J. Paul Roth
R. M. Karp

Minimization Over Boolean Graphs*

Abstract: This paper presents a systematic procedure for the design of gate-type combinational switching circuits without directed loops. Each such circuit (Boolean graph) is in correspondence with a sequence of decompositions of the Boolean function which it realizes. A general approach to functional decomposition is given and, in terms of a convenient positional representation, efficient tests for the detection of decompositions are derived. These results are employed in the development of an alphabetic search procedure for determining minimum-cost Boolean graphs which satisfy any given design specifications.

Introduction

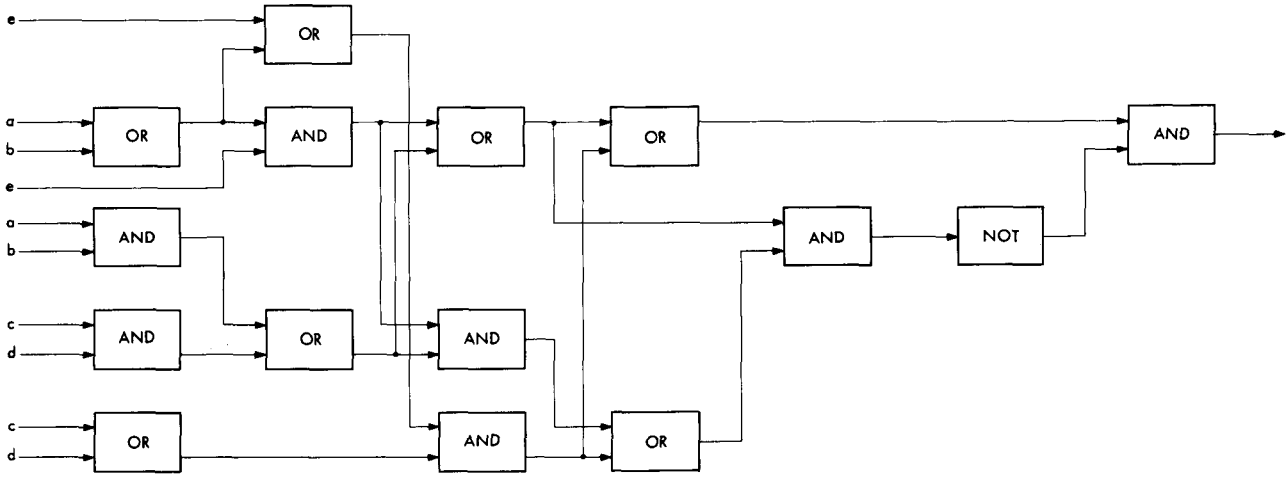
This paper presents a systematic procedure for the design of economical gate-type combinational switching circuits. The procedure requires no restrictive assumptions about the primitive gate elements used and their costs, or about the manner in which they may be interconnected, except that feedback loops are not permitted. Thus the method is a general tool for the design of digital circuits without memory, which may be combined with conventional state-reduction and state-assignment procedures to yield sequential circuit designs as well. An example of a design produced by an IBM 7090 program realizing the procedure is given in Fig. 1.

The algorithm which we present is in contrast with most current synthesis work, which deals almost exclusively with special problems such as the design of two-level AND-OR circuits or of circuits composed of threshold devices. If carried to completion, the algorithm will yield a circuit of strictly minimum cost, but the computation required will usually be excessively long. The algorithm will, therefore, be used principally for the rapid generation of circuits which are economical, but not necessarily minimal.

A previous paper[4] gave a design algorithm for the special case of Boolean trees, in which each primitive

* This is Part V of a series of papers[1-4] devoted to the development of systematic procedures for the design of automata.

Figure 1 **A two-out-of-five validity check circuit.**



element has just one output, and there is a single terminal output. The algorithm of the present paper, while related to that of Ref. 4, is substantially different both in concept and in technique.

Section III, in which the algorithm is presented, is preceded by two sections which provide the requisite definitions and mathematical tools. The first section begins by defining Boolean functions and showing how they can be represented in a convenient positional notation using so-called ON- and OFF-arrays. Boolean graphs are then defined and procedures are given for determining the cost and output functions of a Boolean graph. Finally, the concept of a Boolean graph satisfying a design specification is defined.

The mathematical results and computational techniques underlying the synthesis algorithm are developed in Section II, entitled "Decomposition". This development is, to a great extent, motivated by Ashenhurst's fundamental paper on the decomposition of switching functions,[5] but the approach taken in the present paper is in many respects more abstract and general. A *decomposition* of a function $F(x, y)$ is a representation of the form $z = F(x, y) = G(w, y)$, where $w = \alpha(x)$ and w, x, y, and z are elements of arbitrary finite sets. The function G is called the *image* of the decomposition. Problems concerning the existence of decompositions are formulated and solved in terms of a relation of *compatibility* between specifications of x. Particular attention is given to decompositions of the form $F(x, y) = G(\alpha_1(x), \alpha_2(x), \cdots, \alpha_t(x), y)$, where the indicated functions are Boolean (perhaps incompletely specified), and x and y represent possibly overlapping sets of Boolean variables. Efficient computational procedures, employing operations on the ON- and OFF-arrays defined in Section I, are given for the detection of such decompositions and the computation of their images.

In Section III it is shown that any Boolean graph may be described by a sequence of decompositions, each, after the first, operating on the image of its predecessor. This fact leads to an alphabetic search procedure which is shown to encompass, but not to treat independently, every valid sequence of decompositions, and hence to yield a Boolean graph of minimum cost.

An IBM 7090 program for the synthesis of economical Boolean graphs has been written by Dr. J. R. Wilts of IBM Endicott. The development of this program was the joint effort of F. E. McFarlin, J. R. Wilts, and the authors. The program produced the circuit of Fig. 1 after a few seconds of computation but, in almost an hour of further computation, failed either to yield a less expensive circuit or to establish that the given one is of minimum cost. Reference 6 describes the status of this synthesis program.

I. The synthesis problem

The situation that we wish to treat may be represented as in Fig. 2 by a black box having n input lines on which binary signals $x_1, x_2, \cdots, x_n$ may be impressed, and m output lines carrying binary signals $z_1, z_2, \cdots, z_m$. Our task is to construct the interior of this black box by interconnecting primitive elements of given types so that, at any time, the output signals depend in a specified way on the inputs presently being applied. We proceed to formalize some of the concepts needed for the systematic performance of this task.

• 1.1 *Boolean functions*

Let V^n be the set of all n-tuples (ordered sequences of length n) of 0's and 1's. Such an n-tuple is sometimes called a *vertex*, to connote the geometric picture of a vertex of the unit n-cube. Each vertex corresponds to a configuration of the inputs $x_1, x_2, \cdots, x_n$, and the required dependence of a given output z_i on the inputs is specified by a *Boolean function* F^i having as its domain E, a subset of V^n, and V^1 as its range. If $E = V^n$, F^i is called a *total* Boolean function; otherwise F^i is called a *partial* Boolean function, and the set $V^n - E$ constitutes the DON'T-CARE conditions.

Most practical design problems require many outputs, and the associated Boolean functions usually have DON'T-CARE conditions.

• 1.2 ON- *and* OFF-*arrays*

A Boolean function F of n variables may, of course, be specified by giving a list of all vertices to which F assigns the value 1 and a list of all those to which it assigns the value 0. A more compact notation can be obtained, however, by using n-tuples of symbols 0, 1, and x; these n-tuples are called *cubes*, and represent sets of vertices obtained by changing the x's to 0's and 1's in all possible ways. For example, the cube $0x1x$ represents the four vertices 0010, 0011, 0110, 0111; it is said to *cover* these vertices. If W is a set of vertices, and C, a set of cubes, C is said to be a *cover* of W if (1) each vertex of W is covered by at least one cube of C and (2) each cube of C covers *only* vertices of W.

We shall specify a Boolean function F by means of two covers C_1 and C_0, where C_1 covers the vertices to which F assigns the value 1 and C_0 covers the vertices to which F assigns the value 0. We call C_1 an ON-array of F and C_0, an OFF-array. The DON'T-CARE conditions, if any, are thus not covered by either C_1 or C_0. Our computational procedures do not require that they be specified explicitly.

Figure 3 gives ON- and OFF-arrays for a Boolean function $F(a, b, c, d, e)$. Here each row is a cube, each

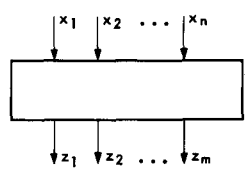


Figure 2 **Black-box interpretation of the synthesis problem.**

column corresponds to an argument. For instance, the first row $A = 10x11$ may be thought of as standing for the vertices 10011 and 10111. In this example there are DON'T-CARE conditions, among them the vertices of $110xx$.

	a	b	c	d	e	
A	1	0	x	1	1	
B	0	1	1	x	x	
C	x	0	1	1	x	C_1
D	0	0	0	0	0	
	-	-	-	-	-	
E	0	1	0	1	0	
F	0	x	0	x	1	C_0
G	1	1	1	1	1	

Figure 3 **Example of a function defined by ON- and OFF- arrays C_1 and C_0.**

In the terminology of logic, a cube corresponds to a term, and a cover, to a normal-form expression. For instance, the first row $A = 10x11$ stands for the term $a\bar{b}de$. The ON-array C_1 corresponds to a normal form expression consisting of the disjunction of four terms, one for each cube:

$$a\bar{b}de \vee \bar{a}bc \vee \bar{b}cd \vee \bar{a}\bar{b}\bar{c}\bar{d}\bar{e} .$$

Likewise C_0 defines a normal-form expression for a function prescribing precisely the conditions for which F is 0.

The basic operations employed in the computational procedures of the following sections are performed on ON- and OFF-arrays. Two relations between cubes are particularly important in these procedures. Cubes A and B are said to *intersect* if they cover a common vertex. Cube A is said to *subsume* cube B if all the vertices covered by A are covered by B. If A and B are cubes in a cover, and A subsumes B, then the function described by the cover is not changed by deleting A.

Since the length of the calculations grows rapidly with the numbers of cubes in the ON- and OFF-arrays, it is best to go through a preliminary calculation which makes them as concise as possible. Procedures for performing such simplifications are given in Refs. 1 and 2.

1.3 *Boolean graphs*

Boolean functions may be used to specify the required external, or black-box, behavior of a circuit. We next describe the internal structure of the circuits considered. It is assumed that a definite set $\mathfrak{B}$ of primitive elements is available for use in circuit construction. Each primitive element may itself be viewed as a black box, with specified numbers of inputs and outputs, and with a set of Boolean functions giving the input-output dependence. Most primitive elements used in practice realize a single function (AND, OR, NOR, NAND, MAJORITY, et cetera) which, however, may be "fanned out" to several output lines, but we allow for the possibility of an element which realizes several distinct functions. A positive integral *cost* is associated with each element of $\mathfrak{B}$.

It is convenient to exclude the NOT element from $\mathfrak{B}$, and instead to introduce into $\mathfrak{B}$ all variants of the primitive elements obtained by putting NOT elements on some of their input and output lines. Sometimes it is desirable to consider $\mathfrak{B}$ as partitioned into sets of elements having the same distinct output functions, but differing in their fan-out properties and costs. Each such set must then be accompanied by *fan-out restrictions* specifying the permissible extent of fan-out, and a specification of the dependence of cost on fan-out.

The circuits which we construct from elements of $\mathfrak{B}$ are in correspondence with Boolean graphs, defined as follows:

Definition 1. A Boolean graph is an acyclic directed graph with its nodes and branches labeled as follows:

a) Any node v with s branches directed into it, and t branches directed out, is labeled with the name of an s-input, t-output element $\alpha \in \mathfrak{B}$.

b) The names of the output functions of α are associated in one-to-one fashion with the branches directed out of v.

c) The branches not labeled in (b) (i.e., the input branches) are labeled with the names of primary input variables.

A *Boolean tree* is a Boolean graph composed of primitive elements having exactly one output line.

It follows from Definition 1 that the signals on the output lines of a Boolean graph are Boolean functions of the input signals. Thus, for the graph of Fig. 4 the output functions are

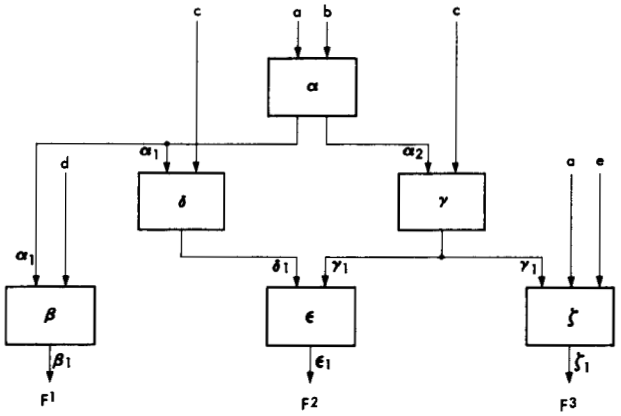
$$F^1 = \beta_1(\alpha_1(a, b), d)$$

$$F^2 = \varepsilon_1(\delta_1(\alpha_1(a, b), c), \gamma_1(\alpha_2(a, b), c))$$

$$F^3 = \zeta_1(\gamma_1(\alpha_2(a, b), c), a, e) .$$

It is possible to give a systematic procedure (specifically

Figure 4 **A Boolean graph with several outputs.**



the Π^*-operation of Refs. 4 and 7) for computing explicitly ON- and OFF-arrays for the output functions of a Boolean graph.[7]

• 1.4 *The minimization problem*

We have now shown, on the one hand, how a problem may be specified by Boolean functions, and on the other hand, how the output functions of a Boolean graph are determined. In order to relate these two types of information, some further definitions are needed. If F is a Boolean function with domain E, let $F^{-1}(1)$ and $F^{-1}(0)$ denote the subsets of E for which F is required to be, respectively, 1 and 0. Let G be another Boolean function, with $G^{-1}(1)$ and $G^{-1}(0)$ similarly defined. Then G is said to be an *extension* of F if and only if $G^{-1}(1) \supseteq F^{-1}(1)$ and $G^{-1}(0) \supseteq F^{-1}(0)$. In other words, G is an extension of F if it agrees with F wherever F is defined. Suppose we are given a Boolean graph with output functions $z_i(x_1, x_2, \cdots, x_n)$, $i = 1, 2, \cdots, m$, and a design problem specified by the functions $w_i(x_1, x_2, \cdots, x_n)$, $i = 1, 2, \cdots, m$. Then the Boolean graph is said to *satisfy* the design specification if, for each i, z_i is an extension of w_i.

Suppose further that we associate with any Boolean graph a *cost*, such as the sum of the costs of the primitive elements composing it. Then we may set the following problem:

Problem: Devise an algorithm which efficiently computes, for any design specification and given set $\mathfrak{B}$, a Boolean graph which satisfies the specification and has a minimum cost.

The present paper gives a solution to this problem. The key to the solution lies in the theory of decomposition, having its roots in the pioneering work of Ashenhurst and in the study of projection operators carried out in Refs. 3 and 4.

The following section, which is self-contained, provides a new formulation of functional decomposition. The algorithms use covers of ON- and OFF-arrays as a base for computation. They constitute a substantial improvement over previous procedures.

II. Decomposition

The ways in which a Boolean graph satisfying given design specifications may be constructed from primitive elements are governed by certain decomposition properties of the Boolean functions to be realized. The present section gives a general discussion of these properties.

• 2.1 *Abstract decomposition*

Let X, Y, Z, and W be arbitrary finite sets, and let E be a subset of the Cartesian product $X \times Y$. Then given a function F mapping E into Z, denoted $E \xrightarrow{F} Z$, we may ask the following questions:

i) Given α, $X \xrightarrow{\alpha} W$, does there exist a function G, $W \times Y \xrightarrow{G} Z$, such that,

for all $(x, y) \in E$, $F(x, y) = G(\alpha(x), y)$? (1)

ii) Under what conditions do there exist functions α, $X \xrightarrow{\alpha} W$, and G, $W \times Y \xrightarrow{G} Z$, such that (1) holds? The representation (1) is called a *decomposition* of F; G is called the *image* of the decomposition.

The answer to question (i) may be formulated in terms of a relation of *compatibility* between elements of X. We say that $x_1 \in X$ and $x_2 \in X$ are *compatible* with respect to F (denoted $x_1 \sim x_2$) if, for all $y \in Y$ such that $(x_1, y) \in E$ and $(x_2, y) \in E$, $F(x_1, y) = F(x_2, y)$; otherwise, x_1 is *incompatible* with x_2, denoted $x_1 \nsim x_2$.

Proposition 1. Given F and α, defined above, there exists G such that (1) holds if and only if, for all $x_1 \in X$ and $x_2 \in X$,

$$\alpha(x_1) = \alpha(x_2) \Rightarrow x_1 \sim x_2 , \qquad 2(a)$$

or, equivalently,

$$x_1 \nsim x_2 \Rightarrow \alpha(x_1) \neq \alpha(x_2) . \qquad 2(b)$$

Proof. Let $\tilde{x} \in W$ be in the range of α, and let $\alpha^{-1}(\tilde{x}) = \{x_i, i = 1, 2, \cdots, q\}$ be the elements of X mapping into $\tilde{x}$. Then, if (2a) holds, for each $y \in Y$ there exists a unique element $\zeta \in Z$ such that, if $x_i \in \alpha^{-1}(\tilde{x})$ and $(x_i, y) \in E$, $F(x_i, y) = \zeta$ (unless, for all $x_i \in \alpha^{-1}(\tilde{x})$, $(x_i, y) \notin E$). Thus we may define $G(\tilde{x}, y) = \zeta$; the function G may similarly be defined consistently for each element of $W \times Y$. Conversely, if (2a) does not hold, there exist $(x_1, y) \in E$, $(x_2, y) \in E$ and $\tilde{x} \in W$ such that $F(x_1, y) \neq F(x_1, y)$ and $\alpha(x_1) = \alpha(x_2) = \tilde{x}$. Then $G(\tilde{x}, y)$, however defined, cannot agree with both $F(x_1, y)$ and $F(x_2, y)$.

The two situations considered in this proof are depicted in Figure 5.

We may also give a simple answer to the second question posed above, in which only F is given, and both α and G are at our disposal. The crucial consideration here is the *number* of elements in W, for if W has too few elements it will not be possible to produce a function such that all elements of X mapping into the same element of W are compatible. These considerations are made precise in Proposition 2, which follows easily from Proposition 1.

Proposition 2. If k is the least integer such that X may be partitioned into k classes of mutually compatible elements, then there exist α and G such that (1) holds if and only if W has at least k elements.

In order to apply Proposition 2, we must supply one missing link—the determination of the number k.

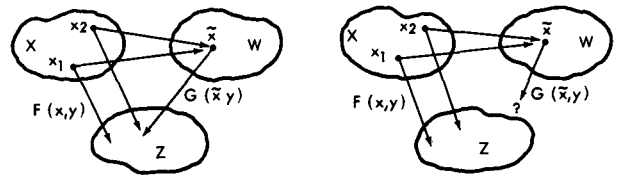


Figure 5 **Situations encountered in the proof of Proposition 1.**

If F is total, this is particularly easy. For in this case, $x_1 \sim x_2$ if and only if, for all $y \in Y$, $F(x_1, y) = F(x_2, y)$. Compatibility is thus an equivalence relation, and k is simply the number of maximal equivalence classes. But if F is undefined for some elements of $X \times Y$, the following situation is possible:
$x_1 \sim x_2$, $x_2 \sim x_3$, $x_1 \nsim x_3$. Compatibility is no longer an equivalence relation, and the determination of a minimum "cover" of X by sets of mutually compatible elements is nontrivial. We shall return to this problem in Section 2.2, where specific computational routines are presented.

• 2.2 *Decomposition tests for Boolean functions*

At this point we depart from our general discussion of decomposition, and make some specializations appropriate to the synthesis of Boolean graphs. We consider decompositions of the form

$$F(x_1, x_2, \cdots, x_n) = G(\alpha_1(v), \alpha_2(v), \cdots, \alpha_t(v), w), \quad (3)$$

where

1) Each of the indicated functions is Boolean.

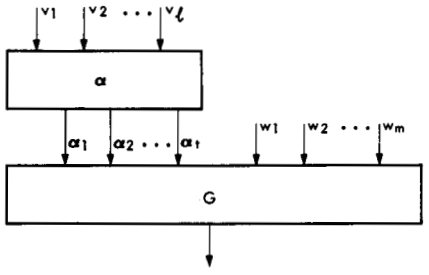
2) v and w, respectively, are specifications of subsets $\lambda = \{v_1, v_2, \cdots, v_l\}$ and $\mu = \{w_1, w_2, \cdots, w_m\}$ of input variables, together exhausting $x_1, x_2, \cdots, x_n$. The decomposition is called *disjunctive* if $\lambda \cap \mu$ is empty (i.e., $n = l + m$), and otherwise, *nondisjunctive* ($n < l + m$).

In the disjunctive case, (3) is a special case of (1), with the sets and mappings specialized as follows: $X = V^l$, $Y = V^m$, $W = V^t$, $Z = V^1$, $V^n \xrightarrow{F} V^1$, $V^{t+m} \xrightarrow{G} V^1$, $V^l \xrightarrow{\alpha} V^t$, where α is represented by the t-tuple $(\alpha_1, \alpha_2, \cdots, \alpha_t)$ of Boolean functions. In the nondisjunctive case (3) is not, strictly speaking, a decomposition. For if we take $X = V^l$ and $Y = V^m$, then the domain of F is not $X \times Y$. This difficulty is easily surmounted. We first define a mapping S, $V^n \xrightarrow{S} V^{l+m}$, in which $(x_1, x_2, \cdots, x_n)$ maps into $(v_1, v_2, \cdots, v_l, w_1, w_2, \cdots, w_m)$; thus, if $\lambda = (x_1, x_2, x_3)$ and $\mu = (x_2, x_3, x_4)$, $S(1, 1, 0, 1) = (1, 1, 0, 1, 0, 1)$. Then we may define the function $\hat{F}(v_1, v_2, \cdots, v_l, w_1, w_2, \cdots, w_m)$ as follows: if $(v_1, v_2, \cdots, v_l, w_1, w_2, \cdots, w_m) = S(x_1, x_2, \cdots, x_n)$, then $\hat{F}(v_1, v_2, \cdots, v_l, w_1, w_2, \cdots, w_m) = F(x_1, x_2, \cdots, x_n)$. Note that $\hat{F}$ is necessarily a partial function, since V^{l+m} has more elements than V^n, and S is one-to-one.

For example, again assuming that $\lambda = (x_1, x_2, x_3)$ and $\mu = (x_2, x_3, x_4)$, $\hat{F}(1, 1, 0, 0, 1, 1)$ is undefined, since it could not have been derived from any specification of (x_1, x_2, x_3, x_4). Such combinations are called *inconsistent*. Instead of (3) we can now write $\hat{F}(v, w) = G(\alpha_1(v), \alpha_2(v), \cdots, \alpha_t(v), w)$, which is, in the strict sense, a decomposition. But we shall refer to (3) also as a decomposition, and shall show that it is possible to test directly for such decompositions without explicitly forming the artificial function $\hat{F}$.

The class of decompositions defined by (3) is more general than those treated by previous writers. Ashenhurst, in Ref. 5, has made a thorough study of disjunctive decompositions such that F is total and $t = 1$. In unpublished papers Ashenhurst[8] and Curtis[9] also considered the nondisjunctive case, but retained their other assumptions.*

A decomposition of F is directly in correspondence with a circuit realization of the type shown below, where the blocks G and α represent Boolean graphs realizing, respectively, the functions G and $\alpha_1, \alpha_2, \cdots, \alpha_t$.



In the synthesis algorithm of Section III we consider only decompositions such that the block α can be realized by a single element of $\mathfrak{B}$, but it is also useful to consider procedures in which this requirement is not imposed.

Let us suppose that a given function F is represented by covers $C_1 = \{a^i, i = 1, 2, \cdots, n\}$ and $C_0 = \{b^j, j = 1, 2, \cdots, m\}$, where the a^i and b^j are cubes, and that λ and μ are specified. Then if Propositions 1 and 2 are to be employed for the detection of decompositions, it is necessary to determine which specifications of λ are compatible. In the disjunctive case ($\lambda \cap \mu$ empty), any cube a of C_1 or C_0 can be divided into a "λ-part a_λ" and a "μ-part" a_μ, which are, respectively, cubes of V^l and V^m, by collecting coordinates associated with elements of λ and μ, respectively. Thus, for the cube

$$\begin{matrix} a & b & c & d \\ 1 & 0 & 1 & x, \end{matrix}$$

with $\lambda = \{a, c\}$ and $\mu = \{b, d\}$, the λ-part is 11, and the μ-part is $0x$, The covers C_1 and C_0 may therefore be written as follows:
$C_1 = \{(a^i_\lambda, a^i_\mu) | i = 1, 2, \cdots, n\}$, and
$C_0 = \{(b^j_\lambda, b^j_\mu) | j = 1, 2, \cdots, m\}$,
and in terms of these covers we may state the following lemma, which follows directly from the definition of compatibility.

Lemma 1. Given $v_0 \in V^l$ and $v_1 \in V^l$, $v_0 \nsim v_1$ if and only if there are cubes $(a^i_\lambda, a^i_\mu) \in C_1$ and $(b^j_\lambda, b^j_\mu) \in C_0$ such that

i) a^i_μ intersects b^j_μ

ii) either a^i_λ covers v_0 and b^j_λ covers v_1, or a^i_λ covers v_1 and b^j_λ covers v_0.

Example 2.1 Suppose $F(a, b, c, d)$ is given by the covers C_1 and C_0 shown below, and that $\lambda = \{a, b\}$ and $\mu = \{c, d\}$.

Note added in proof: In a recent paper, Curtis[12] discussed disjunctive decompositions of total functions, without requiring that $t = 1$.

	C_1					C_0				
	a	b	c	d			a	b	c	d
a^1	1	0	1	x		b^1	1	1	x	1
a^2	1	x	1	0		b^2	1	x	0	x
a^3	0	1	x	x		b^3	x	0	0	x

Then, inspecting in turn each pair a^i, b^j, we find:

${a^1}_\mu$ intersects ${b^1}_\mu$ $\quad 10 \sim 11$

${a^3}_\mu$ intersects ${b^1}_\mu$ $\quad 01 \sim 11$

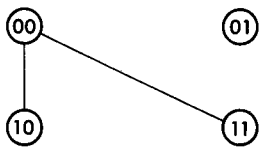
${a^3}_\mu$ intersects ${b^2}_\mu$ $\quad 01 \sim 1x$ (i.e., $01 \sim$ any vertex covered by the cube $1x$).

${a^3}_\mu$ intersects ${b^3}_\mu$ $\quad 01 \sim x0$

By the lemma, the above computation exhibits all pairs of incompatible vertices. Propositions 1 and 2 may now be employed to answer various questions concerning the existence of decompositions. For example:

1) Can F be written in the form $F(a, b, c, d) = G(\alpha(a, b), c, d)$, where $\alpha(a, b) = a \cdot b$? No, because $\alpha(0, 1) = \alpha(1, 0) = 0$, but $01 \sim 10$, contradicting the condition of Proposition 1.

2) Does there exist any Boolean function α such that $F(a, b, c, d) = G(\alpha(a, b,), c, d)$? Since α is required to be a Boolean function, its range is V^1, which has two members. Thus, according to Proposition 2, a decomposition of the type sought exists if and only if V^2 can be partitioned into two sets of mutually compatible elements. The following diagram shows the elements of V^2, with lines connecting pairs of compatible elements. It is clear that a suitable partition cannot be found. We note in passing that $10 \sim 00$, $00 \sim 11$, but $10 \not\sim 11$; this is an example of the nontransitivity of the relation $\sim$, which is possible because F is a partial function.



Lemma 1, as stated, applies only to the disjunctive case. In order to apply it to the nondisjunctive case, one can form $\hat{F}(v, w)$ from F, and then test for disjunctive decompositions of $\hat{F}$. This suggests the following problem: given covers C_1 and C_0 defining F, find covers $\hat{C}_1$ and $\hat{C}_0$ defining $\hat{F}$. An obvious procedure is to replace each cube of C_1 or C_0 by a cube in which the coordinates common to λ and μ are repeated. Thus, if $\lambda = \{a, b, c\}$ and $\mu = \{c, d\}$, the cube $\begin{matrix} a & b & c & d \\ x & 0 & 1 & x \end{matrix}$ would be replaced by the cube $\begin{matrix} a & b & c & c & d \\ x & 0 & 1 & 1 & x \end{matrix}$. This procedure is nearly correct, but there is a pitfall. If we apply the procedure to the cube $\begin{matrix} a & b & c & d \\ 1 & 0 & x & x \end{matrix}$, we obtain the cube $\begin{matrix} a & b & c & c & d \\ 1 & 0 & x & x & x \end{matrix}$, which covers not only the consistent cubes $1\,0\,0\,0\,x$ and $1\,0\,1\,1\,x$, but also the inconsistent ones $1\,0\,0\,1\,x$ and $1\,0\,1\,0\,x$, which should not appear in $\hat{C}_1$ (or $\hat{C}_0$). Thus, the situation is now complicated by the presence of x's in coordinates common to λ and μ. If a cube in C_1 has s such x's, it must be replaced by 2^s cubes in C_1, corresponding to the 2^s consistent assignments of the values 0 and 1 to these coordinates. In practice $\hat{C}_1$ and $\hat{C}_0$ are not actually formed, but certain columns in C_1 and C_0 are thought of as appearing both in the λ-part and in the μ-part.

Example 2.2 We consider the function F defined in Example 2.1, but assume that $\lambda = \{a, b\}$, and $\mu = \{b, c, d\}$. Thus ${a^1}_\mu$ does not intersect ${b^1}_\mu$, due to the presence of the b-coordinate in the μ-part. The following incompatibilities are found:

$i = 3, j = 1 \quad 01 \sim 11$

$i = 3, j = 2 \quad 01 \sim 11 \quad$ (N.B.: $01 \sim 10$).

Since $00 \sim 01$, $00 \sim 10$, and $01 \sim 10$, F can be written in the form $F(a, b, c, d) = G(a \cdot b, b, c, d)$.

There is an alternate way to reduce nondisjunctive decompositions to disjunctive ones. One can simply treat the variables in $\lambda \cap \mu$ as if they were Boolean functions of one variable produced by the primitive element associated with the decomposition. Thus the decomposition of Example 2.2 may be interpreted in two ways, as follows:

(a) $\lambda = \{a, b\}$,
$\mu = \{b, c, d\}$, $\alpha: V^2 \xrightarrow{\alpha} V^1$, $\alpha(a, b) = a \cdot b$

(b) $\lambda = \{a, b\}$,
$\mu = \{c, d\}$, $\alpha: V^2 \to V^2$, $\alpha_1(a, b) = a \cdot b$, $\alpha_2(a, b) = b$;

in case (b), Proposition 1 can be applied directly, without the construction (explicit or implicit) of an artificial function $\hat{F}$. The compatibility conditions of Example 2.1 must be considered:

$10 \sim 11 \quad \alpha_1(1, 0) \neq \alpha_1(1, 1)$

$01 \sim 11 \quad \alpha_1(0, 1) \neq \alpha_1(1, 1)$

$01 \sim 1x \quad \alpha_2(0, 1) \neq \alpha_2(1, 0); \alpha_1(0, 1) \neq \alpha_1(1, 1)$

$01 \sim x0 \quad \alpha_2(0, 1) \neq \alpha_2(0, 0); \alpha_2(0, 1) \neq \alpha_2(1, 0)$.

The inequalities show that any two incompatible vertices are assigned different values of $\alpha = (\alpha_1, \alpha_2)$, and we have a second verification of the validity of the decomposition.

In general, Lemma 1 can be applied in two ways:

a) In conjunction with Proposition 1, it can be used to determine, given F, λ, μ, and $\{\alpha_1, \alpha_2, \cdots, \alpha_t\}$, whether there is a decomposition of the form $F = G(\alpha_1(v), \alpha_2(v), \cdots \alpha_t(v), w)$. This is done simply by determining which incompatibilities exist, and ascertaining whether any of them violate the conditions of Proposition 1.

b) In conjunction with Proposition 2, it can be used to determine, given F, λ, μ, and t, whether there exist functions $\alpha_1, \alpha_2, \cdots, \alpha_t$ such that (1) is satisfied. In the language of Proposition 2, W, the range of $\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_t)$, has at most 2^t elements, and a decomposition exists if and only if $k \leqq 2^t$, where k is the minimum number of mutually compatible elements into which V^l, the domain of α, can be partitioned. Moreover, if $k > 2^t$, all l-input, t-output elements of $\mathfrak{B}$ are eliminated without the necessity of testing each separately. The information conveyed by the value of k is therefore considerable, and we next consider how k can be calculated.

• 2.3 *The determination of k*

The first step of the calculation is to compute all subsets $S \subseteq V^l$ having the properties that

1) Any two elements of S are compatible.
2) If $S \subset T \subseteq V^l$, T does not satisfy (1).

The procedure used to compute these *maximal compatible sets* is analogous to one employed by R. E. Miller[10] in an algorithm for the reduction of states in sequential machines. The sets will be represented by sequences of ZEROS and ONES, with a position for each element of V^l; the presence of a ONE signifies that the corresponding element is contained in the subset of V^l represented by the sequence. Given $C_1 = \{(a^i_\lambda, a^i_\mu), i = 1, 2 \cdots, n\}$, and $C_0 = \{(b^j_\lambda, b^j_\mu), j = 1, 2, \cdots, m\}$, we define B^i_λ to be the set of b^j_λ such that b^j_μ intersects a^i_μ. Then every vertex in a^i_λ is incompatible with every vertex in B^i_λ and no maximal compatible set can contain vertices from both sets. We represent a^i_λ and B^i_λ by ZERO-ONE vectors $\mathbf{A}^i$ and $\mathbf{B}^i$ having ONES exactly in those positions *not* corresponding to vertices in a^i_λ and B^i_λ, respectively. The computation of the maximal compatible sets proceeds as follows:

1) Let S^0 be a single vector having every element a ONE, representing the set of all vertices of V^l.
2) Do the following successively for $i = 1, 2, \cdots, n$:
   for each $s \in S^{i-1}$, form $s \cdot \mathbf{A}^i$ and $s \cdot \mathbf{B}^i$, where $\cdot$ denotes logical intersection. From this set of vectors, delete successively all vectors covered by other vectors (V *covers* W if V has a ONE wherever W does); the resulting set is S^i.
3) The sets represented by elements of S^n are the maximal compatible sets.

Example 2.3

	C_1										C_0					
	a^i_λ		a^i_μ					b^j_λ		b^j_μ						
1	1	1	x	0	1	x	0	0	x	1	x	x	0	x		
2	0	1	0	x	x	1	x	1	0	0	1	x	x	0		
3	1	1	x	0	x	0	x	1	1	x	1	1	0	1		
4	x	1	x	x	x	1	1	0	x	1	1	1	x	0		
5	1	x	x	x	0	x	1	0	0	x	x	x	x	0		
6	1	1	x	0	x	x	x	1	0	1	1	1	0	0		
								0	0	1	0	0	0	x		

i	a^i_λ		B^i_λ	$\mathbf{A}^i$				$\mathbf{B}^i$				S^i			
				00	01	10	11	00	01	10	11	00	01	10	11
1	1	1	0 x	1	1	1	0	0	0	1	1	1	1	1	0
			0 0									0	0	1	1
2	0	1	1 0	1	0	1	1	0	1	0	1	1	0	1	0
			0 0									0	0	1	1
												0	1	0	0
3	1	1	0 x	1	1	1	0	0	0	1	1	1	0	1	0
			0 0									0	1	0	0
			0 0									0	0	1	1
4	x	1	0 0	1	0	1	0	1	1	1	1	1	0	1	0
												0	1	0	0
												0	0	1	1
5	1	x	0 x	1	1	0	0	0	0	1	1	1	0	0	0
			0 0									0	1	0	0
												0	0	1	1
6	1	1	0 x	1	1	1	0	0	0	1	1	1	0	0	0
			0 0									0	1	0	0
			0 0									0	0	1	1

The maximal compatible sets are: {00}, {01}, {10, 11}. Once the maximal compatible sets have been obtained, it is necessary to select a minimum cover of V^l, i.e., a minimum number of maximal compatible sets having V^l as their union. This can be done for example, by means of the general extraction algorithm of Ref. 3. Given such a minimum cover consisting of the sets $N_1, N_2, \cdots, N_k$, we may obtain a cover with the same number of *disjoint* sets (and therefore a partition of V^l) as follows:

$$N'_1 = N_1; \quad \text{for } i > 1,$$

$$N'_i = N_i - (N_i \cap (N^1 \cup N^2 \cup \cdots \cup N^{i-1}).$$

• 2.4 *Single-output decomposition*

Usually a primitive element of the type encountered in practice realizes a single output function, although that function may be associated with several output lines. Accordingly, decompositions of the form $F(v, w) = G(\alpha(v), w)$ where α is a Boolean function, have special importance, and a special procedure will be given to determine whether they exist.

We shall take the existence of a single-output decomposition as a hypothesis, and, if no decomposition exists, will obtain a contradiction. It follows from this hypothesis that V^l can be partitioned into two sets of mutually compatible elements, and the following "antitransitivity" property is therefore implied: if $v_0 \nsim v_1$, and $v_1 \nsim v_2$, then $v_0 \sim v_2$.

We shall give a computational scheme based on this assumed property, and requiring calculation of the "incompatible pairs" a^i_λ, B^i_λ defined in 2.3. In what follows, P^i will denote a set $\{(p^1_0, p^1_1), (p^2_0, p^2_1), \cdots\}$ of pairs of covers such that every element of p^l_0 is incompatible with every element in p^l_1. The computation proceeds recursively according to the following rules:

1) P^0 is empty.
2) If any cover p_k^l in an element of P^{i-1} intersects both $a^i{}_\lambda$ and $B^i{}_\lambda$, a contradiction has been found and no decomposition exists. Otherwise, P^i is formed from P^{i-1} as follows:
   a) if $B^i{}_\lambda$ is empty, $P^i = P^{i-1}$.
   b) if neither $a^i{}_\lambda$ nor $B^i{}_\lambda$ intersects any p_l, the new pair $(a^i{}_\lambda, B^i{}_\lambda)$ is simply adjoined to the list: $P^i = P^{i-1} \cup \{(a^i{}_\lambda, B^i{}_\lambda)\}$.
   c) if some intersection does exist, all pairs intersecting either $a^i{}_\lambda$ or $B^i{}_\lambda$ are replaced by a single pair $(a^i{}_\lambda \cup S^i)$, $(B^i{}_\lambda \cup T^i)$, where
   $S^i = \{p_k^l | p_k^l$ intersects $a^i{}_\lambda$ or p_{1-k}^l intersects $B^i{}_\lambda\}$
   $T^i = \{p_k^l | p_k^l$ intersects $B^i{}_\lambda$ or p_{1-k}^l intersects $a^i{}_\lambda\}$.

If the process terminates without a contradiction, then a decomposition exists for any Boolean function α such that, if $v_0 \in p_k^l$ and $v_1 \in p_{1-k}^l$, $\alpha(v_0) \neq \alpha(v_1)$.

Example 2.4

We treat the problem of Example 2.3:

i	$a^i{}_\lambda$	$B^i{}_\lambda$	P^i
1	1 1	0 x	$\{(11, 0x)\}$
2	0 1	x 0	

p_1^1 intersects both $a^2{}_\lambda$ and $B^2{}_\lambda$; no single-output decomposition exists.

Example 2.5

	C_1						C_0					
	a	b	c	d	e	f	a	b	c	d	e	f
1	x	x	0	1	x	x	0	x	0	0	0	x
2	x	x	0	x	1	x	x	1	0	0	0	x
3	x	x	1	0	0	x	x	x	0	0	0	0
4	1	1	1	0	x	x	1	0	1	x	1	x
5	0	1	x	1	0	x	1	1	1	1	x	x
6	1	0	x	1	0	x	0	x	1	0	1	1
							x	0	1	0	1	x
							x	1	1	1	1	0

a) $\lambda = \{a, b, c\}$, $\mu = \{d, e, f\}$.

i	$a^i{}_\lambda$	$B^i{}_\lambda$	P^i
1	x x 0	1 0 1 1 1 1 x 1 1	$\left\{\begin{pmatrix} x & x & 0 & 1 & 0 & 1 \\ & & & x & 1 & 1 \end{pmatrix}\right\}$
2	x x 0	1 0 1 1 1 1 0 x 1	$\left\{\begin{pmatrix} x & x & 0 & 0 & x & 1 \\ & & & x & 0 & 1 \\ & & & x & 1 & 1 \end{pmatrix}\right\}$
3	x x 1	x 0 1 x 1 1 0 x 0 x 1 0 x x 0	$\{(x\ x\ 0 \quad x\ x\ 1)\}$
4	1 1 1	0 x 0 0 x 0 x x 0 1 0 1 0 x 1 x 0 1	x x 1 intersects both $a^4{}_\lambda$ and $B^4{}_\lambda$; no single-output decomposition exists.

b) $\lambda = \{a, b, c\}$, $\mu = \{d, e, f\}$

i	$a^i{}_\lambda$	$B^i{}_\lambda$	P^i
1	x x 0	empty	empty
2	x x 0	empty	empty
3	x x 1	empty	empty
4	1 1 1	1 0 1 0 x 1 x 0 1	$\left\{\begin{pmatrix} 1 & 1 & 1, & 0 & x & 1 \\ & & & & x & 0 & 1 \end{pmatrix}\right\}$
5	0 1 x	1 1 1	$\left\{\begin{pmatrix} 1 & 1 & 1, & 0 & x & 1 \\ & & & & x & 0 & 1 \end{pmatrix}\right\}$
6	1 0 x	1 1 1	$\left\{\begin{pmatrix} 1 & 1 & 1, & 0 & x & 1 \\ & & & & x & 0 & 1 \end{pmatrix}\right\}$

For any Boolean function α such that $\alpha(0, 0, 1) = \alpha(0, 1, 1) = \alpha(1, 0, 1) \neq \alpha(1, 1, 1)$, there is a decomposition of the form $G(\alpha(a, b, c), c, d, e, f)$.

• 2.5 *Vertex functions*

Primitive devices constructed of diodes, transistors, or vacuum tubes usually realize Boolean functions of a particular type—the vertex functions; α is said to be a *vertex function* if there is a *distinguished vertex* v_0 such that $\alpha(v_0) \neq \alpha(v)$, where v is any other vertex. The AND, OR, NAND, and NOR are vertex functions; the EXCLUSIVE OR and MAJORITY are not. It is particularly easy to give conditions for the existence of "vertex-type" decompositions.

Lemma 2. There exists a decomposition $F(v, w) = G(\alpha(v), w)$, where α is a vertex function, if and only if there is a vertex v_0 such that, whenever $a^i{}_\lambda$ and $B^i{}_\lambda$ are both nonempty, either $a^i{}_\lambda = v_0$ or $B^i{}_\lambda = v_0$.

Thus, in Example 2.5a, $a^i{}_\lambda = (x\ x\ 0)$, $B^1{}_\lambda = \begin{pmatrix} x & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$; we can assert, by Lemma 2, that no single-output vertex-type decomposition exists. In Example 2.5b there is a single-output vertex-type decomposition with $v_0 = 1\ 1\ 1$.

Decompositions involving several vertex functions are characterized by the following lemma.

Lemma 3. There exists a decomposition of the form $F(v, w) = G(\alpha_1(v), \alpha_2(v), \cdots, \alpha_t(v), w)$, in which the α_i are vertex functions, if and only if V^l has a subset of $2^l - t$ mutually compatible elements.

Proof. Let v_i denote the distinguished vertex for α_i. Then two elements of V^l yield the same values for $\alpha_1, \alpha_2, \cdots, \alpha_t$ if and only if neither element is one of the v_i. By Proposition 1, therefore, the decomposition is valid if and only if any two elements of $V^l - \{v_i\}$ are compatible. The given hypothesis is clearly necessary and sufficient for the existence of a suitable set of v_i.

• 2.6 *The image of a decomposition*

Through the use of Propositions 1 and 2 it has been possible to characterize decompositions of a function F without explicitly specifying the image G. In the synthesis algorithm of the next section, however, we shall be concerned with sequences of decompositions,

each operating on the image of its predecessor, and the computation of the image will therefore be an essential link in the process. The computation of covers D_1 and D_0 specifying G, given C_1 and C_0, covers for F, and the functions $\alpha_1, \alpha_2, \cdots, \alpha_t$, is a simple mechanical process. First we consider the disjunctive case. For each cube (a^i_λ, a^i_μ) in C_1, let A^i_λ be the set of values assumed by $\alpha_1, \alpha_2, \cdots \alpha_t$ for arguments in a^i_λ. Then D_1, the ON-array for G, is given by $\{A^i_\lambda \times a^i_\mu\}$; D_0 is formed in a similar way from C_0.

Example 2.6. The following function exhibits a decomposition of the form $F(a, b, c, d, e) = G(\alpha_1(a, b), \alpha_2(a, b), c, d, e)$, where $\alpha_1(a, b) = a \cdot b$, and $\alpha_2(a, b) = \bar{a} \cdot \bar{b}$.

	C_1					C_0				
	a	b	c	d	e	a	b	c	d	e
1	x	x	0	0	x	0	0	x	1	x
2	0	1	x	1	x	1	1	x	1	x
3	1	0	x	1	x	x	1	1	0	x
4	0	0	1	0	x	1	x	1	0	x

The computation of D_1 is as follows:

i	a^i_λ		A^i_λ			
	a	b	α_1	α_2		
1	x	x	0	0		
			0	1		
			1	0		
2	0	1	0	0	=	0 x
3	1	0	0	0		x 0
4	0	0	0	1		

D_1 is given by the following cover:

α_1	α_2	c	d	e
0	x	0	0	x
x	0	0	0	x
0	0	x	1	x
0	0	x	1	x
0	1	1	0	x

Note that, although C_1 cannot be simplified, the third and fourth cubes of D_1 are identical. In this example, D_0 is as follows:

α_1	α_2	c	d	e
0	1	x	1	x
1	0	x	1	x
x	0	1	0	x

The image of a nondisjunctive decomposition may be computed by applying the above procedure to a derived disjunctive problem, obtained either by forming the function $\hat{F}$ or by treating the variables in $\lambda \cap \mu$ as functions α_i associated with the decomposition. In Example 2.2, we found a decomposition of the form $F(a, b, c, d) = G(a \cdot b, b, c, d)$. Covers for the given function F and the image G are as follows:

F

C_1				C_0			
a	b	c	d	a	b	c	d
1	0	1	0	1	1	x	1
1	x	1	x	1	x	0	x
0	1	x	x	x	0	0	x

G

D_1				D_0			
$a \cdot b$	b	c	d	$a \cdot b$	b	c	d
0	0	1	x	1	1	x	1
1	1	1	0	0	0	0	x
0	1	x	x	1	1	0	x

III. The synthesis algorithm

The fundamental processes required to detect decompositions and to compute their images have now been developed. In this section we show how these processes can be employed as "subroutines" in the determination of minimum-cost Boolean graphs.

The first step is to establish precisely the description of Boolean graphs by sequences of decompositions. Since Boolean graphs are acyclic, any Boolean graph must contain at least one primitive element for which no input is the output of another element. Thus, any Boolean graph may be shown schematically as in Fig. 6.

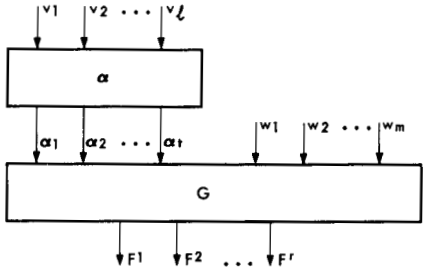


Figure 6 **Decomposition of a Boolean graph.**

Here $v_1, v_2, \cdots, v_l$ are the inputs to the element α; $\alpha_1, \alpha_2, \cdots, \alpha_t$ are the distinct output functions of α (possibly fewer in number than the distinct output lines, since fan-out is allowed), and $w_1, w_2, \cdots, w_m$ are the distinct primary inputs to the elements of the graph other than α. The indicated decomposition of the graph is closely related to the algebraic decompositions of the output functions F^i—for each such function must be expressible as

$$F^i = G^i(\alpha_1(v_1, \cdots, v_l), \cdots, \alpha_t(v_1, \cdots, v_l), w_1, w_2, \cdots w_m).$$

Any Boolean graph may, in fact, be specified completely by a sequence of decompositions. Figure 7 shows a single-output Boolean graph, and (4) gives a sequence of decompositions specifying the graph.

$$\begin{aligned} F(a, b, c, d) &= G_1(\alpha_1(a, b, d), b, c, d) \\ G_1(\alpha_1, b, c, d) &= G_2(\beta_1(b, d), \beta_2(b, d), \alpha_1, c) \\ G_2(\beta_1, \beta_2, \alpha_1, c) &= G_3(\gamma_1(\alpha_1, \beta_1), \beta_1, \beta_2, c) \qquad (4) \\ G_3(\gamma_1, \beta_1, \beta_2, c) &= G_4(\delta_1(\beta_2, c), \gamma_1, \beta_1) \\ G_4(\delta_1, \gamma_1, \beta_1) &= j(\varepsilon_1(\gamma_1, \beta_1, \delta_1)), \quad \text{where } j(x) \equiv x . \end{aligned}$$

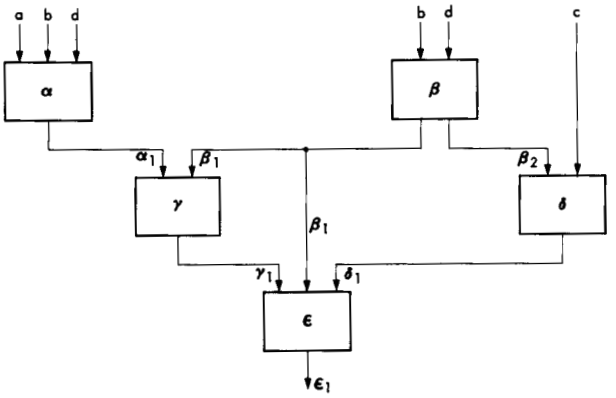


Figure 7 **A single-output Boolean graph.**

Each decomposition corresponds to "factoring off" an element of the graph, and the successive G_i are the functions realized by the remaining subgraphs. There are, altogether, four distinct sequences of decompositions which describe this graph, since the element β could have been taken before α, or δ, before γ. In general, a single-output Boolean graph can be specified by a sequence (perhaps not unique) of the form

$$\begin{aligned} F(v_0, w_0) &= G_1(\alpha_1^1(v_0), \cdots, \alpha_{t_1}^1(v_0), w_0) \\ G_1(v_1, w_1) &= G_2(\alpha_1^2(v_1), \cdots, \alpha_{t_2}^2(v_1), w_1) \\ &\vdots \\ G_n(v_n) &= j(\alpha_1^{n+1}(v_n)) . \end{aligned} \tag{5}$$

It can be shown by an inductive argument that any sequence of the form (5) describes some Boolean graph provided that

a) each set $\{\alpha_1^i, \alpha_2^i, \cdots \alpha_{t_i}^i\}$ consists of the distinct output functions of some element $\alpha^i \in \mathfrak{B}$,
b) fan-out restrictions are satisfied; this is easily checked, since the number of lines on which a given signal appears is just the number of v_i in which it occurs.

Any sequence of the form (5) satisfying (a) and (b) is called *admissible*. A sequence satisfying all these conditions, except that the final image is not of the form $j(\alpha_1^{n+1}(v_n))$, is called an *admissible subsequence*.* An admissible sequence or subsequence is called *valid* for a given function F if and only if the indicated decompositions of F and its successive images $G_1, G_2, \cdots, G_n$ exist, according to the tests developed in Section II. The determination of a minimum-cost Boolean graph realizing F now reduces to the following: find a minimum-cost sequence valid for F.

Before giving the synthesis algorithm, we make some preliminary assumptions. We assume that each primitive element has a positive integral cost, that the cost of a Boolean graph is the sum of the costs of its primitive elements, and that there is an *a priori* upper bound M_0 on $C(F)$, the minimum cost of realizing F. We assume also the existence of some alphabetic ordering of the possible decompositions of any given function, (each specified by a triple (α, λ, μ). Finally, we assume the existence of a means of obtaining, for any Boolean function G_i, $l(G_i)$, a lower bound on $C(G_i)$. Many such lower bounds are available:

1) $l(G_i) \equiv 0$.
2) The bound obtained in Appendix A of Ref. 4, which is determined by the number of variables on which G_i depends.
3) If G_i is a function of three or fewer variables, the value given in a table of optimum realizations of three-variable functions, obtained either by the algorithm to be presented, or by a procedure due to M. H. McAndrew.[11] McAndrew's procedure also gives lower bounds for the costs of many functions of more than three variables.
4) $l(G_i) = \max_{H \in L(G_i)} l(H)$, where $L(G_i)$ is the set of three-variable subfunctions of G_i obtained by setting primary inputs to constant values or setting primary inputs equal to each other.

The synthesis procedure consists of a "tree search", which traces through the valid sequences in alphabetic order, seeking a realization of minimum cost. These sequences are built up one decomposition at a time. Thus, at a general stage in the process, we consider some valid subsequence S which may be represented as follows:

$$F \xrightarrow{(\alpha^1, \lambda_0, \mu_0)} G_1 \xrightarrow{(\alpha^2, \lambda_1, \mu_1)} G_2 \rightarrow \cdots \rightarrow G_{k-1} \xrightarrow{(\alpha^k, \lambda_{k-1}, \mu_{k-1})} G_k .$$

Certain tests can be applied to determine whether sequences containing this subsequence need be considered. First, we wish to prevent the formation of several different subsequences which specify the same circuit structure, but take the primitive elements in different orders. Simple tests can be given which eliminate all but one member (the first alphabetically) of such a class of equivalent sequences.

The second means of terminating subsequences is through the use of cost bounds. Suppose that

$$\text{cost}(\alpha^1) + \text{cost}(\alpha^2) + \cdots + \text{cost}(\alpha^k) + l(G_i) > M,$$

where M is the best available upper bound on $C(F)$. Then any valid sequence for F, obtained by extending the subsequence S, will yield a circuit costing more than M, so that S need not be considered further. This use of cost bounds is highly important; for the termination of a subsequence at a relatively early point prevents the formation of an enormous number of uneconomical realizations of F.

Whenever a subsequence S is terminated, its final decomposition $(\alpha^k, \lambda_{k-1}, \mu_{k-1})$ is deleted, and the next decomposition in alphabetic order applicable to G_{k-1} is sought. If no such decomposition is found, the process "backs up" to G_{k-2}, and so on until some valid decomposition is found; decompositions are then

* Admissible sequences and admissible subsequences are analogous, respectively, to the projective words and partial projective words of Ref. 4.

sought which extend the subsequence terminating in this new decomposition.

The synthesis procedure is clearly finite, since, under the given cost assumptions, the number of subsequences having costs $\leqq M_0$ is finite. Moreover, the procedure yields all Boolean graphs of minimum cost, since every valid sequence is either generated or ruled out by a cost bound when partially formed.

The minimization algorithm is indeed laborious, but only to a degree commensurate with the difficulty of the problem solved. The authors know of no other systematic synthesis procedure which approaches the generality of this algorithm. Moreover, the algorithm is of definite practical value. When the program described in Ref. 6 assumes its final form it is expected to determine a minimum-cost circuit for nearly any four-variable problem in less than ten minutes on the IBM 7090. Many problems in five and six variables should also admit of solutions in reasonable lengths of time.

In larger problems it will usually be necessary to accept an approximate solution. Fortunately, it is possible to arrange the alphabetic ordering of decompositions so that good approximate solutions tend to occur early in the computation. For example, priority can be given to a decomposition represented by the triple (α, λ, μ) if the overlap between λ and μ is small, or if α is an inexpensive element of $\mathfrak{B}$. Through the use of such heuristics, it has proved possible in many cases to obtain circuits which, although they are not known to be of minimum cost, appear quite economical, and could not have been obtained by any other systematic procedures known to the authors.

The following simple example illustrates some aspects of the algorithm,

Example 3.1. Suppose that the function $F(a, b, c, d)$ to be realized in the following:

C_1				C_0			
a	b	c	d	a	b	c	d
0	0	x	0	x	x	x	1
x	x	1	0	1	x	0	x
				x	1	0	x

and that $\mathfrak{B}$ is specified as follows:

	Function	*Cost*
α	$\bar{x}\bar{y}$	1
β	$x\bar{y}$	2
γ	$\bar{x}y$	2
δ	$x \cup y$	2
ε	$x \cup \bar{y}$	3
ζ	$\bar{x} \cup y$	3
η	xy	3
θ	$\bar{x} \cup \bar{y}$	4

Clearly, any function of n variables requires at least $n - 1$ elements for its synthesis, and a lower bound l on the cost of a function of n variables is $n - 1$.

We assume that decompositions are considered in increasing order of the number of variables in $\lambda \cap \mu$, and, within this criterion, in order of increasing cost.

Thus the first valid sequence of decompositions considered is the following:

$$F(a, b, c, d) = G_1(\alpha(a, b), c, d)$$
$$= G_2(\alpha(\alpha(a, b), c), d) = j(\alpha(\alpha(\alpha(a, b), c), d)) .$$

This sequence defines the Boolean graph of Fig. 8, which has a cost of 3. Then $M = 3$ is an upper bound on $C(F)$, and the cost bound rules out all decompositions except those such that $\lambda \cap \mu$ is empty and the primitive element is α. No further decompositions of this kind can be found, and the circuit of Fig. 8 is the unique minimum-cost realization of F.

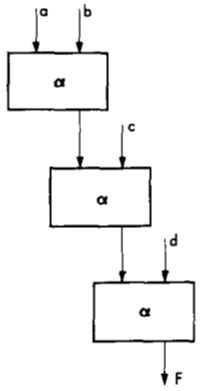


Figure 8 **A minimum-cost Boolean graph.**

The synthesis procedure for Boolean graphs with several terminal outputs follows the lines of the single-output procedure, but is considerably more complex. The essential reason for the increased complexity is that, in general, each output function will not depend on all the signals generated within the circuit. Thus, in the Boolean graph of Fig. 4, the output function F^3 depends on α_2, γ_1, and ζ_1, but not on α_1, β_1, or δ_1.

Thus the influence of any given element must be specified separately for each output function. This is done through the use of *multiple decompositions* having several *components*, which are themselves decompositions of individual output functions. All the components of a given multiple decomposition specify the same element $\omega \in \mathfrak{B}$ applied to the same λ-part, but the μ-part is specified independently for each component, and a different subset of the outputs of ω may apply to each terminal output. If this subset of outputs is empty, for a given terminal output, then the associated component simply represents an identity function, and can be omitted. The Boolean graph of Fig. 4 is described by the following sequence of multiple decompositions:

$$F^1(a, b, d) = G_1(\alpha_1(a, b), d)$$
$$F^2(a, b, c) = G_1^2(\alpha_1(a, b), \alpha_2(a, b), c)$$
$$F^3(a, b, c, e) = G_1^3(\alpha_2(a, b), a, c, e)$$

$$G_1^1(\alpha_1, d) = j(\beta_1(\alpha_1, d))$$

$$G_1^2(\alpha_1, \alpha_2, c) = G_2^2(\gamma_1(\alpha_2, c), \alpha_1, c)$$

$$G_1^3(\alpha_2, a, c, e) = G_2^3(\gamma_1(\alpha_2, c), a, e)$$

$$G_2^2(\gamma_1, \alpha_1, c) = G_3^2(\delta_1(\alpha_1, c), \gamma_1)$$

$$G_3^2(\delta_1, \gamma_1) = j(\varepsilon_1(\delta_1 \cdot \gamma_1))$$

$$G_3^2(\gamma_1, a, e) = j(\zeta_1(\gamma_1, a, e)) .$$

A sequence of multiple decompositions is called *admissible* if each of its component sequences is admissible and fan-out restrictions are satisfied, and *valid* if it is admissible and each of its component sequences is valid for the associated terminal output. The cost associated with such a sequence is *not* the sum of the costs of its component sequences, but rather the sum of the costs of the primitive elements associated with multiple decompositions in the sequence.

The synthesis problem for Boolean graphs with several outputs thus reduces to the determination of all valid sequences of minimum cost. An alphabetic search procedure patterned after the single-output procedure can be used for this calculation. The number of valid sequences of multiple decompositions is very great, however, and hence the time required for strict minimization tends to grow quite large.

Acknowledgments

The authors would like to express their thanks to M. H. McAndrew, who made substantial contributions to the developments reported in this paper, and to F. E. McFarlin and J. R. Wilts, who suggested many improvements and innovations.

References

1. J. P. Roth, "Algebraic Topological Methods for the Synthesis of Switching Systems. I," *Transactions of American Mathematical Society*, **88**, 301-326 (1958).
2. J. P. Roth, "Algebraic Topological Methods in Synthesis," *Proceedings of an International Symposium on the Theory of Switching*, Harvard University, April 2-5, 1957. *The Annals of the Harvard Computation Laboratory, XXIX*, Harvard University Press, Cambridge, Massachusetts, 1959, pp. 57-73.
3. J. P. Roth and E. G. Wagner, "Algebraic Topological Methods for the Synthesis of Switching Systems III. Minimization of Non-Singular Boolean Trees," *IBM Journal*, **4**, No. 4, 326-344, (1959).
4. J. P. Roth, "Minimization over Boolean Trees," *IBM Journal*, **4**, No. 5, 543-558, (1960).
5. R. L. Ashenhurst, "The Decomposition of Switching Functions," *Proceedings of an International Symposium on the Theory of Switching*, Harvard University, April 2-5, 1957. *The Annals of the Harvard Computation Laboratory XXIX*, Harvard University Press, Cambridge, Massachusetts, 1959, pp. 74-116.
6. R. M. Karp, F. E. McFarlin, J. P. Roth, and J. R. Wilts, "A Computer Program for the Synthesis of Combinational Switching Circuits," *Proceedings of the Second AIEE Symposium on Circuit Switching Theory and Logical Design*, October, 1961, pp. 182-194.
7. *Added in proof*: Cf. J. M. Galey, R. E. Norby, and J. P. Roth, "Techniques for the Diagnosis of Switching Circuit Failures," *Proceedings of the Second AIEE Symposium on Switching Circuit Theory and Logical Design*, October 1961, pp. 152-162.
8. R. L. Ashenhurst, "Non-Disjoint Decomposition," Harvard Computation Laboratory Report No. BL-4, Sec. IV, 1953.
9. H. A. Curtis, "Simple Non-Disjunctive Decomposition," Harvard Computation Laboratory Report No. BL-19, Sec. II, 1958.
10. R. E. Miller, "State Reduction for Sequential Machines," IBM Research Report RC-121, June 15, 1959.
11. M. H. McAndrew, private communication.
12. H. A. Curtis, "A Generalized Tree Circuit," *Jour. ACM*, **8**, 484-496 (1961).

Received April 14, 1961