse Code Modulation (DPCM), Run Length encoding (RL), and a form of line-to-line Correlation (Corr) based on a technique developed by EG&G.[2] The bit-planes were also processed by two simple filters, each of which processed single binary elements solely on the basis of contiguous element values. One filter used all 8 contiguous elements; the other used only the 4 vertically and horizontally contiguous elements. In either filter, if all pertinent contiguous elements had the same value, the element that was being processed was forced to that value. The filtered bit-planes were also encoded by the 3 compression schemes.

The entropy, H, of the resultant vocabulary usage was computed for each encoding of each bit-plane. Since the average number of bits, $\overline{n}$, necessary for transmitting the vocabulary by a minimum-redundancy Huffman code satisfies

$$H \leq \overline{n} \leq H + 1,$$

the entropy may be used as a lower-bound approximation to $\overline{n}$. If the encoded bit-plane requires S words from the vocabulary, the total number of bits necessary to describe the encoded bit-plane is $\overline{n}S$, which may be approximated by its lower bound, HS. The entropy per bit,

$$h = \frac{HS}{NM},$$

was then used to compare the various results.

The percentage of improvement in the performance of the compression schemes when using Gray code generated bit-planes instead of the natural binary code is shown in Fig. XXI-3. These results suggest the following generalizations: (a) the use of Gray code allows significant improvement; (b) the advantage is more pronounced in the Correlation compression scheme; and (c) as the complexity of the subject matter increases,

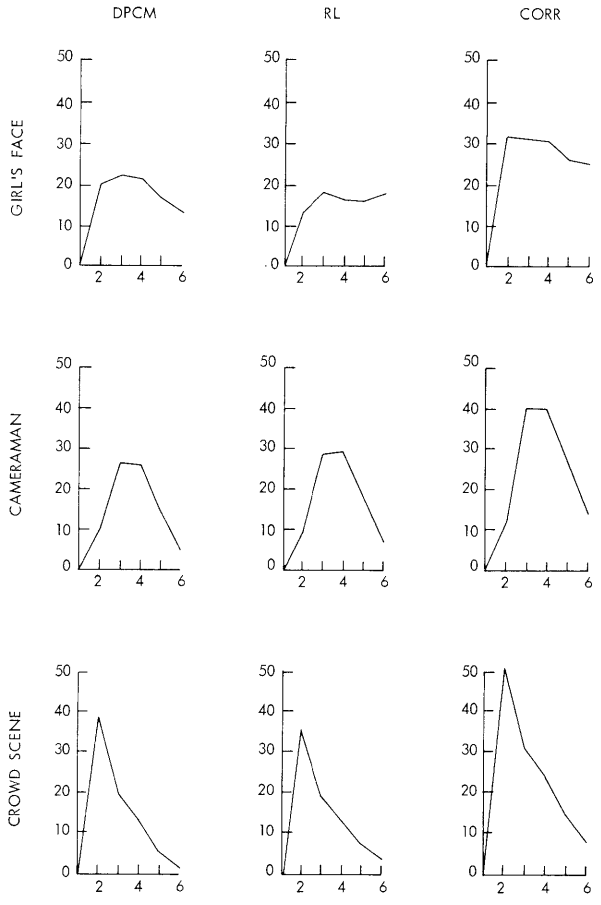

Fig. XXI-3.

Percentage of improvement of Gray code over natural binary code. Six bit-planes, unfiltered.

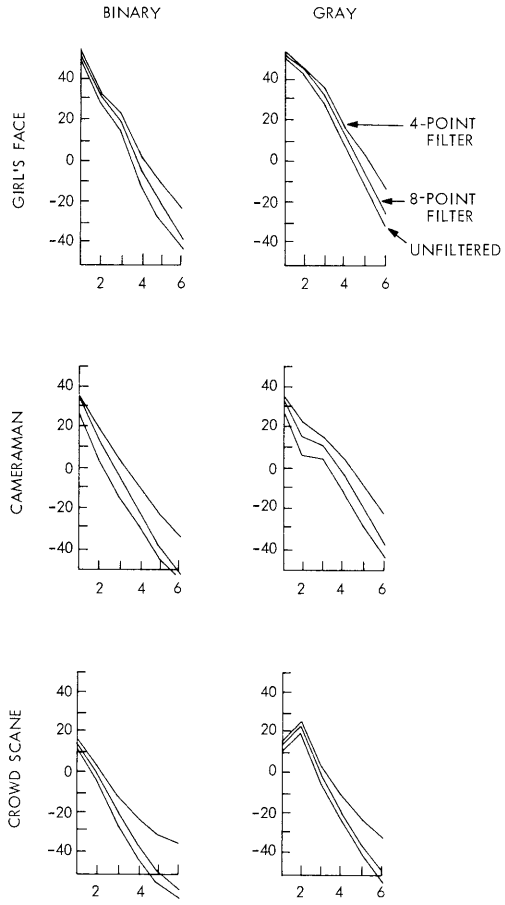

Fig. XXI-4.

Percentage of improvement of Correlation over Run Length encoding.

the Gray code advantage becomes more peaked, increasing for the second bit-plane and decreasing for the bit-planes of lesser significance. Note that for the first bit-plane the improvement is zero, there being no difference between the codes. The advantage of the Gray code should decrease with increasing detail, going statistically to

zero as the bit-plane approaches random noise.

Filtering of the bit-planes introduces relatively little change in these results. The advantage in the second, third, and fourth bit-planes is somewhat decreased, while in the fifth and sixth there is an increase. This result is essentially independent of both compression scheme and picture complexity.

Of particular interest in the comparison of compression encoding schemes is the relative performance of Run Length encoding and line-to-line Correlation. The percentage of improved compression of Corr over RL is sketched in Fig. XXI-4. The Correlation scheme is more advantageous than Run Length encoding for the more significant bit-planes. As the level of detail in the bit-plane increases, a crossover point is reached, after which Run Length encoding is better. Filtering improves the performance of the Correlation scheme more than that of Run Length encoding. The use of Gray code moves the interpolated crossover point almost exactly one bit-plane higher. Increased

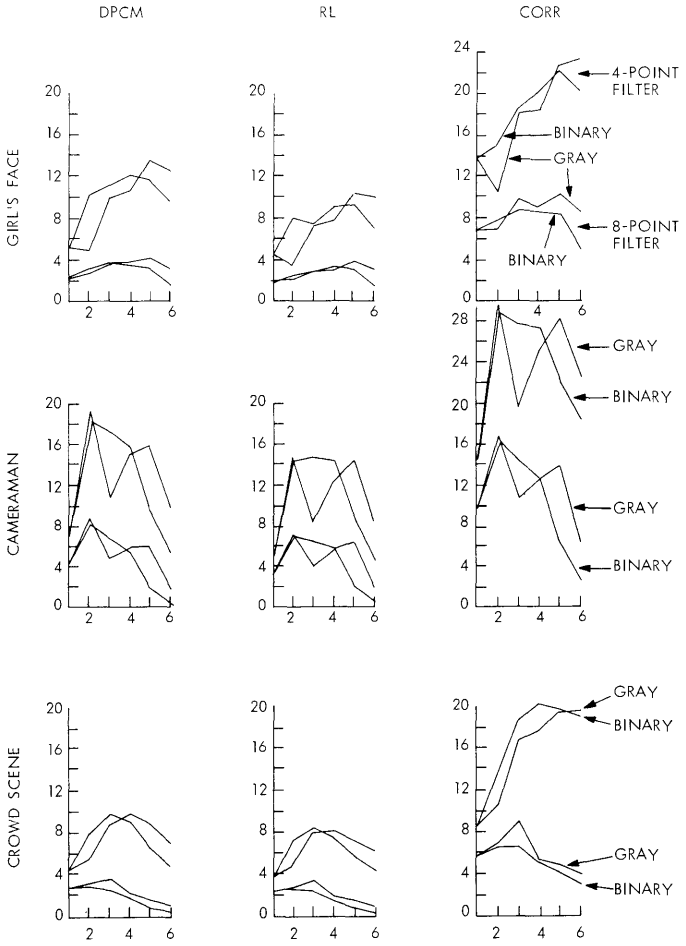

Fig. XXI-5. Percentage of improvement of filtered bit-planes.

picture subject complexity moves the crossover point significantly lower. The highest over-all compression could be achieved by changing from Correlation to Run Length encoding at higher bit-planes.

The percentage of improvement in compression resulting from use of the 8-point and 4-point filters is sketched in Fig. XXI-5. Performance of the filters appears to be relatively independent of bit-plane detail. The 4-point filter is somewhat more than twice as effective as the 8-point filter. The curves for the natural binary code are notably smoother, perhaps because of repetitions of contours in consecutive bit-planes. The line-to-line Correlation scheme profits notably more from the filtering than do the other schemes.

Bit-plane filtering results in degradation of the original picture quality. Typically, single-element spots appear in an area of local intensity gradient. When using natural binary coding, this distortion is somewhat visible in 8-point filtered pictures, and quite noticeable in 4-point filtered pictures. The Gray code filtered pictures appear very close to the original, however, even after 4-point filtering. Numerically, the distortion of the Gray code filtered pictures is approximately 6 dB less than the equivalent natural binary coding.

D. R. Spencer, T. S. Huang

References

1. D. R. Spencer, "Bit-Plane Encoding of Continuous-Tone Pictures," S.M. Thesis, Department of Electrical Engineering, M.I.T., August 1968.
2. Victor Tyler, EG&G, Private communication, Summer 1966.