Encoding of incompletely specified Boolean matrices

T. A. Dolotta
E. J. McCluskey, Jr.

Princeton University
Princeton, New Jersey

Summary — The problem discussed in this paper is a generalization of the problem of choosing the binary codes for the various operations of a digital computer with a view towards minimizing the gating circuitry in the computer's central control. The general procedure is a systematic method for the simultaneous encoding of a large number of arbitrary Boolean functions in an economical manner. The method is applicable to those cases where the designer is allowed to choose the independent variables so as to minimize the circuit which realizes the given Boolean functions. Other applications of the method are indicated. The method given includes a means for determining a lower bound on the cost of the circuit which realizes the given Boolean functions. The method is compared with a previously published procedure for attacking the same problem.

Introduction

In the design of the control part of a digital computer, there may often arise the following problem: We are given the number of bits reserved for the operation code, and also the various control signals that have to be generated for each operation, and we want to choose the codes for the various operations so that the control signals can be obtained from the operation codes by a simple and cheap network.

We will assume that we have m operations to encode with n bits, (of course, m ≤ 2^n), and that we have to generate p control signals. For each operation, we can then specify which control signals must be 0 (i.e., equal to "1"), which must be 1 (i.e., equal to "0"), and which may be either 0 or 1 (i.e., "don't care"). We now formally state the problem.

I. Problem Statement

Assume that we have p columns, F_1 to F_p, each column containing m symbols. There are 3 allowable symbols, namely 0 (zero), 1 (one), and a (don't care). We consider each column as a Boolean function of n independent variables. If any column, F_p, contains only zeros and ones, then we'll say that F_p is completely specified. Otherwise, F_p is incompletely specified. (We do not consider the trivial case where F_p is made up entirely of don't cares; i.e., is completely unspecified.) Our problem is so to choose the n independent variables that the p columns can be synthesized from these variables in a simple and cheap fashion under a set of fairly realistic assumptions. We also want to obtain a lower bound on the cost of synthesizing the p functions.

II. Assumptions and Definitions

1. The number of independent variables, n, has to be specified externally, subject to the restriction that 2^n ≥ m.
2. The n independent variables, denoted by x_1, x_2, ..., x_n, are available together with their complements at no cost.
3. The p Boolean functions F_1, F_2, ..., F_p, are to be synthesized from the independent variables by using 2-stage diode logic only. (Inverters are not allowed.)
4. The cost of synthesizing these functions will be defined as the number of diodes required by the circuit designed under the above assumptions.
5. Given a column, F_i, we define the complement of F_i, denoted by F_i', as the column obtained by changing all the 0's in F_i to 1's and all the 1's to 0's. The a's are not affected.
6. F_i will be said to be a 0(1)-column if it contains no 1's (0's).
7. F_i will be said to be specified in any of its m positions if that position contains a 0 or a 1. Otherwise (if the position contains a a), F_i is said to be unspecified in the particular position.
8. Given two columns, F_i and F_j, there will exist the intersection of F_i and F_j, written as F_i ∩ F_j, if and only if F_i and F_j agree (have the same symbol) wherever both F_i and F_j are specified, F_i · F_j = F_i ∩ F_j. Will be defined as a column of length m which agrees with both F_i and F_j wherever either is specified and contains a a's everywhere else. In figure 1, columns A and B have an intersection, which is shown as column D; columns A and C also intersect. Columns B and C do not intersect. (We write this as B ∩ C = A.)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>a</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>a</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>0</td>
<td>a</td>
<td>a</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 1

9. Three columns, say F_s, F_t, and F_k, will have an intersection, F_s ∩ F_t ∩ F_k, if and only if F_s · F_t · F_k all exist. The
interaction will agree with all three columns wherever any of the three is specified. The extension to the intersection of any number of columns is obvious.

10. Given two columns, \( F_i \) and \( F_j \), \( F_i \) will be said to include \( F_j \) if and only if \( F_j \) agrees with \( F_i \) wherever the latter is specified. We write this relation as \( F_i \supseteq F_j \). In Figure 1, we see that \( A \supseteq B \), and \( B \supseteq F_1 \). If \( F_2 \supseteq F_3 \), we will call \( F_2 \) the including column, and \( F_3 \) the included column. It is obvious that \( F_1 \) has at least as many 0's as \( F_2 \).

11. Below we list some of the more obvious and useful facts which follow from the above definitions:

a. If \( A \supseteq B \), and \( B \supseteq C \), then \( A \supseteq C \).

b. If \( A \supseteq B \), and \( B \subseteq C \), then \( A \subseteq C \).

c. If \( A \supseteq B \), \( B \supseteq C \), and \( C \supseteq A \), then \( A = B = C \).

d. If \( A \supseteq B \), and \( B \supseteq C \), then \( A \supseteq C \).

e. If \( A \neq B \), then \( A \not\supseteq B \), unless \( A \) is completely unspecified.

f. If \( A \supseteq B \), then \( A \not\supseteq B \), whenever \( A \) or \( B \) or both are unspecified in every position.

g. Obviously, if \( A \supseteq B \), then \( A = B \).

h. If \( A \supseteq B \), then \( A \supseteq A \).

i. If \( A \supseteq B \), then \( A \supseteq B \).

j. If \( A \supseteq B \), then \( A \not\supseteq B \), and \( A \neq B \).

k. If \( A \supseteq B \), then \( A \not\supseteq B \), and \( A \neq B \).

12. For any column, \( F_i \), let

\[ N_{i,0} = \text{number of zeros occurring in } F_i \]
\[ N_{i,1} = \text{number of ones occurring in } F_i \]
\[ N_{i,2} = \text{number of } 0 \text{ or } 1 \text{ occurring in } F_i \]

13. Any column, \( F_i \), will be called \text{codable} if and only if

\[ N_{i,0} \leq 2^{n-1} \quad \text{and} \quad N_{i,1} \leq 2^{n-1} \]

Otherwise the column will be called \text{not codable}.

14. We assume that constant signals representing 0 and 1 are available to us.

15. Column \( F_j \) will be said to \text{cover} column \( F_j \) if and only if either \( F_j \supseteq F_j \) or \( F_j \supseteq F_j \).

III. Procedure

The procedure that we will follow in order to obtain an encoding of the various operations is probably best explained by first considering a concrete example. Figure 2 shows a 6 \( \times \) 7 matrix: the six rows correspond to six operations which we must encode, and the seven columns represent seven control signals which must be generated from the operation codes. Each operation must be represented by a three-bit code word. The manner in which the rows and columns are numbered is arbitrary.

We should first examine the matrix row by row to insure that no two rows have an intersection. Should the above not hold, i.e., should there be two rows which have an intersection, then this would mean that the two operations corresponding to the two rows are not distinct, and could be replaced by one operation. In a normal situation, this would not be the case. We note that in Figure 2, there are no two rows which have an intersection.

We now focus our attention on the columns of our matrix. We first note that column 7 is a 1-column. Therefore we need not worry about encoding it, since we can obtain the corresponding control signal by using a constant "1" signal in its place. We therefore discard the 7th column.

At this point we examine the remaining six columns to determine whether there exist any inclusions between any two of them. We find that \( F_6 \supseteq F_1 \). We note that wherever the control signal represented by \( F_6 \) is needed, we may with impunity replace it by the signal corresponding to \( F_1 \), since \( F_6 \) is the same as \( F_1 \) wherever \( F_2 \) is specified. And since \( F_1 \) has to be synthesized, we needn't synthesize \( F_6 \), but merely use \( F_1 \) in its place. Therefore we discard \( F_6 \), and concern ourselves with the remaining five columns, namely \( F_1, F_2, F_3, F_4, \) and \( F_5 \). We call this set of five columns \( \text{Set A} \).

We now examine these five columns to determine whether there exist intersections between any two of these columns. We find that \( F_2 \supseteq F_6 \) exists. Let us define \( F_8 = F_2 \cdot F_6 \). We observe that if instead of encoding \( F_2 \) and \( F_6 \), we encode \( F_8 \), then we can replace \( F_2 \) and \( F_6 \) by \( F_8 \), since the latter agrees with both \( F_2 \) and \( F_6 \) wherever either is specified, so that we will satisfy the original specification. Thus we now have only four columns to encode, namely \( F_1, F_3, F_4, \) and \( F_8 \). Figure 3 shows these four columns.

We should note parenthetically here that in our original matrix (Fig. 2) there also existed the intersections \( F_2 \cdot F_4 \), \( F_2 \cdot F_5 \), \( F_2 \cdot F_7 \), and \( F_2 \cdot F_7 \). All of these, however, involve \( F_5 \) and \( F_6 \) which have been discarded, and therefore these four intersections are of no further interest to us.

At this point we would like to identify each of the four columns in Figure 3 with one of the three independent variables or a complement of such a variable.

This remark may not apply in other applications to which our encoding procedure might be put; and it is conceivable that we might want to have two or more distinct operations even though one might suffice.
variable. Should this be possible, we would then have an ideal (or zero cost) encoding in the sense that each of the seven original signals could be obtained from the independent variables or a constant signal without any intervening logical network. If an ideal encoding is to be possible, then we must be able to identify each one of our gating functions with at least one of the independent variables (or its complement). An ideal encoding very often is not possible for one of two reasons. First of all, in order to be able to identify four columns with 3 independent variables, we must identify at least two of the four columns with one of the independent variables; if this is to be possible, then there must exist at least one inclusion or intersection relation within our four columns. However, we know that this is not the case, and therefore at best only three of the four columns can be identified with the independent variables. Secondly, if a column is to be made into (i.e., identified with) an independent variable, then that column must be codable. This follows from the fact that if we write down the 2^n possible n-bit words, we note that each bit position contains exactly 2^{n-1} "1"'s, and 2^{n-1} "0"'s. Since in our case n = 3, we know that no column which contains more than four "1"'s or "0"'s can be used as an independent variable. We note in Figure 5 that column F_4 is not codable. We conclude on the basis of either or both of the reasons stated above that an ideal solution is not possible in our example, and that at least one 2-diode gate will be needed to synthesize the signal represented by F_4.

We now identify the three remaining columns (namely 1, 3, and 5 calling them Set B) with the three independent variables in any arbitrary fashion. All columns which are members of Set A but not of Set B constitute Set C; in this case Set C contains column 4. Figure 4 shows such an assignment, with 0's replaced by blanks.

<table>
<thead>
<tr>
<th>Oper.</th>
<th>x_1</th>
<th>x_2</th>
<th>x_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 4

It appears at first that we can choose any of the following code words for row 3; 010, 011, 110, 111. However, 010 and 011 are used already by rows 4 and 5, respectively; our choice is limited to 110 and 111. It can be seen that either one of these two choices will give a valid encoding, in the sense that no code word appears more than once. Also, it turns out that either choice will give a simple encoding for F_4. Arbitrarily, we chose 111 as the code word for row 3. Figure 5 shows that final encoding, together with the expressions for the control signals in terms of the independent variables. Of course, we can use the three independent variables in any order we wish, and we may complement any or all of them, so that there are many codes which are symmetries of our code and which give the same type of solution to this problem. 2 This fact is especially useful if we want to avoid the use of a specific code word (such as an all-zero code word), because it is possible to eliminate such a word by complementing one or more of the independent variables whenever n ≈ 2^n. In our example, complementing of x_3 will result in the elimination of the all-zero word. If this is done, then of course x_2 and x_4 have to be substituted for each other in all the expressions where they occur.

We must realize that when we ascertained that columns F_1, F_3, and F_6 were all codable, this was only a necessary, but not a sufficient condition for these columns to make a valid code. In Figure 1, columns C, D, and E are all codable, but they do not make a valid code, since the code 011 occurs twice. (For a code to be valid, it is necessary and sufficient that no code word appear more than once.)

We see that we have obtained here as cheap a 3-bit encoding of our original matrix as is possible, since we have in fact met our minimum estimate of cost. It should be apparent that this is not always the case, and that there may arise other problems when we attempt to encode very large matrices. Consideration of these points as well as a more rigorous and complete statement of our procedure will be the subject of the next sections.

IV. General Considerations

In this section we will discuss some possible situations which have to be considered in order to make our procedure complete.

1. Inclusions. Any time we find that two columns are in the following relationship F_i ≺ F_j we may always replace F_i by F_j and discard F_i. On the other hand, if the relationship is F_i ≻ F_j then F_i may be discarded only if F_j ≺ F_i. (Where F_i is any other column or intersection of columns) is used as an independent variable. Otherwise, we must synthesize F_i; this restriction would not hold if we allowed the use of inverters in addition to 2-stage diode logic.

2. Since there are n! ways of ordering n columns, and 2^n ways in which n columns may be complemented, there are n!·2^n-1 codes which are symmetries of any n-bit code; some of these may be identical with the original code.
2. Intersections. Only those intersections which are codable are of interest to us when we are choosing the columns to be used as independent variables. Non-codable intersections are only useful in the sense that they can be used sometimes to get a cheaper network when the columns which make up the intersections are to be expressed as functions of the independent variables. For instance, if we have two non-codable columns which have an intersection, it may sometimes prove cheaper to synthesize the intersection as one function rather than the two columns as two separate functions. For this reason, non-codable intersections are of interest only if they contain no primed columns. (Again, this last statement would not be true if we allowed the use of inverters.)

It sometimes happens that we find that the following (codable) intersections exist: \( F_1 \cdot F_j \) and \( F_1 \cdot F_k \) (but \( F_j \cdot F_k = 0 \)). In that case, we would use \( F_1 \cdot F_j \) or \( F_1 \cdot F_k \) rather than \( F_1 \cdot F_j \cdot F_k \), since in any case we cover all 3 columns, but \( F_1 \cdot F_j \) has at least one more \( \beta \) than \( F_1 \cdot F_j \). It is in general desirable to have many \( \beta \)'s in the columns used as the independent variables, because this gives us more freedom in "tailoring" the independent variables. It may occur that \( F_1 \cdot F_j = F_1 \cdot F_j \cdot F_k \); this means that \( F_1 \cdot F_j \cdot F_k \); in such an instance we would usually consider the intersections as including \( F_k \), namely as \( F_1 \cdot F_j \cdot F_k \) rather than as just \( F_1 \cdot F_j \).

3. Validity. We have given above the necessary and sufficient condition for a code to be valid. It is quite simple to ascertain whether any column makes a valid code if there are no \( \beta \)'s in these columns. If many \( \beta \)'s do appear, however, it becomes more tedious to determine whether a code can be made valid. However, it is never necessary to revert to enumeration, since the problem of determining whether a code can be made valid is equivalent to the "marriage problem" and can be solved by the algorithms given in references 6, 7, and 8.

4. Cost. Once a valid code covering as many of the columns of our original matrix as possible is found, it is possible to determine (from the number and type of columns which are not covered by this code) the minimum cost of realizing our circuit in terms of either diodes or diode gates. A more detailed discussion of this particular procedure is given in the Appendix.

V. Summary of Procedure

In this section we give a list of the steps which we execute in encoding a matrix.

1. Discard all I-columns and O-columns.
2. Find all inclusion relations of the \( F_i \supseteq F_j \) type which exist among the columns remaining after Step 1. Discard all the including columns found in this manner. Call \( A \) the set of columns remaining now.
3. Make a list of all the inclusions of the type \( F_i \supseteq F_j \) which exist in \( A \) and for which \( F_j \) is codable.
4. If it is desired to eliminate redundant rows from our original matrix, then examine the rows for such redundancies and eliminate them. The procedure here consists essentially of finding all inclusions among rows (or the type \( c_i \supseteq c_j \)), discarding all including rows, and then of finding all the intersections (which contain no primed rows) among the remaining rows and of combining the rows so as to get as small a number of rows as possible. When this is done, we may find that we need fewer bits than \( n \) to encode our matrix, since by eliminating rows we have decreased the value of \( m \). If, however, the smaller number of bits is used in our code, we will often find that the cost of the final circuit will be higher than if we used the original values of \( m \) and \( n \).

5. In Set \( A \), find all pairwise intersections which either contain no primed columns, or are codable (or both).
6. Using the results of Step 5, form all the maximal intersections from Set \( A \) which again are either codable or contain no primes.
7. Using the results of Step 6, form two sets: Set \( B \) will contain all the maximal codable intersections as well as all the codable columns which are members of Set \( A \) but are not covered by any intersections included in Set \( B \). Set \( C \) will contain all the maximal non-codable intersections found in Step 6 as well as all those non-codable columns in \( A \) which are not covered by intersections included in Set \( C \). These two sets cover all the columns in \( A \).

8. From Set \( B \), choose a set of \( n \) members. Call this set \( D \). If \( D \) is so chosen that all the columns of \( A \) which do not appear in \( D \) can be covered with a minimum number of members of Sets \( B \) and \( C \), Set \( D \) is then modified using the results of the second paragraph of Section IV-2 above, so that no column of \( A \) is covered by more than one member of \( D \). Should there be more than one possible choice of members of \( D \), then we choose the members so as to get as many \( \beta \)'s in \( D \) as possible.

9. Check Set \( D \) for validity. If it is not valid, repeat Step 8 and choose the next best set for \( D \). Repeat Steps 8 and 9 until a valid set is found. In executing Steps 8 and 9 we may be able to use some of the relations obtained in Step 3 above to eliminate or simplify some members of Sets \( B \) and \( D \), as indicated in Section IV-1 above.
10. At this point we can compute a minimum bound on the circuit cost (see the Appendix).
11. Once a valid set is found, we may have to replace some of the \( \beta \)'s by 1's or 0's to assure the validity of the set. The remaining \( \beta \)'s are replaced by 1's or 0's so as to make easy
the encoding of the columns which are not identified with the independent variables (but of course still preserving the validity of the code).

12. If the number of possible valid D sets is small, or if this procedure is carried out on a computer, we may want to try several of these sets in order to obtain a better solution. The amount of enumeration here is usually quite small and fairly manageable, especially on a computer.

13. Once a code is obtained, we encode all the members of Set C as well as those members of Set B not included in our final D set. There may be here more than one possible set of columns we may choose for encoding, as indicated in the first paragraph of Section IV-2. The encoding proper is done by standard minimization methods for Boolean functions, and is well described in literature. (See, for instance, references 1 and 2.)

VI. Illustrative Example

The problem which we are treating here was first proposed by E. Hirschhorn (Ref. 3). The procedure proposed by Hirschhorn is somewhat intuitive and depends strongly on the designer's ingenuity. Hirschhorn gives an example in that paper, and we will treat the same example in order to afford us some amount of comparison between the two methods.

The matrix is shown in Figure 6. The value of \( n \) is set at the minimum, namely 5. Hirschhorn finds that if a straightforward code is used (i.e., 00001 is assigned to row 1, 00010 to row 2, 00011 to row 3, etc.), then 45 diodes and 16 gates, or 39 letter symbols (or contacts) are required to synthesize the ten control signals. Hirschhorn's procedure reduces these numbers to 24 diodes and 9 gates, or 25 letter symbols. Both we and Hirschhorn design the circuit for each control signal separately, rather than designing one multiple output network for all the control signals.

<table>
<thead>
<tr>
<th>Op. (C's)</th>
<th>( F_1 )</th>
<th>( F_2 )</th>
<th>( F_3 )</th>
<th>( F_4 )</th>
<th>( F_5 )</th>
<th>( F_6 )</th>
<th>( F_7 )</th>
<th>( F_8 )</th>
<th>( F_9 )</th>
<th>( F_{10} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 6

We will now treat this example by our method.
1. \( F_5 \) is a \( 0 \)-column; it is discarded.
2. There are no inclusions.
3. There are no inclusions.
4. Hirschhorn did not attempt to eliminate any rows. To afford a better comparison, we will do likewise. However, we should point out that the number of rows can be reduced down to seven, by making use of the following intersections which exist in the matrix:

\[
\begin{align*}
&c_7'c_5'c_6'c_7'c_9'c_{11}c_{12}c_{13}c_{16} \\
&c_7'c_{10}'c_{13}'c_{14}'c_{15}' \\
&c_{17}'c_{18}
\end{align*}
\]

These are not the only intersections which occur in the matrix, but they are sufficient to show that the following transformation could be made:

\[
\begin{align*}
&c_a \text{ replaces } c_1 \\
&c_b = c_2 \\
&c_c = c_5c_6c_7c_8c_9c_{11}c_{12}c_{16} \\
&c_d = c_7c_{10}c_{13}c_{14}c_{15} \\
&c_e = c_{17}c_{18} \\
&c_f = c_{19} \\
&c_g = c_{20}
\end{align*}
\]

Therefore \( n \) could be made as small as 3.

5. Set \( A \) consists of all the columns except \( F_5 \).

The following pairwise intersections exist (we omit the letter \( F \), and use the subscripts only):

\[
\begin{align*}
&1 \cdot 9' \\
&3 \cdot 8' \\
&4 \cdot 9' \\
&6 , 8, 6 \cdot 9, 6 \cdot 10' \\
&7 , 8, 7 \cdot 9', 7 \cdot 10 \\
&8 , 9, 8 \cdot 9', 8 \cdot 10 \\
&9 \cdot 10, 9 \cdot 10'
\end{align*}
\]

Columns 1 and 2 are not codable, since \( 2^{n-1} = 16 \) for \( n = 5 \), and \( N_1 \cdot 9' = 18 \) and \( N_2 \cdot 9' = 18 \); therefore we discard the intersection \( 1 \cdot 9' \), since it is not codable and contains a primed variable.

6. The following intersections are maximal and codable (by the rules of section II-11 above):

\[
\begin{align*}
&3 \cdot 8' \\
&4 \cdot 9' \\
&6 , 8, 9' \\
&6 \cdot 9 , 10' \\
&7 , 8 , 9', 10' \\
&7 \cdot 8', 8', 10'
\end{align*}
\]

7. Set \( B \) is identical to the list of intersections immediately above. Set \( C \) contains columns 1 and 2.

8. We note that we can cover all of \( B \) with four columns. In fact, we can choose any of the following
sets:

a. 3·8', 4·9', 6·10', 7·10
b. 3·8', 4·9', 6·9, 7·10
c. 3·8', 4, 6·9·10', 7·9'
d. 3·8', 4, 6·10', 7·9'
e. 3·8', 4·9', 6·7·9'·10'
f. 3·8', 4·9', 6·9·10', 7·8·10'
g. 3·9', 6·8·9·10', 7·9·10'
h. 3·9', 6·10, 7·9·10'
i. 3·9', 6·9, 7·8·10'
j. 3·9', 6·10', 7·9·10'
k. 3·9', 6·9·10', 7·8·10'
l. 3·9', 6·9·10', 7·9·10'
m. 3·9', 6·9·10', 7·8·10'
n. 3·9', 6·10·9·10', 7·8·10'
o. 3·9', 6·7·8·9·10'

Next to each row, we have listed the sum of all the $\Phi$'s in the four columns. Thus we should first choose D from among the first six possibilities. At this point we can further narrow down our choice by considering the columns not covered by D. By examining column 1, we note that if the code words for rows 19 and 20 were to be adjacent (i.e., if they differed in only one bit position), then we could synthesize $F_1$ with only one gate. Similarly column 2 shows us that we want row 1 and row 2 code words to be adjacent. As a result, we would like to avoid the following intersections in our code:

$4·9, 7·9', 7·9·10, 7·8·9', \text{ and } 7·8·9·10$

because the columns resulting from these intersections differ in positions 1 and 2, or 19 and 20. (This is also true of any intersection which covers column 3, but since we want to include this column in our code, we cannot do anything about it.) Thus our first choice narrows down to two sets:

- 3·8', 4·9', 6·10', 7·10
- 3·8', 4·9', 6·9·10'

9. Both of the above sets are valid, so we'll try the first one. Figure 7 shows these four columns and the fifth column ($x_5$) needed to make a 20-word code. Blanks are used instead of $\Phi$'s, and the underlined entries have been made in accordance with the considerations given above to facilitate the encoding of columns 1 and 2. We must make sure that such replacements of $\Phi$'s with 0's and 1's do not make our code invalid. In our case the code is still valid.

10. Since we have found a code which takes care of all but the first two columns, we conclude that the minimum cost for our final circuit will be at least two 2-diode gates, or 4 diodes and 2 gates, or 11 letter symbols. (See Appendix for details of this particular step.)

11. We now fill out the remainder of our code in any arbitrary fashion, with the only restriction being that the code has to be kept valid, i.e., no two rows may be assigned the same code word.

- 3·8', 4·9', 6·10', 7·10
- 3·8', 4·9', 6·9·10'

12. We may, if we wish, try one or more of the remaining fourteen possible D sets, to determine whether a better code is possible. We will not do this, since, as we will see in the next step, our solution is quite good.

13. Since we have used 20 of the 32 possible code words, the remaining 12 words become in effect "don't cares" and may be used to simplify the expressions for $F_1$ and $F_2$. Figure 8 shows the code filled out as per Step 11 above, and the resulting expressions for the 10 gate signals. We see that our solution requires 7 diodes and 2 gates, or 14 letter symbols. This is quite close to the minimum cost bound as determined above, and considerably better than the solution proposed by Hirschhorn, which requires 24 diodes and 9 gates, or 25 letter symbols.

We should note that while performing Step 8, we could have also observed that since with six columns we could have covered all the members of Set A, and that if we were to use six bits, all the columns of Set A would be codable, so that it is conceivable that with $n = 6$ we could have obtained an ideal solution, provided we could have made a valid code. In our case this is not possible for $n = 6$, because if we use columns 1, 2, and 3 (or 3·8') as three of the six code columns (and we must do this to obtain an ideal solution), then we note that rows 3 through 14
Another possible use it has is in the synthesis of Boolean functions with some number of variables. We do come quite close to this bound, and thus get not guaranteed that our solution will be as good and cheap as possible, except in the cases where we meet our lower bound on the cost; in most cases, however, we do come quite close to this bound, and thus get quite a good solution. It is also true that this bound cannot always be met, but it does give us an idea of how good our solution is.

As is true for most procedures of this type, our method is quite systematic, and can be mechanized on a digital computer. We have defined above the cost of a circuit as being the number of diodes needed to construct that circuit in a 2-stage form. We will give here a method for estimating the lower bound on the cost of a circuit in terms of both the diodes and the gates needed to realize the circuit in 2-stage form. The reason for including the cost in terms of gates is that in some realizations (for instance, when resistor-transistor logic is used) the number of gates or packages is of more interest than the number of diodes.

Let us define $P_d$ and $P_g$ as the minimum costs in terms of diodes and gates, respectively, of our circuit. We determine these values in the following fashion:

1. Consider Sets $B$ and $D$, as defined in the text above. Let $b$ be the minimum number of columns needed to cover all those columns in $B$ which are not covered by the largest valid $D$ set, then it is obvious that we will need at least $b$ gates and $2b$ diodes to realize the signals corresponding to all the columns in Set $B$.

2. Consider a column, $F_i$, which is not codable. We can easily count the $N_{i,0}$ and $N_{i,1}$ for this column, and

$$N_{i,b} = 2^{b} - N_{i,0} - N_{i,1}$$

Let $N_i^b$ stand for either $N_{i,0}$ or $N_{i,1}$. If the following relation holds:

$$N_i^b s 2^{b} - N_i^{b} s b, \text{ for } 0 \leq s \leq n$$

then this means that it might be possible to make $N_i^b = 2^{b}$ by replacing the proper number of $b$'s by $1$'s (or $0$'s). If this is possible and is in fact done, then $F_i$ can be realized with a single gate and $n-s$ diodes. Thus we want $s$ as large as possible. As an example, consider $F_i$ in our illustrative example:

In order to make this presentation more complete, we will mention here a few considerations pertinent to the use of our encoding method.

First of all, we should realize that while we discussed the procedure in terms of design of central control of a digital computer, this is by no means the only application to which our method can be put. Another possible use it has is in the synthesis of the output networks of a sequential switching circuit. In fact, we can use this procedure whenever we need to simultaneously encode any number of Boolean functions with some number of variables.

As is true for most procedures of this type, our method will be better suited to some problems than to others. In particular, it lends itself better to encoding matrices which contain many "don't care" than to completely specified ones. Since the method does not try all the possible assignments (there are $1,4 \times 10^{25}$ possible distinct assignments for a 20-word code which uses 5 bits — see ref. 5), we are not guaranteed that our solution will be as good and cheap as possible, except in the cases where we meet our lower bound on the cost; in most cases, however, we do come quite close to this bound, and thus get quite a good solution. It is also true that this bound cannot always be met, but it does give us an idea of how good our solution is.

The main advantage of our procedure is that it is quite systematic, and lends itself well to being programmed on a digital computer. Such a mechanization might in fact greatly enhance the usefulness of our method by allowing us to try several of the possible codes as determined by our procedure (see Sec. IV, Step 12).

Finally, we should point out that this problem has also been treated from a somewhat different point of view by E. J. Schubert (Ref. 4).

VIII. Conclusion

A method has been presented for simultaneously encoding an arbitrary number of Boolean functions in an economical manner. Included is a means for estimating a lower bound on the circuit cost, so that it is possible to obtain some indication of how good a particular encoding is. The entire procedure is quite systematic, and can be mechanized on a digital computer. The application of this procedure to the synthesis of gating signals in the central control of a digital computer is given, and other applications are indicated. The advantages and limitations of the method are discussed.

Appendix

We have defined above the cost of a circuit as being the number of diodes needed to construct that circuit in a 2-stage form. We will give here a method for estimating the lower bound on the cost of a circuit in terms of both the diodes and the gates needed to realize the circuit in 2-stage form. The reason for including the cost in terms of gates is that in some realizations (for instance, when resistor-transistor logic is used) the number of gates or packages is of more interest than the number of diodes.

Let us define $P_d$ and $P_g$ as the minimum costs in terms of diodes and gates, respectively, of our circuit. We determine these values in the following fashion:

1. Consider Sets $B$ and $D$, as defined in the text above. Let $b$ be the minimum number of columns needed to cover all those columns in $B$ which are not covered by the largest valid $D$ set, then it is obvious that we will need at least $b$ gates and $2b$ diodes to realize the signals corresponding to all the columns in Set $B$.

2. Consider a column, $F_i$, which is not codable. We can easily count the $N_{i,0}$ and $N_{i,1}$ for this column, and

$$N_{i,b} = 2^{b} - N_{i,0} - N_{i,1}$$

Let $N_i^b$ stand for either $N_{i,0}$ or $N_{i,1}$. If the following relation holds:

$$N_i^b s 2^{b} - N_i^{b} s b, \text{ for } 0 \leq s \leq n$$

where $s$ is an integer, $0 \leq s \leq n$, then this means that it might be possible to make $N_i^b = 2^{b}$ by replacing the proper number of $b$'s by $1$'s (or $0$'s). If this is possible and is in fact done, then $F_i$ can be realized with a single gate and $n-s$ diodes. Thus we want $s$ as large as possible. As an example, consider $F_i$ in our illustrative example:

<table>
<thead>
<tr>
<th>Op.</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 8

VII. Discussion
Thus s can be 1, 2, or 3. If we choose \( s = 3 \), then conversely we could synthesize \( F_1 \) with one gate and \( n - s = 2 \) diodes. If relation (1) above does not hold, but the following relation holds,

\[
N_{1,x} \leq 2^s + 2^t \leq N_{1,x} + N_{1,y}
\]

where \( t \) is an integer, \( 0 \leq t < s \), then the number of 1's (or 0's) in \( F_1 \) can be made equal to \( 2^s + 2^t \), by substituting the proper number of 1's and 0's for the \( g \)'s in \( F_1 \). If this is done, then the following possibilities exist:

a) If \( s = n - 1 \), then we can synthesize \( F_1 \) with 2 gates and \( n - t + 1 \) diodes. As an example, consider a case where \( N_{1,x} = 17 \), \( n = 5 \). We see that \( s = 4 \) (\( n + 1 \)), and \( t = 0 \). The function \( f = x_1 x_2 x_3 x_4 = x_1 (x_2 x_3 x_4) \) is seen to contain 17 points, and can be realized with 2 gates and \( n - t + 1 = 5 - 0 + 1 = 6 \) diodes.

b) If \( s < n - 1 \), and \( t = s - 1 \), then we can realize \( F_1 \) with 2 gates and \( n - s + 2 \) diodes. For instance, if \( N_{1,x} = 12 \), \( n = 5 \), then \( s = 3 \), \( t = s - 1 = 2 \), and the function \( f = x_1 x_2 x_3 = x_1 (x_2 x_3) \) can be realized with 2 gates and \( n - s + 2 = 5 - 3 + 2 = 4 \) diodes.

c) If \( s < n - 1 \), \( t < s - 1 \), 3 gates are needed and \( 2n - t + s + 1 \) diodes. Let, for example, \( N_{1,x} = 10 \), \( n = 5 \); then \( s = 3 \), \( t = 1 \). The function \( f = x_1 x_2 x_3 x_4 x_5 \) contains 10 points. It can be realized with 3 gates and \( 2n - a - t + 1 = 10 - 3 - 1 + 1 = 7 \) diodes.

The formulas given above are fairly easy to derive. However, it becomes quite difficult to generalize this procedure to the cases where \( F_1 \) doesn't satisfy either relation (1) or relation (2) above. At present, it appears best to set the lower bound in such cases to 3 gates and 6 diodes (since each gate has at least 2 inputs).

If we now consider Set C as defined in the text, we can obtain the minimum bound on the cost of covering all the columns in \( C \) by considering all the possible combinations of columns and/or intersections which cover all the columns in \( C \), getting the bound on each set, and choosing the lowest of these bounds. Let this lowest bound be \( R_d \) and \( R_g \), for diodes and gates, respectively. For instance, if a Set C consisted of \( F_1 \cdot F_2 \), \( F_2 \cdot F_3 \), and \( F_4 \), then we would have to compute the following sums, where \( Q(i) \) stands for the cost of \( F_i \) as determined by Step 2 above:

a) \( Q(1) + Q(2) + Q(3) + Q(4) \)
b) \( Q(1) + Q(2) + Q(4) \)
c) \( Q(1) + Q(3) + Q(4) \)

This is done both for the diodes and gates. We then pick the lowest of these sums as \( R_d \) and \( R_g \) respectively.

4. Our minimum cost bounds are:

\[
P_d = R_d + 2b \\
P_g = R_g + b
\]

We must note here that the minimum cost bounds have been computed on the assumption that the networks which realize the various signals represented by the columns in \( B \) and \( C \) sets are synthesized separately for each such column. This is often the case in many applications, since it makes the design and trouble-shooting easier (a fault in any part of such a network affects only one gating signal). It is, however, possible that in certain cases a saving could be obtained by designing only one multiple output network for all the gating signals. It must of course be realized that the minimum cost bound as computed by the method given above is not necessarily realizable so that in some cases there may exist no solution which meets that bound. However, this bound does give a fair indication of how good any given solution is, and in those cases where it is met, it guarantees that we have indeed an optimum solution.

Acknowledgment

Part of the research presented in this paper was done while both authors were employed by the Bell Telephone Laboratories in Whippany, N. J., and the remainder of the research was supported by the Higgins Funds at the Department of Electrical Engineering of Princeton University.

References


---

4 One should make \( t \) as large as possible.
5 In all three of the above examples it has been assumed for the sake of simplicity that \( N_{1,y} = 0 \).