MINIMIZING THE NUMBER OF THRESHOLD GATES IN AN ERROR CORRECTING NETWORK USING MULTIPLEXING TECHNIQUES

A Thesis

Presented to

the Faculty of the Department of Electrical Engineering

The University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

---

by

David L. Cauthron

May 1969

ACKNOWLEDGEMENT

I would like to express my deep appreciation to Dr. J. D. Bargainer, Jr., for his many suggestions and his encouragement during the preparation of this thesis.

I would like to thank Mr. Bob Loofbourrow and Mr. Don Howlett, Texaco Inc., Bellaire, Texas, for their understanding in the completion of this thesis.

I would like to thank Mrs. Tommy Weigle for her excellent work in typing this thesis.

MINIMIZING THE NUMBER OF THRESHOLD GATES IN AN ERROR CORRECTING NETWORK USING MULTIPLEXING TECHNIQUES

An Abstract of a Thesis

Presented to

The Faculty of the Department of Electrical Engineering

The University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

---

by

David L. Cauthron

May, 1969

ABSTRACT

In this thesis, three methods were discussed for designing error-correcting capabilities into threshold gate networks so that the logic gates themselves will correct errors of other gates. These methods are extensions of work presented by Bargainer and Coates (1).

Method II is a procedure for taking a given threshold logic network and finding the minimum number of gates required to correct t errors using multiplexing techniques. It requires that all weights remain the same as read in. Method III is also a procedure for taking a given threshold logic network and finding the minimum number of gates required to correct t errors using multiplexing techniques. However, in Method III a search is performed over the weights for one gate feeding into another gate (β_{ij}) in an attempt to optimize the β_{ij} values.

Both Method II and Method III modify a given realization by the addition of redundant gates to obtain an error-correcting network. The methods require that an error of any specified number of gates be corrected in the next level of logic. This is known as a multiplexed realization. Errors of the output gate are not corrected.

The procedure developed, in this paper, for both Method II and Method III requires the minimization of some cost function. The cost function consists of the sum of all gates required to correct a specified number of errors, plus an error factor due

to the constraints not being satisfied. The number of gates k_i in each set $\{A_i\}$ are adjusted by a multi-dimensional search technique with the minimization of the cost function as a performance criterion. The search is accomplished by means of a digital computer program. Method III goes one step further than Method II and an attempt is made to reduce the number of gates even further by searching for better weights for inputs from other gates.

The results of problems run by both methods are shown in Chapter V. Method II finds the minimum number of gates necessary to correct the specified number of errors, using the weights of the original realization. Method III finds the minimum number of gates necessary to correct the specified number of errors, using an optimum value for the weights of inputs from other gates. Either method may have a minimum gap for all gates specified, to insure that the solution have gates with a reasonable gap width. The procedure to specify a minimum gap is also presented in Chapter V.

TABLE OF CONTENTS

CHAPTER I

THRESHOLD LOGIC

The input-output relationship for a digital network is expressed as a logical function of binary valued inputs and outputs. A threshold gate has binary inputs X_1, X_2, X_3, ... X_n and a binary output y. Associated with each X_i is an internal weight a_i. Each threshold gate has a threshold value T and a separating function f(p) defined in the following manner:

$$f(p) = \sum_{i=1}^{n} a_i X_i(p). \tag{1.1}$$

$X_i(p)$ is the value of X_i at $p \in \{0,1\}^n$ and the sum and product operations are the usual arithmetic ones. The output y at each $p \in \{0,1\}^n$ is determined by

$$y = 1 \text{ if } f(p) \geq T$$
$$y = 0 \text{ if } f(p) < T. \tag{1.2}$$

Although the definition of the threshold gate might lead one to believe that all of its inputs are independent variables, this is not a necessary requirement. In general, a threshold gate may have n+m inputs, some of which are independent variables and some of which are the outputs of other threshold gates, where n inputs are independent variables and m inputs are the outputs of other threshold gates. The equation for the gate output can be expressed as in Equation 1.3.

$$y = \left\langle \sum_{i=1}^{n} a_i X_i(p) + \sum_{j=1}^{m} a_j y_j(p) \right\rangle_T \tag{1.3}$$

where the law of operation is defined by Equation 1.2. For the convenience of notation, the weight associated with each independent

input X_i will be called a_i, and the weight associated with each input y_i will be called β_i.

For the remainder of the paper, it will be assumed that all threshold gates have non-negative weights. This does not restrict the results. For any realization with one or more negative weights there is an equivalent realization having all positive weights.

The threshold gate has a binary-valued output for each combination of the binary-valued inputs; therefore the output y is a switching function of the input variables. The Boolean-function representation is:

$$y = F(X_1, X_2, \ldots . X_n) \tag{1.4}$$

in which the value of the function is expressed in terms of the independent variables X_i and the Boolean operators +, ., and $\bar{}$. Therefore, y_i and $F_i(p)$ are equivalent. The form of the function may imply an associated realization.

Each equation for the separating function implies a particular threshold gate realization. Therefore, the notation for the separating function can be directly related to the realization. Equation 1.5 implies the gate with threshold T, input variables X_i, and corresponding weights a_i.

$$\langle\langle F(X_1, X_2, \ldots . X_n)\rangle\rangle_T = \left\langle \sum_{i=1}^{n} a_i X_i \right\rangle_T \tag{1.5}$$

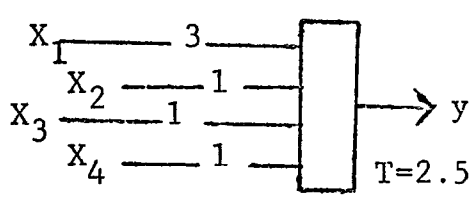


Figure 1.1. A single gate threshold realization.

Example 1.1. For the threshold gate shown in Figure 1.1, the output expressed in terms of the separating function is

$$y = \langle 3X_1+X_2+X_3+X_4 \rangle_{T=2.5} \quad .$$

The output expressed as a Boolean function of the input variables is

$$y = \overline{X}_1X_2X_3X_4 + X_1\overline{X}_2\overline{X}_3\overline{X}_4 + X_1\overline{X}_2\overline{X}_3X_4 + X_1\overline{X}_2X_3X_4$$
$$+ X_1X_2\overline{X}_3\overline{X}_4 + X_1X_2\overline{X}_3X_4 + X_1X_2X_3\overline{X}_4 + X_1X_2X_3X_4$$

This Boolean function can be simplified to the form

$$y = X_1 + X_2X_3X_4 \quad .$$

There are many other separating-function realizations which will realize the same function. Another example which will realize the same function is

$$y = \langle 6X_1 + 2X_2 + 2X_3 + 2X_4 \rangle_6 \quad .$$

An alternate rule of operation for the threshold gate can be stated using the following definition. Define

$$u = \min \{f(p) | F(p) = 1\} \text{ and}$$
$$\ell = \max \{f(p) | F(p) = 0\} \quad .$$

The interval having an upper boundary u and a lower boundary ℓ is referred to as the gap of the realization. Equation 1.3 requires that the threshold T be restricted as follows:

$$\ell < T \leq u. \qquad (1.7)$$

The Boolean function that is realized by a threshold gate with separating function f and gap u:ℓ is represented in the following manner:

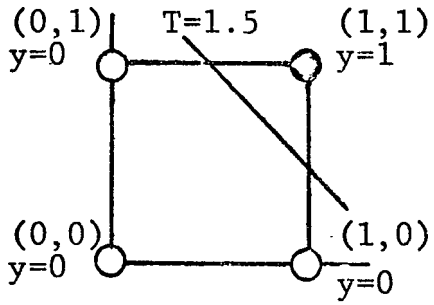
$$y = F(X_1, X_2, X_3, \ldots . X_n) = f\left\langle \sum_{i=1}^{n} a_i X_i \right\rangle u{:}\ell . \qquad (1.8)$$

Any Boolean function is defined over an n-dimensional Euclidean space where each coordinate axis corresponds to one of the independent variables. Each combination of values for the independent variables plots as a vertex of a unit cube in the n space. The function F assigns to each $p = p_0, p_1, \ldots, p_2^{n-1}$ a corresponding value F(p) which is either 1 or 0. P_0 and P_1 are defined to be the subsets of the vertices of the n cube for which F(p) is equal to 0 and 1 respectively.

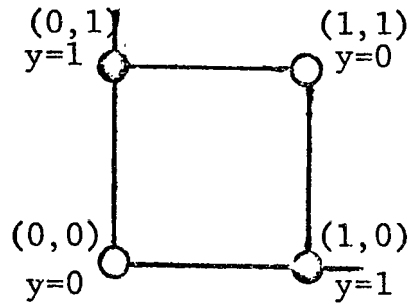
From the rule of operation established by Equations 1.1 and 1.8, the P_0 vertices of the n cube are separated from the P_1 vertices by the hyperplane whose equation is

$$\sum_{i=1}^{n} a_i X_i = T$$

where $X_i = 0,1$. Figure 1.2a and 1.2b show a separating plane for a two variable function.



(a) $y = X_1X_2$



(b) $y = X_1\overline{X}_2 + \overline{X}_1X_2$

Figure 1.2.(a) A linearly separable function
and (b) A function which is not linearly separable.

Not all functions can have the 2^n vertices of the n cube separated into P_0 and P_1 sets by a hyperplane. Therefore, not all functions may be realized by a single threshold gate. However, these functions can be realized by a network of threshold gates.

Example 1.2. The realizations for the functions illustrated by Figure 1.2 are shown in Figure 1.3. The AND function is represented in Figure 1.2a and may be realized with one gate. There is an allowable region so that with a threshold placed in this region the P_0 vertices will be separated from the P_1 vertices. The region is defined by u = 2 and ℓ = 1 and is represented by

$$1 < T \leq 2.$$

By placing a threshold anywhere in this region, the function can be realized.

The function pictured in Figure 1.2b is not separable. There is no allowable region where the threshold could be placed and separate all the P_1 vertices from all the P_0 vertices. This function may be realized with two threshold gates.

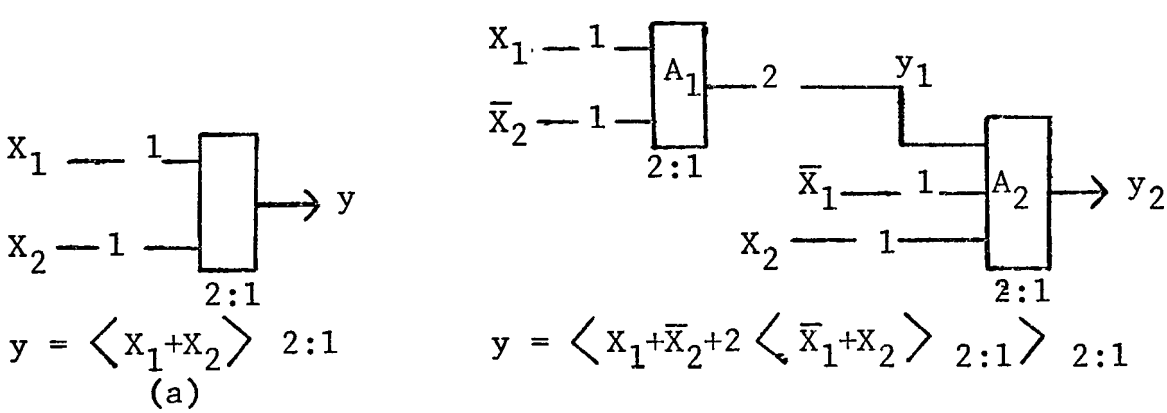


Figure 1.3(a) Linearly separable realization
and (b) A function which is not linearly separable.

The AND and OR functions can be realized by threshold gates. The general representation of the AND and OR functions respectively is as follows:

$$F(X_1,X_2,....X_n) = X_1X_2...X_n = \langle X_1+X_2+...+X_n \rangle_{n:n-1} , \quad (1.9)$$

$$F(X_1,X_2,....X_n) = X_1+X_2+...+X_n = \langle X_1+X_2+...+X_n \rangle_{1:0} . \quad (1.10)$$

This implies that any function can be realized by a network of threshold gates requiring no more gates than would be necessary using conventional logic.

One advantage of using threshold logic over conventional logic is the significant reduction in the number of gates required. In Example 1.1 the function would require one AND and one OR gate for the realization, while only one threshold gate is required. For more complex functions, the reduction in number of gates becomes much more significant.

Every threshold gate has a specific separating function f and a logical function F both of which are defined on the vertices of the n cube. For each vertex p there is an associated ordered triplet [p, f(p), F(p)] where f(p) is some real number and F(p) = 0 or 1. The set {[p, f(p), F(p)]} of ordered triplets is known as the map of F by f. A convenient visualization of the map is the line graph shown in Figure 1.4 for the function of Example 1.1, where

$$F(X_1,X_2,X_3,X_4) = X_1 + X_2X_3X_4$$

$$f(X_1,X_2,X_3,X_4) = \langle 3X_1 + X_2 + X_3 + X_4 \rangle_{3:2} .$$

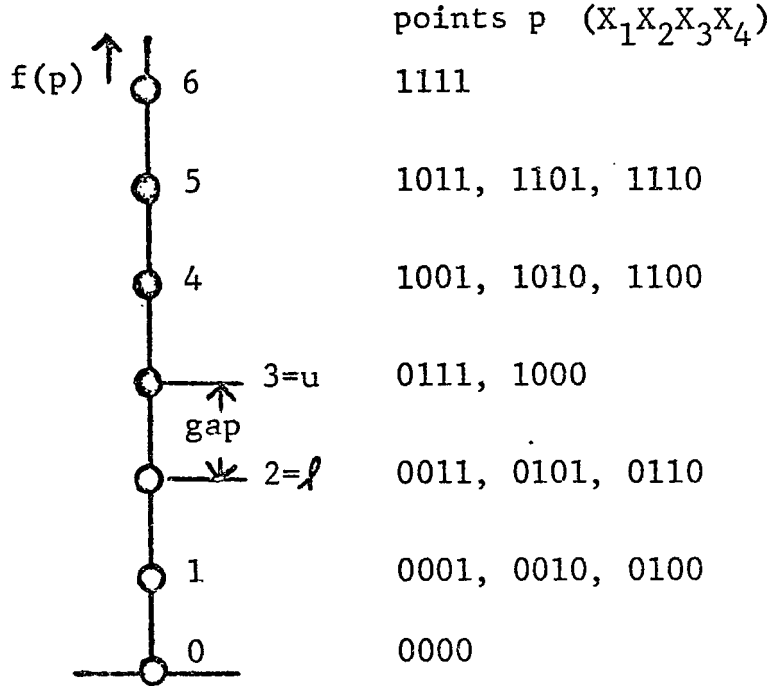


Figure 1.4. Line graph of Example 1.1.

For each p, a point is located on the real line at location f(p). The point is represented by ○ when F(p) = 0 and ● when F(p) = 1. The map is the result of the separating function f mapping the vertices of the unit cube in n space onto the real line. Each f(p) is called the image of p.

A map is divided into disjoint subsets, called the zero and unit parts, corresponding to F(p) = 0 or 1. As indicated earlier, u is the smallest f(p) where F(p) =1 and ℓ is the largest f(p) where F(p) = 0. Then u and ℓ comprise the gap of the function denoted by u:ℓ. Since the threshold may be any value within the range

$$\ell < T \leq u$$

and still realize the function, it is common to indicate a range in the equation of the separating function rather than specify a particular threshold T. In Example 1.1 the output in terms of the gap of the separating function would be expressed as in Equation 1.11 instead of in terms of a particular threshold value.

$$y = \langle 3X_1 + X_2 + X_3 + X_4 \rangle_{3:2} \quad . \qquad (1.11)$$

The above separating functions imply single threshold gate realizations, but the notation for multigate networks is quite similar. Any arbitrary logical function may be realized by an interconnection of threshold gates. The final gate in such a network will, in general, have independent inputs $X_1, X_2, \ldots . X_n$, inputs from the output of other gates $y_1, y_2, \ldots . y_m$, and a single output y of its own. In the one-gate case, each separating function representation implies a threshold gate realization. Therefore, such a representation is a realization of the logical function.

Example 1.3 represents a typical multiple gate function. The realization of the Boolean function is shown in Figure 1.5.

Example 1.3. A multiple gate function with a separating function as follows:

$$f(p) = \langle X_1 + X_2 + \overline{X}_3 + 2X_5 + 5 \langle 2\overline{X}_1 + 2\overline{X}_2 + X_3 + \overline{X}_4 + 2X_5 + 5 \langle X_1 + X_3 + X_4 + X_5 \rangle_{4:3} \rangle_{7:6} + 5 \langle \overline{X}_1 + X_2 + \overline{X}_3 + \overline{X}_5 \rangle_{4:3} \rangle_{5:4} \quad .$$

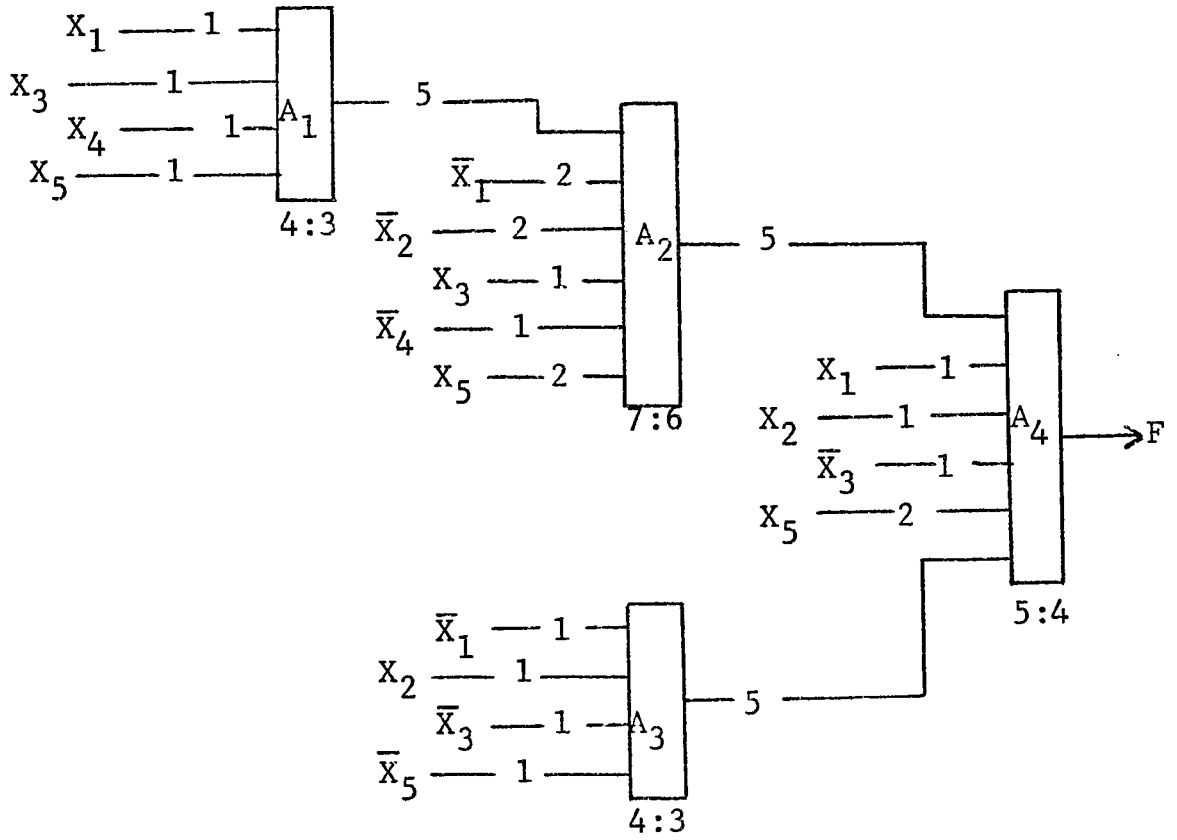
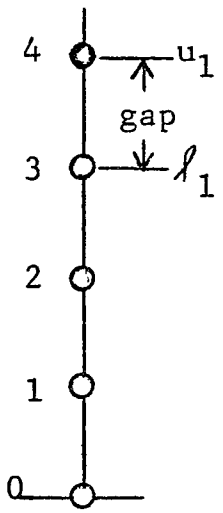


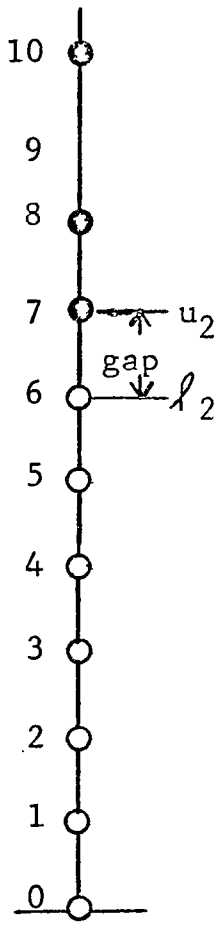
Figure 1.5. A five variable multigate realization of

$$F = X_5 [X_1(X_2\bar{X}_3 + X_3X_4) + \bar{X}_1 \bar{X}_2(X_3+\bar{X}_4) + \bar{X}_1X_2\bar{X}_3\bar{X}_5 .$$

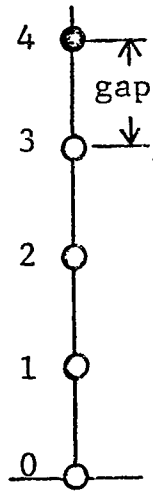
The corresponding line graph for each gate of Example 1.3 is shown in Figure 1.6a, b, c, and d. Each gate has its own line graph shown, and the gates correspond to those in the realization shown in Figure 1.5.



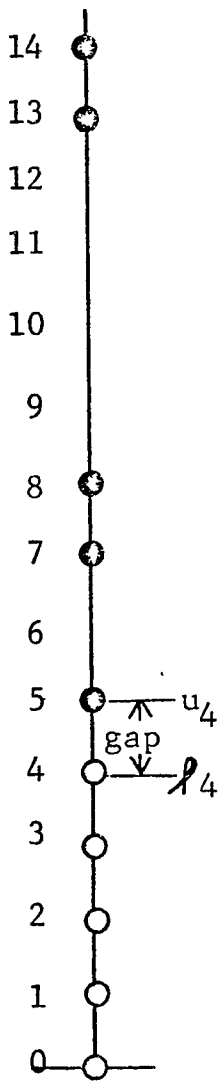
(a) Line Graph for A_1



(b) Line Graph for A_2



(c) Line Graph for A_3



(d) Line Graph for A_4

Figure 1.6. Line graphs for each gate in Example 1.3.

An example of a threshold gate circuit is that given by Coates and Lewis (3) and shown in Figure 1.7. This is a typical circuit actually used in a computer built using threshold gates. In this circuit the resistors $R_1, R_2, \ldots . R_n$ are inversely proportional to the weights. The threshold is set by R_T, V_T, and V_{BE} the emitter-base drop across the transistor.

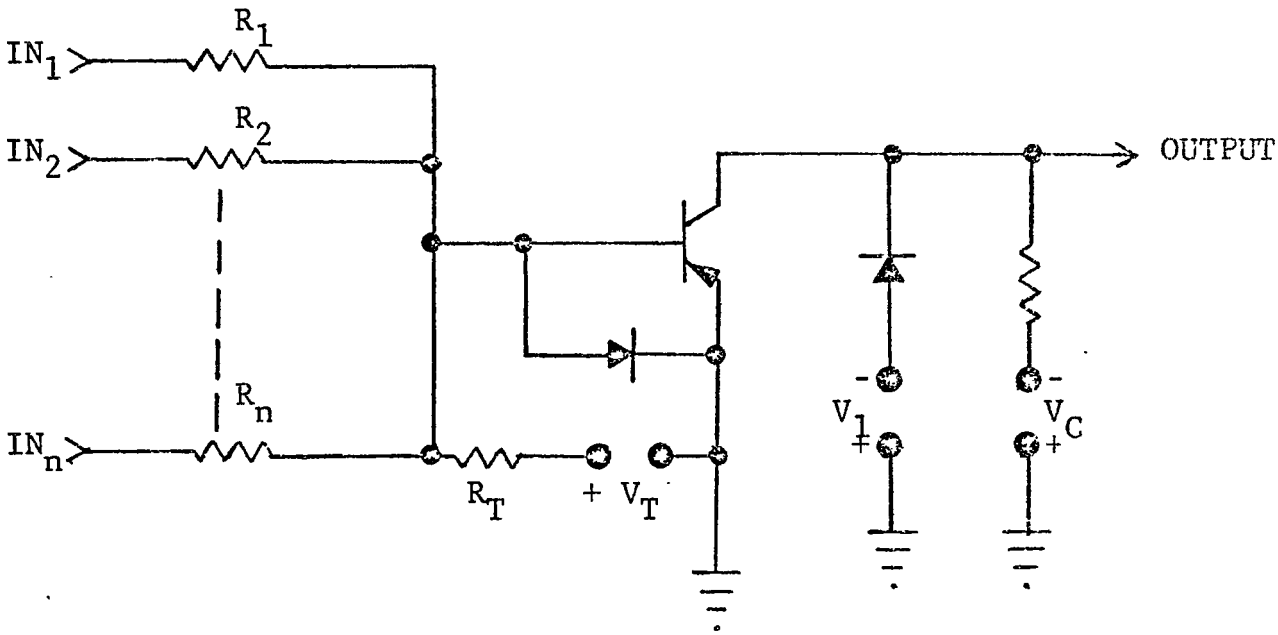


Figure 1.7. A typical example of a single threshold gate.

CHAPTER II

REDUNDANCY IN THRESHOLD LOGIC

In 1956 Von Newman (8) suggested two means of correcting errors in a logic network. These two methods were triplication and multiplexing.

Triplication is a technique by which the entire nonredundant network is triplicated, and the output of each of the triplicated networks is fed into a two-out-of-three majority gate. Figure 2.2 shows a triplicated network of the nonredundant network shown in Figure 2.1. The outputs of the networks are binary, therefore, two inputs to the majority gate will be the same at any time. The majority gate will have the same output as at least two of its inputs. Then the majority gate will have the correct output even if one of its inputs is in error. Any number of errors in the same network would be corrected by the majority gate because it would only affect one of its inputs. To correct t errors, one must duplicate the nonredundant realization 2t+1 times and use a (t+1)-out-of-(2t+1) majority gate.

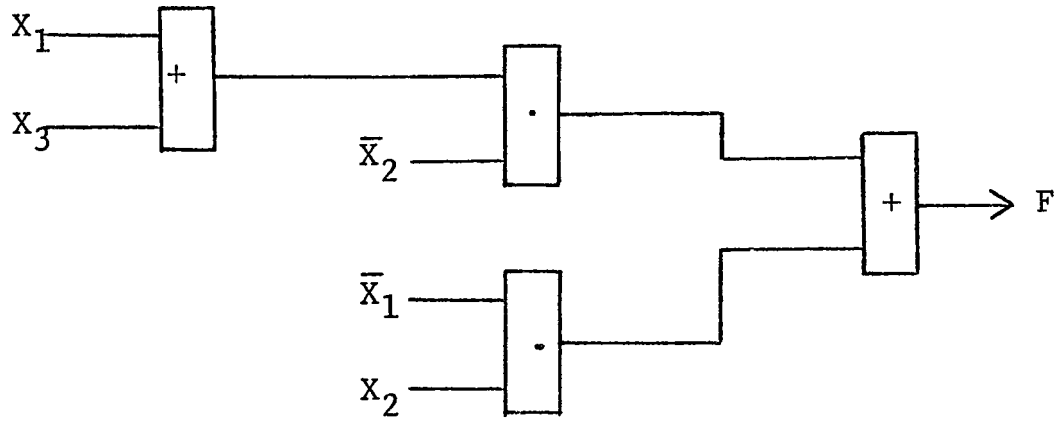


Figure 2.1. A nonredundant realization

$$F = \bar{X}_2(X_1+X_3) + \bar{X}_1X_2$$

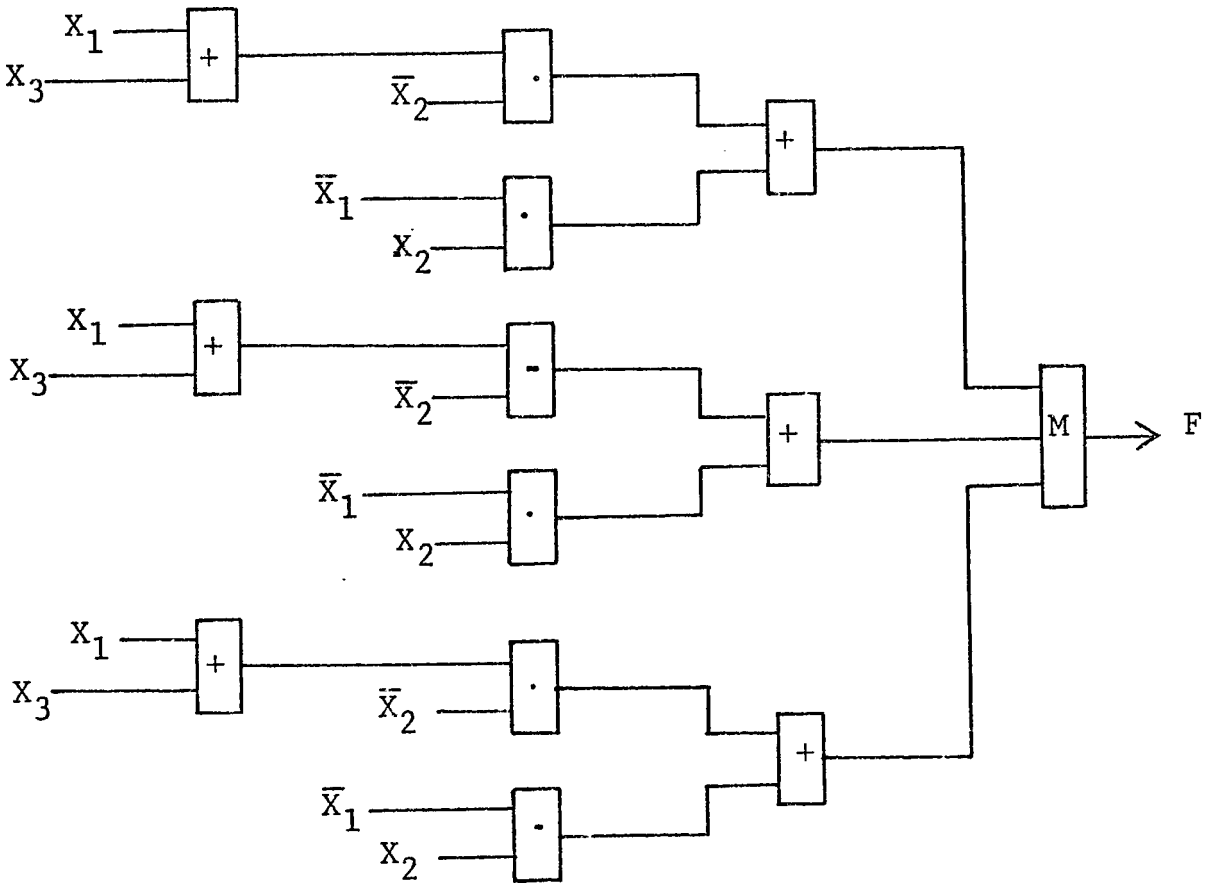


Figure 2.2. A triplicated version of the network in Figure 2.1.

A characteristic feature of multiplexing is that the error is corrected in the next level of logic following the occurrence of the error. In multiplexing, each gate is reproduced three times and the output of each of the three gates is an input to each of the three majority gates. The output of each of the majority gates is an input to one of the three gates on the next level of logic. An error is corrected by the majority gate immediately following the error. This technique requires more gates, but it will also correct an error in each set of three identical gates. The multiplexing technique is illustrated in Figure 2.3.

The first person to study the effects of errors on threshold gates was Muroga (7). He found that by replacing each X_i input with a bundle of k inputs, each identical to X_i, the gate could correct errors of these inputs.

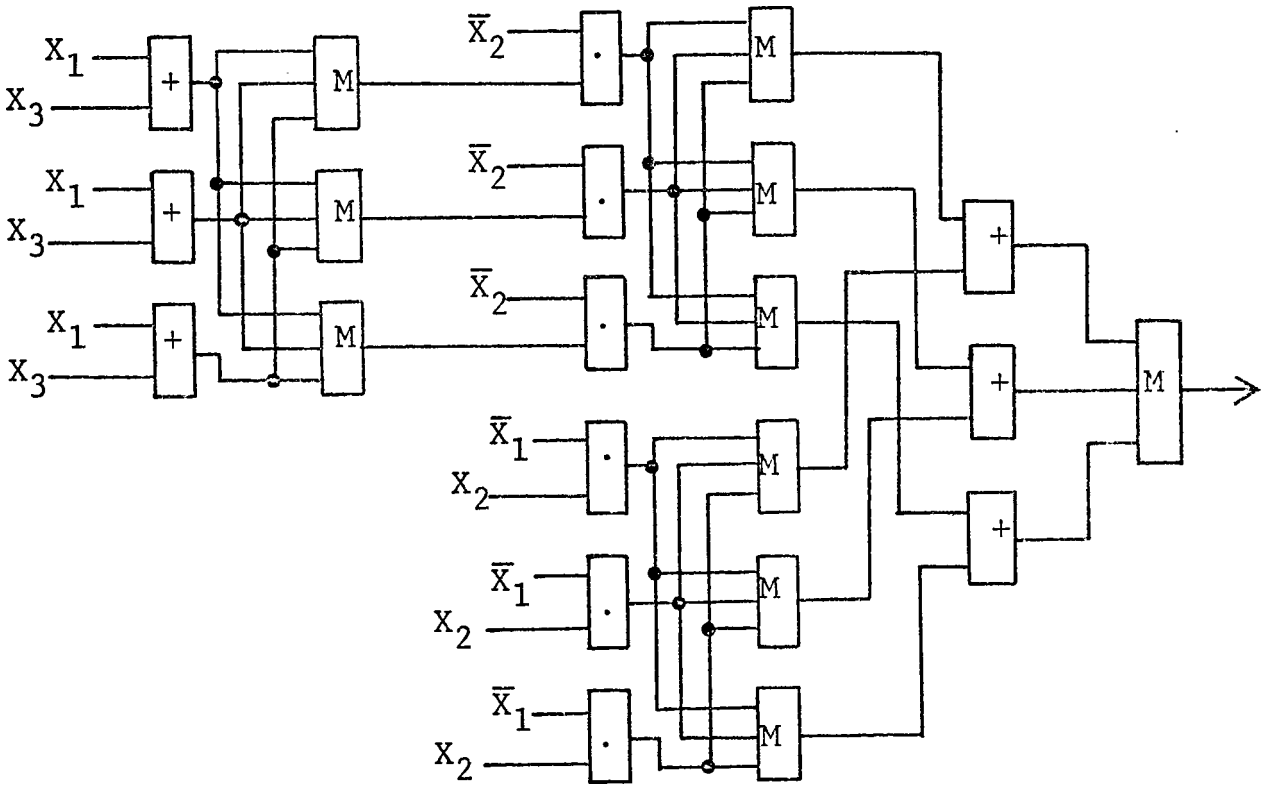


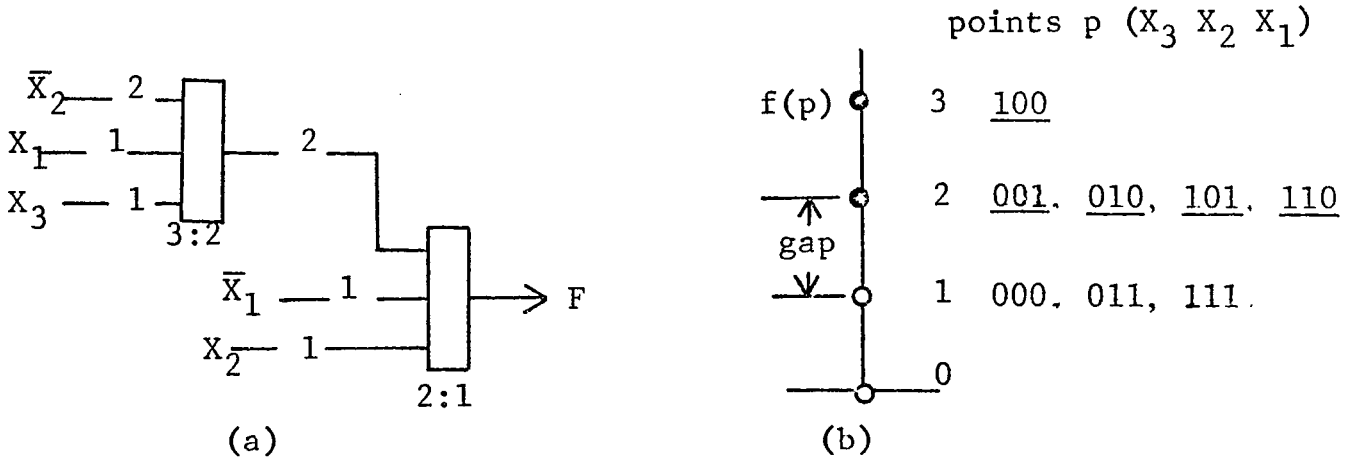
Figure 2.3. The multiplexed version of Network Figure 2.1.

Liu and Liu (6) took the approach that Muroga used and designed multiplexed realizations so that the logic gates of the network corrected the errors. If the outputs of Gates A_i (i=1,2, .n) are inputs to Gate A_j in a realization, Liu and Liu reproduce all A_i the same number of times k and connect the output of these gates to A_j. The value for k is found by taking the maximum of

all k_i, where k_i is obtained by a condition similar to Theorem 1 which is stated in Chapter III. This technique sufficiently solves the problem, but it may require more gates than are actually necessary.

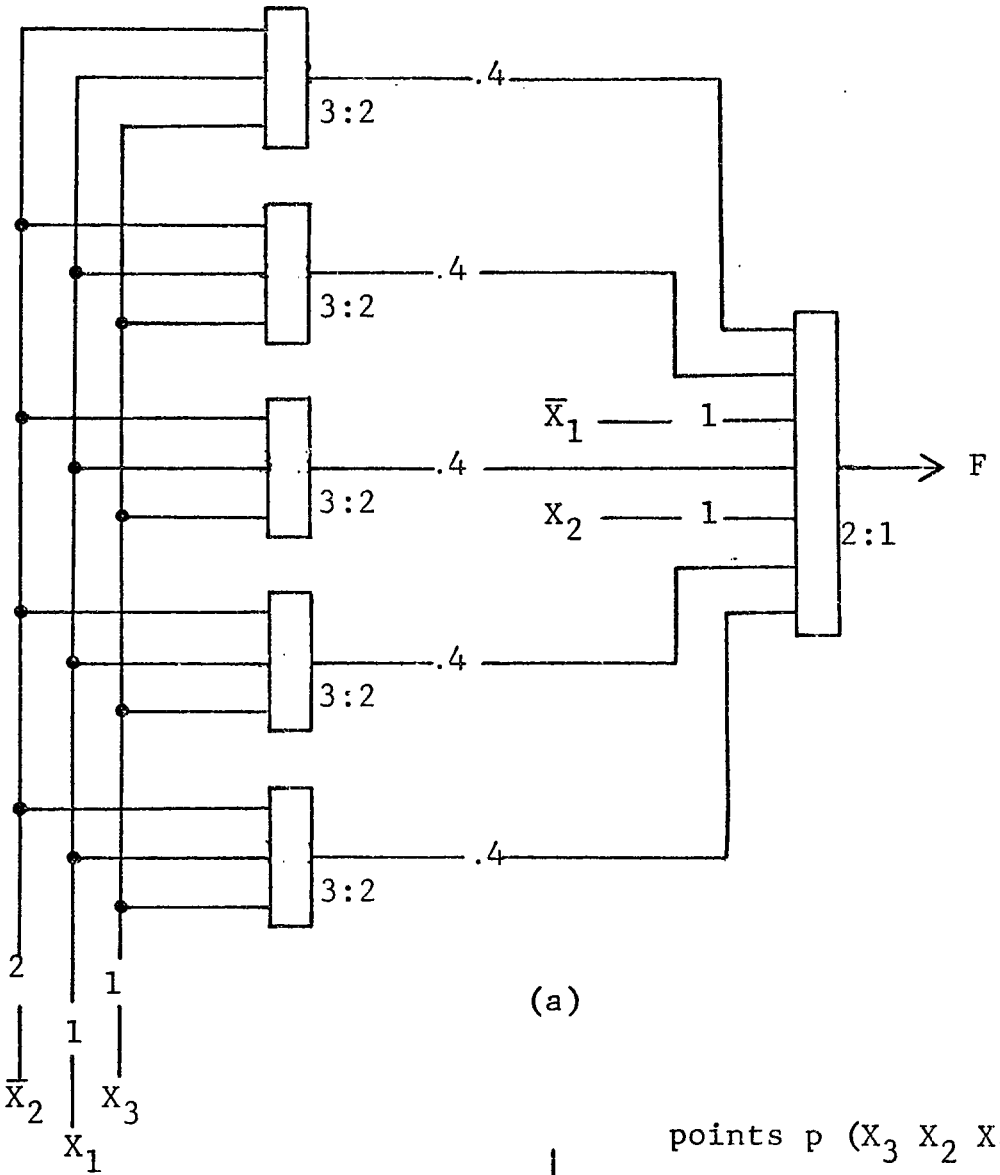
Three methods for designing error correcting capabilities into threshold gate networks were presented by Bargainer and Coates (1) in 1968. By these three methods the error correcting capabilities are designed into the logic gates themselves. The methods use multiplexing techniques, and the errors are corrected by the level of logic immediately following the occurrence of the error. The first method is quite similar to that used by Liu and Liu.

In the redundant networks of Figures 2.2 and 2.3, the function of the majority gate is to correct errors and the other gates perform the computation. A threshold gate can perform the function of both the computing gate and the restoring (error correcting) gate. Figure 2.4 represents a nonredundant threshold realization of Figure 2.1. Figure 2.5 represents a redundant realization of Figure 2.4.

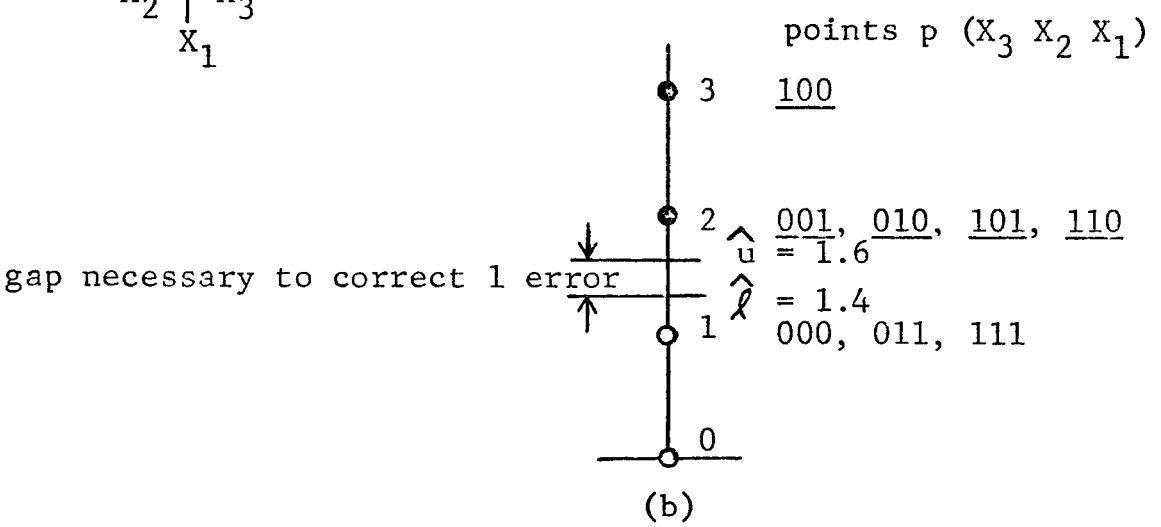


$$F(X_1,X_2,X_3) = \langle \bar{X}_1+X_2+2\langle X_1+X_3+2\bar{X}_2 \rangle_{3:2} \rangle_{2:1}$$

Figure 2.4. A realization with the corresponding map of $F(X_1,X_2,X_3) = \bar{X}_2(X_1+X_3) + \bar{X}_1X_2$



(a)



(b)

$$F(X_1,X_2,X_3) = \langle \bar{X}_1+X_2+.4 \langle 2\bar{X}_2+X_1+X_3 \rangle^{5}_{3:2} \rangle_{2:1}$$

Figure 2.5. A redundant realization with the corresponding map for $F(X_1,X_2,X_3) = \bar{X}_2(X_1+X_3) + \bar{X}_1X_2$.

Figure 2.5(a) is a redundant realization for the function shown in Figure 2.1. The superscript on the inner gate indicates the presence of 5 identical gates. Figure 2.5(b) is a map showing p_j and $f(p_j)$ for each $p_j \varepsilon \{0,1\}^n$. The points which are underlined are points for which the first level gate should have an output 1. Suppose one of the second level gates fails at 001. Then, f(001) would be decreased by .4 from 2 to 1.6. If an error occurs at 000, then f(000) increases from 1 to 1.4. However, if the threshold is between 1.4 and 1.6, then there is still no error of the output with an error in the second level of gates. Therefore, the output gate corrects one error of the second level gates if $1.6 \geq T > 1.4$. Then for the network in Figure 2.5 to correct an error of the second level gates, the gap of the output gate must be changed from 2:1 to 1.6:1.4.

The objective of this paper is to correct errors in a given network by using multiplexing techniques. When correcting errors in a network, it is assumed that the given realization does have the correct output function. Throughout the remainder of the paper some threshold realizations will be taken and modified by using multiplexing techniques to correct t errors, starting with a nonredundant realization R that consists of a set of gates $[A_i]$ (i=1,....n). This realization is modified to obtain a redundant realization $\hat{R}$ by using the following set of rules:

1) Each Gate A_i in R will be replaced by a set of k_i identical gates, and the set will be represented by $[\hat{A}_i]$. The output Gate A_0 in R will be replaced by a single gate $\hat{A}_0$ in $\hat{R}$.

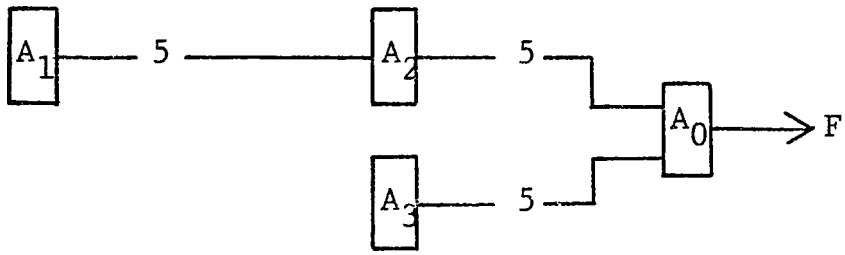
2) The independent variables, feeding into each gate of $[\hat{A}_i]$ in $\hat{R}$, and their associated weights will remain the same as those feeding into A_i of R. The Boolean function of each $\hat{A}_i$ in $\hat{R}$ will be the same as that realized by A_i in R.

3) If the output of A_i in R feeds into A_j with a weight of β_i, then the output of each $[\hat{A}_i]$ in $\hat{R}$ will have an input of $\hat{\beta}_i$ into each $\hat{A}_j$. (If the output of A_i feeds into more gates than A_j in R, then $\hat{\beta}_i$ will be set by the gate requiring the largest value for k_i. This is no real problem, however, because in the solution each A_j gate and its A_i input gates are considered one at a time. The largest value for k_i required by any of the A_j gates it feeds into is the value that is selected for the final solution.)

The realization $\hat{R}$ is called the multiplexed version of R. Each $\hat{A}_j$ will correct a given number of errors that are made in $\{[\hat{A}_i] | i \in J\}$, and J is the set of integers in R where the output of A_i is an input to A_j. It is noted that $\hat{A}_j$ will correct t errors of $\{[\hat{A}_i] | i \in J\}$ if and only if for t or fewer errors in $[\hat{A}_i]$ the output of $\hat{A}_j$ at $p \in \{0,1\}^n$ is the same as the output of A_j at $p \in \{0,1\}^n$ with no errors in A_i.

Errors at the output gate are not corrected, since errors are corrected in the level of logic immediately following the error. Any t gate errors, with the exception of the output gate, are corrected in the redundant realization. The inputs to the network are assumed to be correct. A nonredundant

realization R is shown in Figure 2.6(a) and its redundant realization $\hat{R}$ is pictured in 2.6(b). The independent variable inputs to the gates in Figure 2.6(a) and 2.6(b) have been omitted to avoid congestion of the figure. However, each separate gate in $[\hat{A}_i]$ in $\hat{R}$ has the same independent variable inputs as A_i of R.



$$F = \langle X_1+X_2+\bar{X}_3+2X_5+5 \langle 2\bar{X}_1+2\bar{X}_2+X_3+X_4+2X_5+5 \langle X_1+X_3+X_4+X_5 \rangle_{4:3} \rangle_{7:6}$$
$$+5 \langle \bar{X}_1+X_2+\bar{X}_3+\bar{X}_5 \rangle_{4:3} \rangle_{5:4}$$

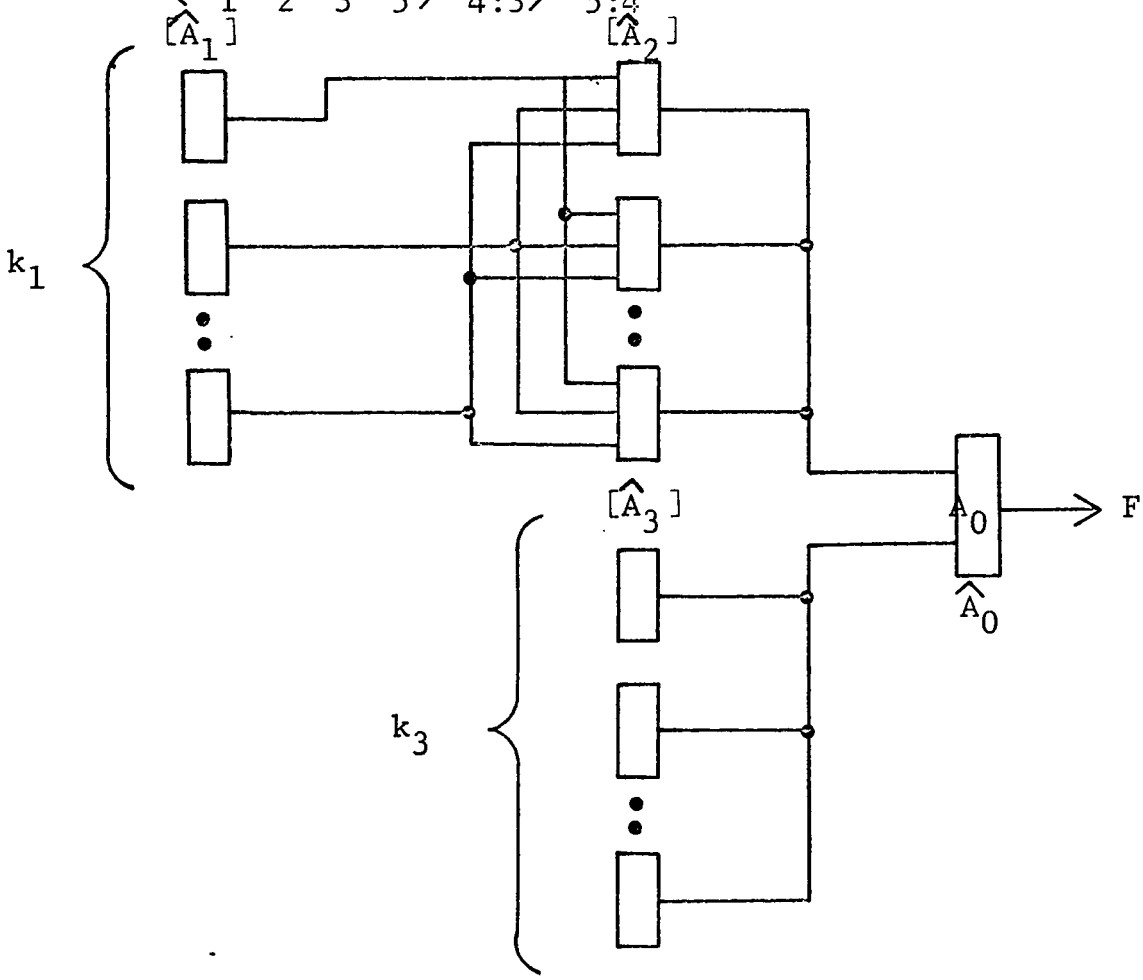


Figure 2.6(a) The gate interconnection to realize the function F.

Figure 2.6(b) The general multiplexed version of (a).

The notation used to relate $\hat{R}$ to R is defined as follows:

J	denotes the set of integers i in R, in which the output of A_i is an input to A_j;
k_i	denotes the number of gates in the set $[\hat{A}_i]$ in $\hat{R}$;
β_i	denotes the weight of the output of A_i as an input to A_j in R;
$\hat{F}_j$ (or F_j)	denotes the Boolean function realized by $\hat{A}_j$ in $\hat{R}$ (or A_j in R) and is defined on $\{0,1\}^n$;
$\hat{f}_j$ (or f_j)	denotes the separating function of $\hat{A}_j$ in $\hat{R}$ (or A_j in R) and is defined on $\{0,1\}^n$;
$i \epsilon J$	denotes that Gate A_i feeds into Gate A_j;
$u_j : \ell_j$	denotes the gap of A_j in R, and this sets the allowable region for the threshold where $\ell_j < T_j \leq u_j$;
$\hat{u}_j : \hat{\ell}_j$	denotes the gap of $\hat{A}_j$ in $\hat{R}$, and this sets the allowable region for the threshold where $\hat{\ell}_j < \hat{T}_j \leq \hat{u}_j$;
$\hat{g}_j$ (or g_j)	denotes the gap length where $\hat{g}_j = \hat{u}_j - \hat{\ell}_j$ (or $g_j = u_j - \ell_j$).

These definitions apply for the remainder of the paper regardless of the method being considered at any particular time.

The relation between R and $\hat{R}$ has been established. Given R one must determine $\hat{R}$. The values for k_i and $\hat{\beta}_i$ must be calculated in order to determine $\hat{R}$. It will be assumed that each

gate in R is necessary for the function F to be realized. If any gate is removed the function realized by R is changed.

CHAPTER III

METHODS FOR ADDING REDUNDANCY TO THRESHOLD NETWORKS

In Reference (1) three methods for determining $\hat{R}$, given R are presented. Method I is a simplified procedure which determines sufficient values for k_i and $\hat{\beta}_i$. This form of multiplexed realization usually requires more gates than are necessary, since k_i for $i \in J$ is calculated independently of all other k_i. Both Method I and Method II require that $\hat{\beta}_i = \beta_i / k_i$. This requires that $\hat{f}_j(p) = f_j(p)$ for all $p \in \{0,1\}^n$ in the absence of errors. It should be pointed out that these separating functions in R and $\hat{R}$ do not have to be equal for the corresponding gates to have the same output F(p). However, if the separating functions are the same in the absence of errors, the output would be the same.

Although Method I was not used as such in this paper, it might prove useful to consider it briefly. This approach will help clarify the objective of the other methods.

METHOD I

Method I is a procedure that determines sufficient values for all k_i and $\hat{\beta}_i$. The following notation is used throughout the remainder of this paper:

$\lfloor A \rfloor$ denotes the least integer greater than A.

Theorem 1: Any gate $\hat{A}_j$ in $\hat{R}$, a multiplexed version of R, will correct t or fewer errors in gates $\{[\hat{A}_i] \mid i \in J\}$ if

$$k_i = \left\lfloor \frac{2\beta_i t}{g_j} \right\rfloor \text{ for all } i \in J \text{ ;}$$

$$\hat{\beta}_i = \frac{\beta_i}{k_i} \quad \text{for all } i \in J \ ;$$

$$\hat{u}_j = u_j - \max \{\hat{\beta}_i t \mid i \in J\}$$

$$\hat{\ell}_j = \ell_j + \max \{\hat{\beta}_i t \mid i \in J\} \quad .$$

The procedure to follow, in order to produce a redundant realization $\hat{R}$ given a nonredundant realization R, is:

1) Each A_i of R should be replaced by a set $[\hat{A}_i]$ of identical gates, and each $\hat{A}_i$ should have identical independent inputs with the same corresponding weights as A_i.

2) If A_i feeds into A_j in R with a weight of β_i, then each $\hat{A}_i$ of $[\hat{A}_i]$ feeds into each $\hat{A}_j$ of $[\hat{A}_j]$ in $\hat{R}$ with weight $\hat{\beta}_i = \beta_i / k_i$, where $k_i = \frac{2\beta_i t}{g_j}$. If the output of some A_i feeds into all A_j such that $j \in M$, then let $k_i = \max \{ \left\lfloor \frac{2\beta_i t}{g_j} \right\rfloor \mid j \in M\}$ and let $\hat{\beta}_i{}^j = \frac{\beta_i{}^j}{k_i}$ where $\beta_i{}^j$ is the weight of the output of A_i into A_j. Each A_j has a gap $(u_j - \hat{\beta}_m t) : (\ell_j + \hat{\beta}_m t)$ where $\hat{\beta}_m = \max \{\hat{\beta}_i \mid i \in J\}$.

3) All Gates A_q in R that have no other gate output feeding into them will have a gap $\hat{u}_q : \hat{\ell}_q = u_q : \ell_q$.

4) The output Gate A_0 in R will be replaced by $\hat{A}_0$ in $\hat{R}$.

Liu and Liu (6) used a method quite similar to this except that they have the same number of gates in each $[\hat{A}_i]$ where $i \in J$. This number k^j is expressed as

$$k^j = \left\lfloor \frac{2t\beta_m}{g_j} \right\rfloor$$

where $\beta_m = \max \{\beta_i / i \epsilon J\}$. Theorem 1 indicates that if all gates whose outputs feed into A_j are reproduced the same number of times, more gates will probably be used than are actually needed.

METHOD II

Method I requires a small amount of computation, but usually requires more gates than are necessary because the value of each k_i for $i \epsilon J$ is calculated independently of all other gates whose outputs may also be inputs to A_j. Method II is concerned with multiplexed realizations that require a minimum number of gates for $\hat{R}$, given the realization R, and subject to the restriction that $\hat{\beta}_i = \beta_i / k_i$. As pointed out before, this restriction requires that each separating function $\hat{f}(p)$ in $\hat{R}$ be equal to the corresponding separating function in R.

Calculating the minimum number of gates in a multiplexed realization, subject to the above restriction, requires the simultaneous calculation of all k_i for which the output of A_i is an input to A_j and minimization of the sum of these k_i. Theorem 2 is used to accomplish this and the following notation is required:

w_i denotes the minimum value of $f_j(p)$ where $F_j(p) = F_i(p) = 1$ and the output of A_i is an input to A_j;

z_i denotes the maximum value of $f_j(p)$ where $F_j(p) = F_i(p) = 0$ and the output of A_i is an input to A_j.

In correcting errors in a realization that is known to realize the correct function, without errors, only two out of

four possibilities need be considered. If $F_i(p)$ feeds into Gate A_j along with other inputs, $F_i(p) = 1$ and $F_j(p) = 1$ is one possibility that is of interest, and $F_i(p) = 0$ and $F_j(p) = 0$ is the other. Making an error means that an output which should be 1 is 0 or vice versa. If $F_i(p)$ should equal 1 and it is 0, then the separating function of Gate A_j will be reduced from its proper value by β_{ij}. However, if $F_i(p) = 1$ and $F_j(p) = 0$, an error by Gate A_i would cause no problem because it would reduce $f(p)$ of A_j which is already below ℓ_j. In the same manner, if $F_i(p) = 0$ and $F_j(p) = 0$ and A_i made an error, then $f(p)$ of A_j is increased by β_{ij} which may cause an error in $F_j(p)$. However, if $F_i(p) = 0$ and $F_j(p) = 1$, an error caused by $F_i(p)$ going to 1 would only increase $f(p)$ of A_j by β_{ij}, and $f(p)$ is already $\geq u_2$.

When the output of A_i feeds into A_j, an error in the output $F_i(p)$ only results in an error in the output of A_j if the outputs should be $F_i(p) = 0$ and $F_j(p) = 0$, or $F_i(p) = 1$ and $F_j(p) = 1$. This assumes that the given realization does realize the correct function. These conditions will be the ones primarily dealt with for the remainder of the paper.

Theorem 2: In a realization R, consider some Gate A_j and the associated Gates A_i where $i \epsilon J$. Define

$$v_i = w_i - \frac{\beta_i t}{k_i}, \quad h_i = z_i + \frac{\beta_i t}{k_i}$$

$$v_o = u_j, \quad h_o = \ell_j.$$

Then in the multiplexed realization $\hat{R}$ of R where $\hat{\beta}_i = \beta_i / k_i$ for all $i \epsilon J$, $\hat{A}_j$ will correct t or fewer errors of $\{[\hat{A}_i] | i \epsilon J\}$ if and only if $\hat{T}_j$ has an allowable region as follows:

$$v_i \geq \hat{T}_j \text{ for all } i \epsilon \{J,0\},$$

$$h_q < \hat{T}_j \text{ for all } q \epsilon \{J,0\} .$$

The restriction that $v_o = u_j$ and $h_o = \ell_j$ simply requires that the gap for $\hat{A}_j$ in $\hat{R}$ fall within the gap of A_j in R. This restricts $\hat{u}_j \leq u_j$ and $\hat{\ell}_j \geq \ell_j$, meaning that the gap does fall within the original gap. Otherwise, it would be possible to find a gap, but it would not necessarily be within an acceptable range and the function realized would be changed.

When t errors occur in $\{[\hat{A}_i] i \epsilon J\}$, t_i would be the number of errors which occur in $[\hat{A}_i]$. For a given set of errors $e = \{t_i\}$, the minimum value for $\hat{f}_j(p)$ at any p such that $F_j(p) = 1$ which is designated $f_j(p)/e$ would be represented by

$$f_j(p)|e = \min\{(w_i - \frac{\beta_i t_i}{k_i}) | i \epsilon J\} = w_q - \frac{\beta_q t_q}{k_q}$$

for some $q \epsilon J$

But $t = \sum_{i \epsilon J} t_i$

Therefore

$$w_q - \frac{\beta_q t_q}{k_q} \geq w_q - \frac{\beta_q t}{k_q}$$

A similar situation arises when considering the maximum value for $f_j(p)$ at any p such that $F_j(p) = 0$ and a given set of errors $e = \{t_i\}$ occur. Then $f_j(p)$ is designated by $f_j(p)|e$, and would be represented by

$$f_j(p)|e = \max \{(z_i + \frac{\beta_i t_i}{k_i} | i\epsilon J\} = z_q + \frac{\beta_q t_q}{k_q}$$

for some $q\epsilon J$

But $t = \sum_{i\epsilon J} t_i$

Therefore

$$z_q + \frac{\beta_q t_q}{k_q} \leq z_q + \frac{\beta_q t}{k_q} \quad .$$

This implies that the minimum for $\hat{f}_j(p)$ at any p where $F_j(p) = 1$ will occur when all t errors occur in one $[\hat{A}_i]$ for $i\epsilon J$. Also the maximum for $\hat{f}_j(p)$ at any p where $F_j(p) = 0$ will occur when all t errors occur in one $[\hat{A}_i]$ for $i\epsilon J$.

Therefore this restricts $\hat{T}_j$ so that it can be any value within the following range:

$$\min \{v_i | i\epsilon J\} \geq \hat{T}_j > \max \{h_i | i\epsilon J\} ,$$

Then

$$\hat{u}_j : \hat{\ell}_j = \min \{v_i | i\epsilon J\} : \max \{h_i | i\epsilon J\} .$$

Now assume that no $\hat{T}_j$ exists because $v_i \leq h_q$ for some $i,q\epsilon J$. This implies that

$$f_j(p) \min \leq f_j(p) \max.$$

Then, $\hat{A}_j$ does not correct t errors of $\{[\hat{A}_i] | i\epsilon J\}$ for some p where $F_j(p) = 1$ or some p where $F_j(p) = 0$.

The problem is to solve for a minimum gate solution subject to the inequalities of Theorem 2. Therefore, for each A_i the set of inequalities

$$w_i - \frac{\beta_i t}{k_i} \geq \hat{T}_j$$

$$z_r + \frac{\beta_r t}{k_r} < \hat{T}_j$$

$$\hat{\ell}_j < \hat{T}_j \leq \hat{u}_j$$

must be solved simultaneously for integer values of k_i where $i \epsilon J$, and subject to minimizing the cost function

$$C = \sum_{i \epsilon J} k_i \ .$$

The inequalities are nonlinear due to the variable k_i appearing in the denominator. These inequalities may be solved using nonlinear programming with the additional constraint that k_i be an integer for all $i \epsilon J$.

Although the following information provided by the Corollary to Theorem 2 is not necessary to minimize the cost function by nonlinear techniques, it is useful for pointing out the difficulties in solving the problem analytically. For this reason it is included to help point out what the problem consists of, and what is involved in its solution.

Corollary 1: Necessary restrictions on the set $\{k_i \mid i \epsilon J\}$ that satisfy Theorem 2 are

$$k_i > \begin{cases} \dfrac{2\beta_i t}{w_i - z_i} \\ \dfrac{\beta_i t}{w_m - z_i} & \text{for all } m \epsilon J \\ \dfrac{\beta_i t}{w_m - z_m} & \text{for all } m \epsilon J \end{cases}$$

This sets the lower bound on k_i so that

$$k_i \geq \max\left\lfloor \left\{ \frac{2\beta_i t}{w_i - z_i}, \frac{\beta_i t}{w_m - z_i}, \frac{\beta_i t}{w_i - z_m} \right\} \right\rfloor .$$

Let this lower bound be denoted min k_i.

Theorem 3: A sufficient condition on the number of gates in $\{[\hat{A}_i] \mid i \in J\}$ is

$$k_i \leq \sum_{q \in J} \left\lfloor \frac{2\beta_q t}{g_j} \right\rfloor - \sum_{\substack{r \in J \\ r \neq i}} \min k_r .$$

Let this be the upper bound on k_i and denoted max k_i. This establishes a possible range for k_i which is min $k_i \leq k_i \leq$ max k_i.

METHOD III

Both Method I and Method II required that $\hat{f}_j(p) = f_j(p)$ for all $p \in \{0,1\}^n$ when no errors are made. This results in the requirement that $\hat{\beta}_i = \beta_i / k_i$ for all i. Both methods yielded realizations under this restriction. However, Method II did so with a minimum number of gates. In Method III the restriction that $\hat{\beta}_i = \beta_i / k_i$ is removed and the multiplexed realization R requiring a minimum number of gates will be sought.

In Method III the following will be defined:

$$v_i(p) = \sum_{y \in \alpha_j} a_y X_y(p) + \sum_{w \in J} k_w \hat{\beta}_w F_w(p) - \hat{\beta}_i t \, F_i(p)$$

where $F_i(p) = 1$,

$$h_i(p) = \sum_{y \in \alpha_j} a_y X_y(p) + \sum_{w \in J} k_w \hat{\beta}_w F_w(p) + \hat{\beta}_i t \, \overline{F}_i(p)$$

where $F_i(p) = 0$,

and

$\alpha_j = \{i \mid$ the independent input X_i is an input to A_j in $R\}$.

$F_i(p)$ is the value of the Boolean function realized by A_i at $p \in \{0,1\}^n$, and

$X_y(p)$ is the value of the dependent input X_y at $p \in \{0,1\}^n$.

Then

$$v_i(p) = \min \{\hat{f}_j(p)\Big|_t^i, f_j(p)\} \text{ for } p \text{ where } F(p) = 1,$$

and $\hat{f}_j(p)\Big|_t^i$ denotes $f_j(p)$ when t errors occur in $[\hat{A}_i]$.

Likewise

$$h_i(p) = \max \{\hat{f}_j(p)\Big|_t^i, f_j(p)\} \text{ for } p \text{ where } F(p) = 0.$$

Theorem 4: If A_j is some gate in a realization R, then in the multiplexed realization $\hat{R}$, $\hat{A}_j$ will correct t errors if and only if there exists $\hat{T}_j$ such that

$$v_i(p) \geq \hat{T}_j \text{ for all } i \in J \text{ where } F_j(p) = 1,$$

$$h_i(p) < \hat{T}_j \text{ for all } i \in J \text{ where } F_j(p) = 0.$$

The minimum value for $\hat{f}_j(p)$ at any p where $F_j(p) = 1$ will occur when all t errors are made in one $[\hat{A}_i]$ for $i \in J$. Also the maximum value for $\hat{f}_j(p)$ at any p where $F_j(p) = 0$ will occur when all t errors are made in one $[\hat{A}_i]$ for $i \in J$. This restricts T_j so that it can be any value within the following range:

$$\min \{v_i(p) \mid i \in J\} \geq \hat{T}_j > \max \{h_i(p) \mid i \in J\} \quad .$$

Since the k's and $\hat{\beta}$'s are both variables, these inequalities are nonlinear due to the product term $\hat{\beta}_i k_i$. The purpose is to find the minimum number of gates necessary to correct t errors. The values for the $\hat{k}_i$'s and the $\hat{\beta}_i$'s can be obtained by nonlinear programming by making the added restriction that the k values be integers for all $i \epsilon J$. The objective is to calculate the k_i's and the $\hat{\beta}_i$'s necessary to correct t errors while minimizing the cost function

$$C = \sum_{i \epsilon J} k_i \ .$$

CHAPTER IV

PROCEDURE

As indicated in Chapter III, the problem is to solve the following inequalities simultaneously for each A_i using integer values of k_i where $i \epsilon J$,

$$w_i - \frac{\beta_i t}{k_i} \geq \hat{T}_j$$

$$z_r + \frac{\beta_r t}{k_r} < \hat{T}_j$$

$$\hat{\ell}_j < \hat{T}_j \leq \hat{u}_j$$

and subject to minimizing the cost function

$$C = \sum_{i \epsilon J} k_i \ .$$

To solve the problem analytically would require a great deal of calculation, and it would still involve a trial and error solution. A procedure which might be followed is shown below.

1) Select some A_j so that $A_j \epsilon R$ and the outputs of other gates in R are inputs to A_j. Calculate w_i and z_i for each $i \epsilon J$ where

$$w_i = \min \{f_j(p)/F_i(p) = F_j(p) = 1\}$$

$$z_i = \max \{f_j(p)/F_i(p) = F_j(p) = 0\}$$

2) Calculate min k_i for each $i \epsilon J$ where

$$\min k_i = \max \left\{\frac{2\beta_i t}{w_i - z_i}, \frac{\beta_i t}{w_m - z_i}, \frac{\beta_i t}{w_i - z_m}\right\} \ .$$

3) Determine max k_i by

$$\max k_i = \sum_{q \in J} \frac{2\beta_q t}{g_j} - \sum_{\substack{r \in J \\ r \neq i}} \min k_r .$$

4) Solve for the minimum value of k_i by trial and error for all $i \in J$ such that the total number of gates is minimized, and all the constraints are satisfied.

5) Take another A_j in R and repeat steps 1) through 4).

6) Repeat step 5) until all A_j in R have been used.

7) Take the maximum value of k_i required by any A_j and this is the required answer.

8) Interconnect the gates as in Figure 2.6(b) with $\hat{\beta}_i = \beta_i / k_i$.

A Method II analytical solution involves a great deal of calculation at best. To calculate w_i and z_i in step 1 requires the evaluation of $f_j(p)$ for each gate A_i where $i \in J$, at every point $p \in \{0,1\}^n$ where n represents the number of independent variables. For a five variable function there are $2^5 = 32$ points at which the separating function for each gate must be evaluated. After w_i and z_i are evaluated for all $i \in J$, the min k_i and max k_i can be evaluated in step 2 and 3.

When w_i, z_i, min k_i, and max k_i for $i \in J$ have been evaluated, one must search through the possible values of k_i for a set of k_i that will satisfy the constraints. One must continue to try all possible combinations of k_i values to find the values

of k_i that will minimize the cost function and satisfy the constraints as well. This process must be followed for all Gates A_j.

Since the procedure that must be followed to find a solution by Method II is quite lengthy, a computer program was written to solve the problem using a search technique. The problem is inherently a nonlinear programming problem, as was pointed out in Chapter III. Bargainer and Coates (1) set the problem up to be solved by integer linear programming techniques using existing algorithms. This technique required that the min k_i and the max k_i be calculated for all $i \in J$, and a table of all possible k_i values be constructed. As a result this requires working with a large number of variables. For this reason, it was felt that a nonlinear search technique, which would minimize some cost function, might provide a faster and more direct approach to the solution of the problem. The program written does not consider the possible range of k values, but solves directly for the k_i values that will minimize the cost function and satisfy the constraints

The Search Technique

A direct search technique based on "Optimal Search" developed by Weisman, Wood, and Rivlin (9) was chosen. The optimal search technique is actually an extension of the "Pattern Search" procedure developed by Hookes and Jeeves (4). The extension consists of the addition of constraints on the variables of the Pattern Search. This technique was chosen because:

1) Hookes and Jeeves found it to be quite effective in locating minima in "steep valleys", and
2) it appeared to be the most straight-forward method of handling the constraints that had to be satisfied.

The pattern search technique is based upon the principle that any successful parameter adjustment is worth trying again. The object is to minimize a given cost function. The value of the function after a parameter adjustment is compared to the value before the adjustment, and if the function has decreased, the move is considered to be a success and the incremented parameter is retained.

There are two basic kinds of moves in the Pattern Search Technique; the Exploratory and the Pattern. The exploratory move is used to obtain some knowledge about the function to be minimized without any consideration of the gradient of the function. The exploratory search starts from some initial base point and upon completion establishes a new base point. The pattern move starts from the base point established by the exploratory move and uses the information obtained by the exploratory move in an attempt to establish still another base point.

The exploratory move increments each parameter one at a time and attempts to decrease the cost function. The first parameter is increased by some predetermined amount, and if successful, the incremented value is retained. If the increased parameter adjustment is unsuccessful, then the parameter is decreased by the same predetermined amount, and if successful, the decreased value is retained. If it is unsuccessful, the parameter is restored to its original value. After each parameter is incremented in this manner, one at a time, a new base point will have been reached.

The pattern move starts from the new base point established by the exploratory move and increments every parameter at

the same time by an amount equal to the difference between the new base point and the previous base point. If the pattern move is unsuccessful, the parameters are all restored to their previous values. If the pattern move is successful, a new base point is established. At this point, another exploratory move is performed. However, it seems reasonable to assume that if one pattern move is successful, another by an equal amount might be successful also. Following this assumption, the search technique used will continue to make a pattern move until the value of the cost function increases, and the previous base point will then be restored.

In this procedure, any time an exploratory move is attempted and the cost function cannot be reduced by incrementing any parameter either positive or negative, the predetermined step increment is reduced and another exploratory search is attempted. When the step increment falls below some predetermined minimum the search is terminated. By setting the predetermined minimum to the proper value, the correct value may be found to the desired accuracy.

In order to illustrate the search technique just described, an example of a two-dimensional pattern move is given in Figure 4.1. In the example, C_1, C_2, C_3, C_4, and C_5 are equal value contours for the cost function with $C_1 > C_2 > C_3 > C_4 > C_5$.

The search begins at some initial point B_0. First k_1 is incremented positive by some amount δ to $k_1+\delta$. If the cost function is decreased, this value of k_1 is retained. If there is no decrease, then $k_1-\delta$ is tried. If the cost function is reduced, the vector V_1 is established. k_2 is then incremented by the same amount δ to $k_2+\delta$. If the value of the cost function is reduced, a new base point B_1 is established.

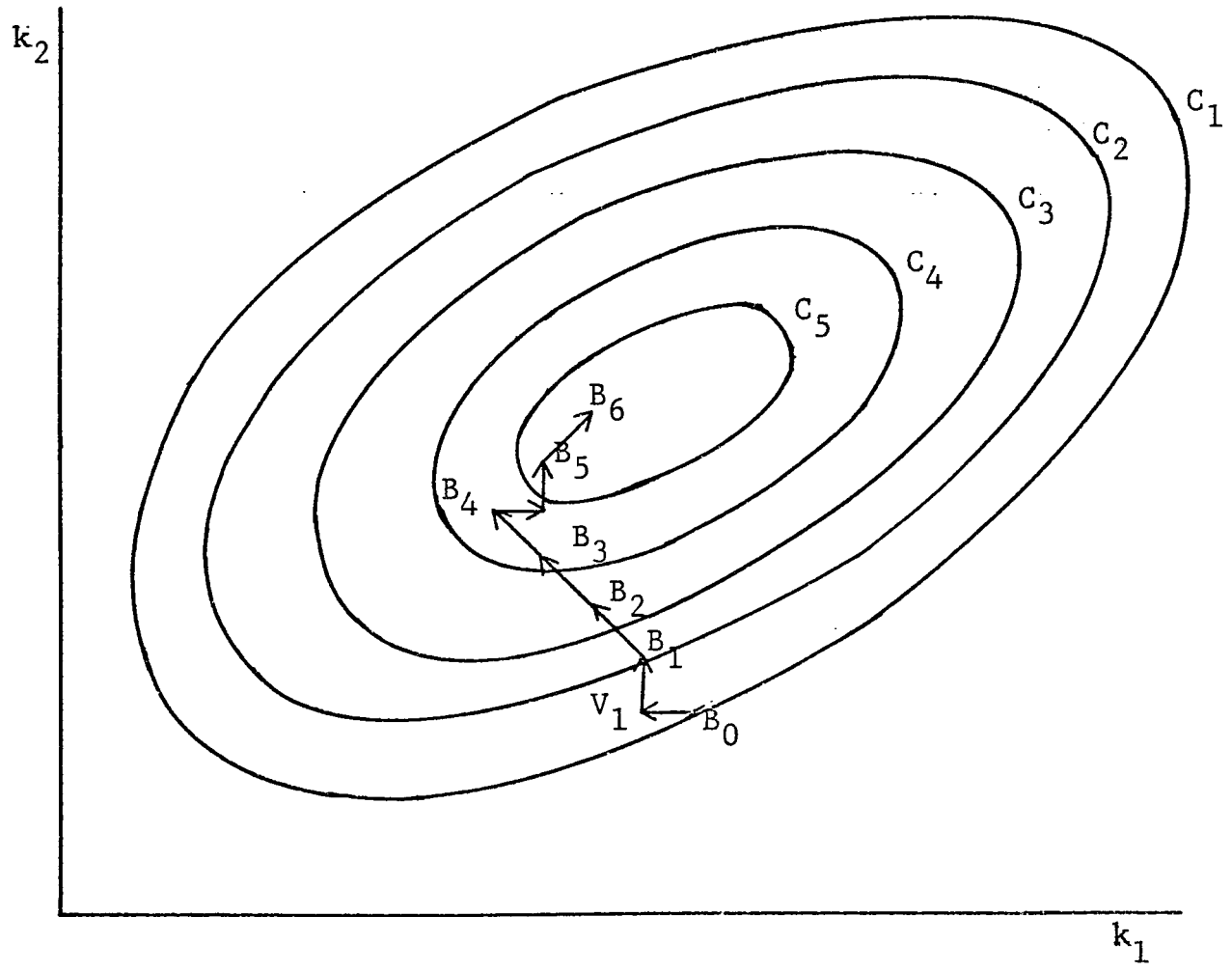


Figure 4.1. A two-dimensional pattern search.

From the new base point B_1 a pattern move is attempted. Both k_1 and k_2 are incremented by the same amount necessary to move them from B_0 to B_1. If the cost function is reduced, a new base point B_2 is established. This process is continued until the cost function increases, then the previous base point is restored and an exploratory search is attempted again.

This same procedure is followed until no attempt to increment k_1 or k_2 either positive or negative, by an exploratory search, is successful. In the procedure of Hookes and Jeeves (4),

the increment δ is reduced and the entire process repeated. The increment δ is reduced and another attempt made to decrease the cost function until the increment δ drops below some predetermined value. The search is terminated at this point.

In the actual program the k_1 and k_2 would represent the number of times Gates A_1 and A_2 would have to be duplicated respectively in order to add a specific redundancy to the network. Therefore, the k values must be integer values. For a network requiring a reasonable number of gates, it did not seem practical to allow k to be incremented by more than one at a time. Each k may remain constant, increment positive by one, or increment negative by one in any given step. The value one is the only allowable step increment.

Weisman, Wood, and Rivilin (9) used the idea of a penalty factor in their optimal search technique to insure that constraints were satisfied on minimization problems. The problem they solved is very similar to this one. They wanted to minimize some function subject to certain constraints. They took some value for the variables and checked to see if the constraints were satisfied and, if not, they increased some error function which is added to the function to be minimized. Therefore, by making the penalty factors large enough one can insure that the constraints are satisfied or the function can not be minimized.

The objective of Method II is to solve simultaneously for integer values of k_i which will minimize the cost function

$$C = \sum_{i \epsilon J} k_i$$

subject to the following constraints:

$$v_i > \hat{T}_j \qquad \text{where } v_i = w_i - \frac{\beta_i t}{k_i},\ h_i = z_i + \frac{\beta_i t}{k_i}$$

$$h_i < \hat{T}_j$$

$$\hat{\ell}_j < \hat{T}_j \leq \hat{u}_j \qquad v_o = u_j,\ h_o = \ell_j \ .$$

Therefore, an error is made if any $v_i \leq h_q$ where $i,q \epsilon J$. An error results when there is no required gap for the threshold T. An error is also made if $v_i \leq \ell_j$ or $h_i \geq u_j$ for $i \epsilon J$. This error occurs because the gap found is not within the original gap. By subtracting $h_q - v_i$, it is possible to get an indication of how far the constraints are from being satisfied or how large the error is. Likewise, by subtracting $\ell_j - v_i$ and $h_i - u_j$, it is possible to determine how far the gap is from the original gap.

Penalty factors are added to the cost function for the constraints not being satisfied. The cost function then appears as below:

$$C = \sum_{i \epsilon J} k_i + C_1 \sum_{i \epsilon J} \sum_{q \epsilon J} (h_q - v_i) + C_2 \sum_{i \epsilon J} [(h_i - u_j) + (\ell_j - v_i)]$$

where C_1 and C_2 are positive constants which weight the penalty factors. Only a positive error or zero can be added. If $h_q - v_i$, $h_i - u_j$, or $\ell_j - v_i$ is negative, zero will be summed into the cost function for that value. Otherwise, the error function would add in negative error if a constraint were satisfied and this would cancel the effect of an error elsewhere. By making C_1 and C_2

large enough, C can only be minimized if the constraints are satisfied.

The cost function may be expressed in the following way:

$$C = \sum_{i \epsilon J} k_i + EF$$

where EF represents the error factor

$$EF = C_1 \sum_{i \epsilon J} \sum_{q \epsilon J} (h_q - v_i) + C_2 \sum_{i \epsilon J} (h_i - u_j) + (\ell_j - v_i).$$

After some trials to minimize the cost function, the best approach seemed to result when C_2 was made a large constant value requiring that the final gap $\hat{g}$ approach the original gap g before the other factors became significant. Certainly the $\sum_{i \epsilon J} k_i$ would have to be small compared to the error due to any $v_i < \ell_j$ or $h_i > u_j$. This technique caused the routine to approach the correct answer very quickly.

A solution is reached with a given initial value for C_1. If EF is not zero for this solution, C_1 is increased and another solution found. This procedure is continued until C is minimized and EF is zero, meaning all the constraints are satisfied. Once the k values are large enough for the constraints to be satisfied EF = 0, and the k values can be minimized. In effect, this means that the error due to any constraint not being satisfied multiplied by its constant multipling factor (C_1 or C_2) must be greater than one. If one more gate is added to satisfy the constraint then the reduction in the cost function must be greater than one or the cost function will not be reduced.

Solution by Method II

In order to find a Method II solution using nonlinear computing techniques, a certain procedure must be followed. The following procedure represents a brief outline of the steps the program must use.

Method II Procedure:

1) Select some J so that $A_j \epsilon R$ and the outputs of other gates in R are inputs to A_j. Calculate w_i and z_i for each $i \epsilon J$ where

$$w_i = \min \{f_j(p) | F_i(p) = F_j(p) = 1\}$$

$$z_i = \max \{f_j(p) | F_i(p) = F_j(p) = 0\} .$$

2) Solve by the search technique for the minimum value of k_i for all $i \epsilon J$ which will minimize the cost function.

$$C = \sum_{i \epsilon J} k_i + EF$$

and satisfy all the constraints, so that EF = 0.

3) Take another A_j and repeat steps 1) and 2).

4) Repeat step 3) until all A_j in R have been used.

5) Take the maximum value of A_i required by any A_j and this is the correct answer for A_i.

6) Interconnect the gates as in Figure 2.6(a) with $\hat{\beta}_i = \beta_i / k_i$.

The program follows the above procedure and solves for the number of gates necessary to add a required redundancy to a given network. The MAIN PROGRAM reads the information necessary to define the original realization R. This information includes

the number of gates, the number of variables, the number of errors, the initial k values, the weights for all the independent variables (A_{vg}) and their complements feeding into all gates, the weights (β_{ij}) for each gate feeding all other gates, and the upper and lower limit ($u_j : \ell_j$) for the gap of each gate. The MAIN PROGRAM takes the first gate and assigns it as A_j, and then calls subroutine CONST.

The subroutine CONST takes the separating function and evaluates it at each point $p \in \{0,1\}^n$. Therefore, it solves for w_i and z_i for all $i \in J$. Then subroutine CONST returns to the MAIN PROGRAM which immediately calls subroutine SEARCH.

Subroutine SEARCH follows the procedure described in the Search Technique section of this chapter. It solves for the minimum cost function

$$C = \sum_{i \in J} k_i + EF$$

so that all the constraints are satisfied, meaning EF = 0. The subroutine takes the initial k values which were read in and calculates the cost function using these values. It starts the search procedure described by beginning an exploratory search. After indexing the first k value positive, it again calculates the cost function to see if it has been reduced. It continues the search technique, checking the cost function after every move until the cost function is minimized and the constraints satisfied. Then, it returns to the MAIN PROGRAM.

The MAIN PROGRAM takes all gates A_j one at a time and repeats the steps of calling subroutines CONST and SEARCH until each gate A_j has been used. Each time control is returned to the

MAIN PROGRAM it checks the k_i values required for the A_j gate just finished and stores the maximum k_i value required up to that time. After all gates A_j have been used, the program prints out all k_i values, β values, u values, and values which is all the information necessary to construct $\hat{R}$.

Solution by Method III

Method III solves for the minimum multiplexed realization R necessary to correct t errors of the given realization R. The restriction that $\hat{\beta}_i = \beta_i/k_i$ is removed, where β_i is the original Beta that was read in. This means that $\hat{\beta}_i$ is free to vary and is not restricted by the β_i of the original function. Therefore, the $\hat{\beta}_i$ values may be searched to find those $\hat{\beta}_i$'s requiring the smallest total number of k_i's necessary to correct t errors. Since $\hat{\beta}_i$ is the weight of each gate of the set of gates $\{\hat{A}_i\}$ feeding into each $\hat{A}_j$ and k_i is the number of gates in the set $\{\hat{A}_i\}$, the effective weight of all the gates in the set $\{\hat{A}_i\}$ is still $\beta_i = k_i\hat{\beta}_i$ at any given time, but β_i may be different from the original β_i read in. Thus, it is possible to require that $\hat{\beta}_i = \beta_i/k_i$ and perform a search on the β_i values, and receive the same results as would be received if the $\hat{\beta}_i$ values were searched.

For the reasons given above, a search is performed on the β_i values, and a Method II solution found using the new β_i values specified. The minimum cost function

$$C = \sum_{i \epsilon J} k_i$$

is found so the original function is realized and t errors are corrected.

Method III Procedure:

1) Select some A_j so that $A_j \epsilon R$ and the outputs of other gates in R are inputs to A_j.

2) Calculate and store the initial output function values $F_j(p)$ for Gate A_j for each point on the n cube.

3) Calculate w_i and z_i for each $i \epsilon J$ where

$w_i = \min \{f_j(p) | F_i(p) = F_j(p) = 1\}$

$z_i = \max \{f_j(p) | F_i(p) = F_j(p) = 0\}$

4) Perform a search on the k_i values according to the search technique described.

5) After the cost function

$$C = \sum_{i \epsilon J} k_i + EF$$

has been minimized, check to see if EF = 0. If the error function is not equal to 0, increase the constraints error multiplying factor and return to step 3). If the error function is equal 0 go on to step 6).

6) Index θ_i value according to the search technique.

7) Calculate w_i and z_i for each $i \epsilon J$ where

$w_i = \min \{f_j(p) | F_i(p) = F_j(p) = 1\}$

$z_i = \max \{f_j(p) | F_i(p) = F_j(p) = 0\}$

8) Knowing the value of the output function $F_j(p)$ for Gate A_j for each point on the n cube, calculate the minimum value of $f_j(p)$ such that $F_j(p) = 1$ and the maximum value of $f_j(p)$ such that $F_j(p) = 0$ without any errors being made. These two values are the upper and lower limits for the new gap of A_j and are represented by UNE(J) and LNE(J) respectively. If UNE(J) > LNE(J) go on to step 9), and if not return to step 6).

9) Perform a search on the k_i values according to the search technique described.

10) After the cost function

$$C = \sum_{i \epsilon J} k_i + EF$$

has been minimized, check to see if EF = 0. If the error function is not equal to 0, increase the constraints error multiplying factor and return to step 7). If the error function is equal to 0, go on to step 11).

11) If the step increment for the β_i values is below the predetermined minimum, go on to step 12), if not, return to step 6).

12) Take another A_j and repeat steps 2) through 11).

13) Repeat step 12) until all A_j have been used.

14) Take the maximum value required of k_i by any A_j for a minimum cost function solution with the constraints satisfied, and this is the correct k_i value.

15) Interconnect the gates as in Figure 2.6(b).

The Method III program follows the above procedure and solves for the minimum number of gates necessary to add a required redundancy to a given network. The MAIN PROGRAM reads in the information necessary to define the original realization R. This information includes the number of gates, the number of variables, the number of errors, the initial k values, the weights for all the independent variables (A_{vg}) and their complements feeding into all gates, the weights (β_{ij}) for each gate feeding into all

other gates, and the upper and lower limit $(u_j : \ell_j)$ for the gap of each gate. The MAIN PROGRAM takes the first gate and assigns it as A_j, and proceeds to call subroutine OUTFUN.

The subroutine OUTFUN calculates the output function $F_j(p)$ of Gate A_j for each point on the n cube. It stores this information for a reference value to be used later. Then control is returned to the MAIN PROGRAM.

The MAIN PROGRAM calls CONTRL. The subroutine CONTRL sets initial values of the variables and calls subroutine CONST. Subroutine CONST calculates w_i and z_i and then calls subroutine SEARCH. Subroutine SEARCH follows the procedure described in the search technique section until the minimum for the cost function is found. Then the program returns to subroutine CONTRL. If the cost function due to errors is equal to zero, the program returns to the MAIN PROGRAM, if not, the error function is increased and subroutine CONST is called again. The same procedure is followed until the cost function is minimized and the part due to errors is zero, and the program returns to the MAIN PROGRAM.

The MAIN PROGRAM stores the information concerning β_i, k_i, $\hat{u}_j$ and $\hat{\ell}_j$. Then subroutine BETA is called and the search for the best β_i values begins. The first β_i is incremented, and the program returns to the MAIN PROGRAM. The MAIN PROGRAM immediately calls subroutine CONTRL. After setting initial values, subroutine CONST is called. The calculations for w_i and z_i are made. Knowing what the output function $F_j(p)$ of Gate A_j should be for each point p on the n cube, it is possible to calculate the upper and lower limits for the new gap which are UNE(J) and LNE(J) respectively.

If UNE(J) > LNE(J), then there is a gap and subroutine SEARCH is called and the cost function minimized. If UNE(J) ≤ LNE(J), then there is no new gap and the program returns to CONTRL which immediately returns to the MAIN PROGRAM. In this situation the program could not realize the proper function without any error being made. This is an unsatisfactory answer, so subroutine BETA is called and the indexing of β_i values continues.

When UNE(J) > LNE(J), subroutine SEARCH is called, and the cost function is minimized. Then the program returns to CONTRL which checks to see if the cost function, due to constraints not being satisfied (CSAT), is zero. If CSAT ≠ 0, the error multiplying factor is increased and CONST is called again. The procedure is repeated. When the cost function is minimized and CSAT = 0, the program returns to the MAIN PROGRAM. The answers are checked to see if the sum of the k values is less than the previously stored answer. If the sum of the k values is reduced, the stored answer is replaced with the better answer and the indexing of the β_i values continues. If the sum of the k values is greater than the stored value, the stored values are not changed and subroutine BETA is called. If the sum of k values is exactly equal to the previously stored sum, the gap is checked to see if it is wider than the previously stored gap. If the gap is wider the new values replace the previously stored values and BETA is called. If the gap is narrower, then BETA is called immediately and the incrementing of the β_i values continues.

This procedure continues until an exploratory move of all β_i values is unable to improve the stored results. The

increment by which the β_i values are changed in each step is then reduced, and another attempt is made at reaching a better solution. Each time an exploratory move of all β_i values fails to improve the results, the increment by which the β_i values are changed is reduced until the increment falls below some predetermined value. When the step increment falls below its minimum value, the search for that specific gate A_j is complete.

Once the search for the previous A_j is completed, the MAIN PROGRAM takes the next gate and it becomes A_j. The program repeats the entire search procedure for each A_j, and checks, each time, to make sure the max k_i required for any A_j was stored. The β_i values for each A_j are also stored. After all A_j have been considered, the program prints out all $\{\beta_i / i \epsilon J\}$ for each A_j, the k_i values required, the upper gap limit (u_j), the lower gap limit (ℓ_j), and the sum of the k_i values that were required. This includes all of the information necessary to construct the redundant realization $\hat{R}$.

CHAPTER V

RESULTS AND CONCLUSIONS

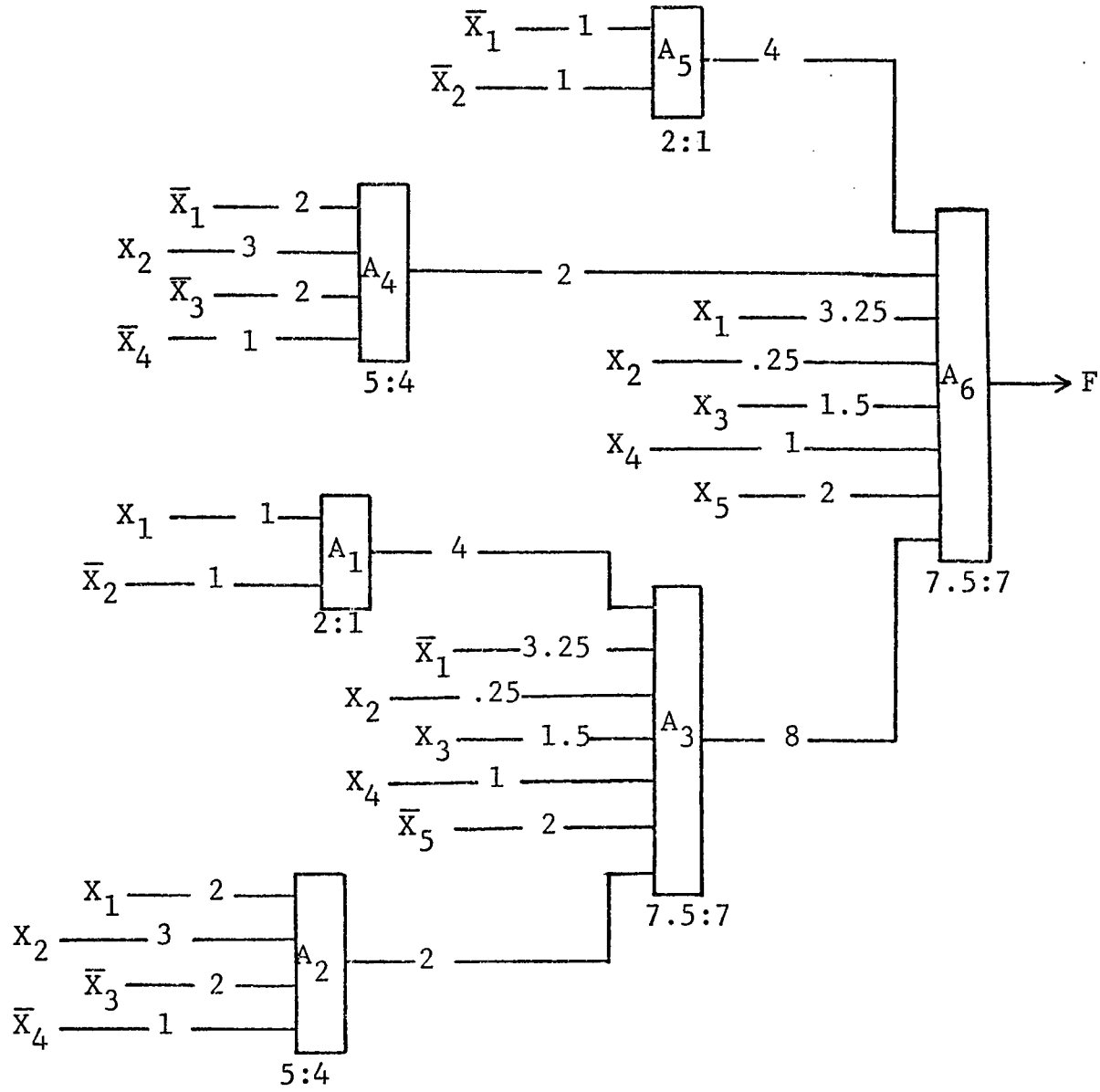
Contours and Results of Method II

Given the weights for all of the inputs to Gate A_j, it is possible to calculate some cost function which can only be minimized if the sum of the gates required is minimized. This cost function is the sum of the gates plus some error function due to constraints not being satisfied. If all the constraints are not satisfied then the answer is not acceptable. In the program, the error function is always checked to make sure it is zero, and if the error is not zero when the minimum is found, the error multiplying constant is increased and the function minimized again.

It is desirable to generate the cost function surface and investigate its characteristics as a function of k. For a large number of gates, the number of points required for a meaningful plot would be quite large, and the information would be very difficult to display for more than two variables.

In order to display the function and gain some knowledge of the nature of the surface, one k_i will be varied while all the other k_i's for the realization will be held at their respective minimum values. The function generated is the cost function with respect to one variable being changed.

The realization shown in Figure 5.1 is a five variable function using six gates. This is a good example to investigate and examine the results that were obtained, since it has a wide



$$f(p) = \langle 3.25X_1 + .25X_2 + 1.5X_3 + X_4 + 2X_5 + 2 \langle 2\bar{X}_1 + 3X_2 + 2\bar{X}_3 + \bar{X}_4 \rangle_{5:4} + 4 \langle \bar{X}_1 + \bar{X}_2 \rangle_{2:1} + 8 \langle 3.25\bar{X}_1 + .25X_2 + 1.5X_3 + X_4 + 2\bar{X}_5 + 2 \langle 2X_1 + 3X_2 + 2\bar{X}_3 + \bar{X}_4 \rangle_{5:4} + 4 \langle X_1 + \bar{X}_2 \rangle_{2:1} \rangle_{7.5:7} \rangle_{7.5:7}$$

Figure 5.1. A six gate, five variable threshold gate realization.

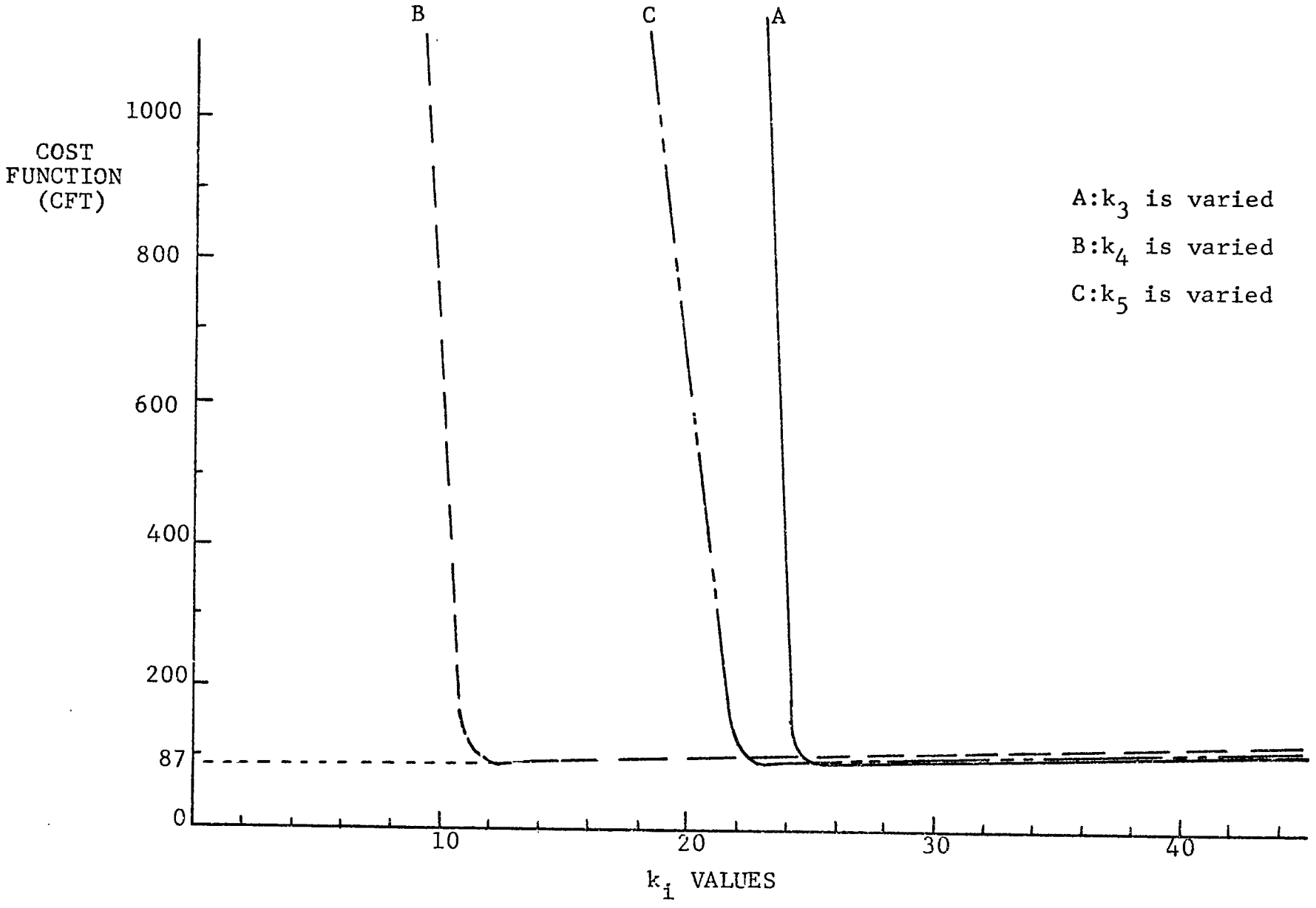


Figure 5.2. The cost function versus one k_i value while other k_i's are at their optimum value and A_j is A_6.

range of weights and relatively narrow gaps which make it more difficult to solve. However, this results in a more interesting contour for most of the values that are plotted.

In Figure 5.2, the cost function versus one k_i value at a time is plotted while all the other k_i's are held at their minimum values. The minimum k_i values found by Method II in this case were k_1=17, k_2=9, k_3=25, k_4=12, k_5=23, and k_6=1.

From Figure 5.2, it is observed that the cost function is a concave function with respect to a variable k_i. The function almost appears to be hyperbolic. This is not the case, however, because there is definitely a minimum value. The function has an extremely steep descent when k_i is too small and approaching the minimum value. After passing the minimum, the function levels off to a steadily increasing function with a slope of 1. This indicates that the error multiplying constant is quite large, but decreasing that constant increases the minimum gap that could possibly be found. Another reason for keeping the constant large is to prevent reaching a minimum without satisfying all constraints.

A characteristic that is very important in minimizing the cost function is the rate of convergence of the k_i values. In order to minimize the cost function successfully in a reasonable length of time, the k_i values must approach their minimum value at an acceptable rate. Figure 5.3 shows the k_i values where $i \varepsilon J$ versus the number of iterations plotted for the realization shown in Figure 5.1 and Gate A_j is gate number 6. An iteration is here defined as a pattern move or a complete exploratory move.

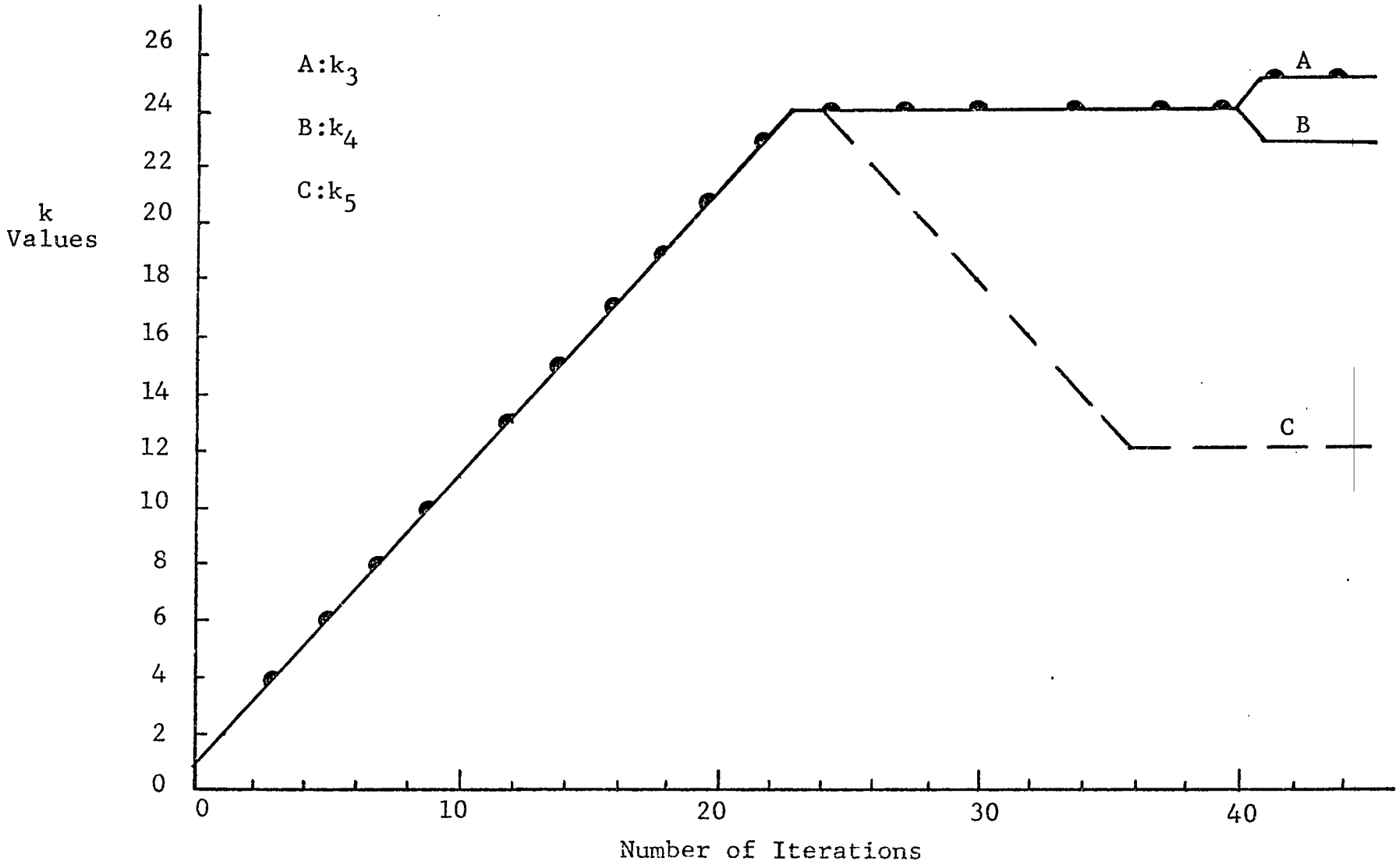


Figure 5.3. Contours showing the rate of convergence of the k_i values for the problem shown in Figure 5.1.($A_j = A_6$).

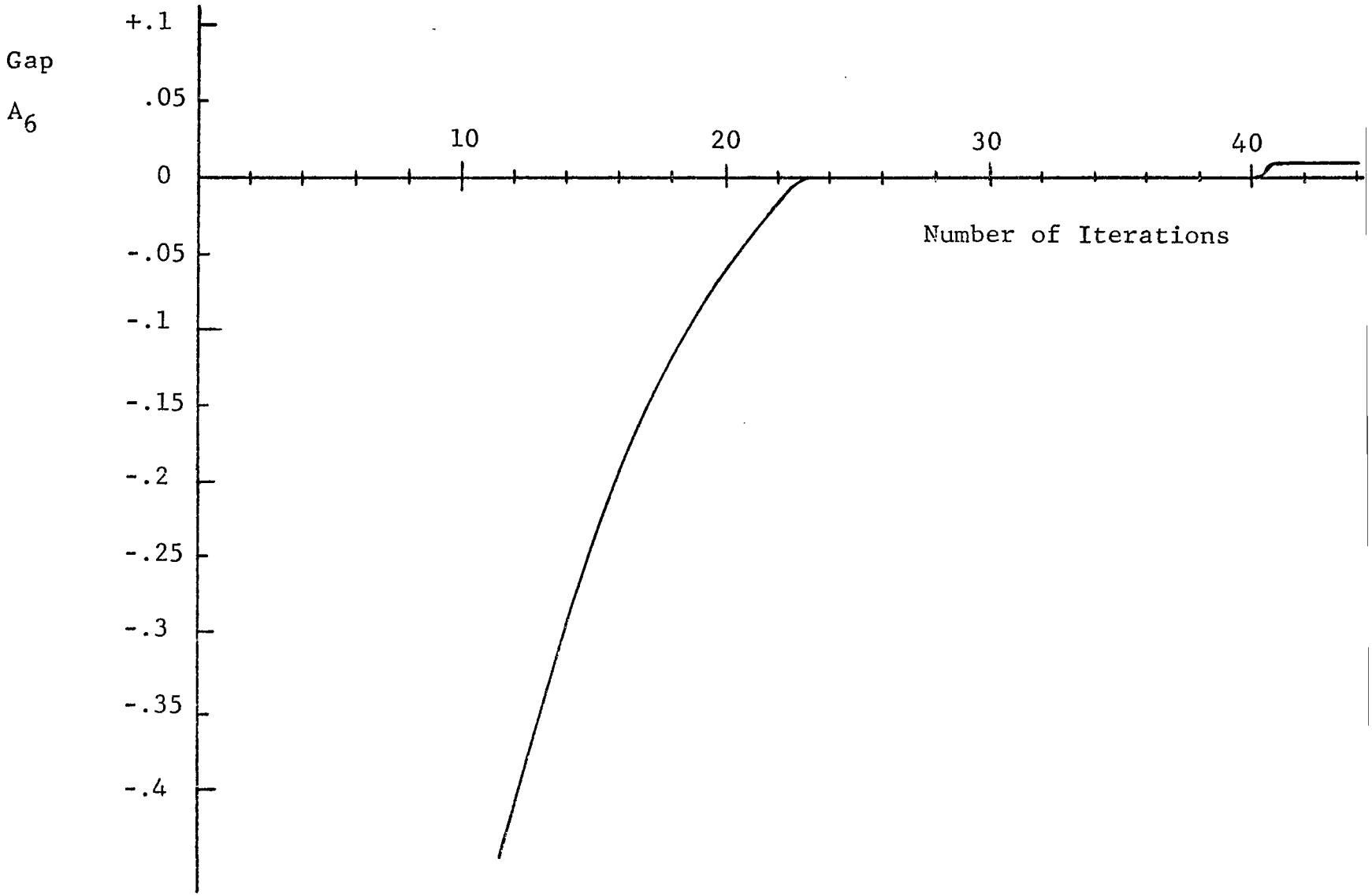


Figure 5.4. The contour of the gap for gate A_6 for the realization in Figure 5.1, solved by Method II.

It is observed, from Figure 5.3, that the gates requiring higher k_i values tend to carry smaller k_i's with them until the larger k_i's approach their required value. This is exactly what would be expected from examining the contour of the cost function. The cost function increases extremely fast if the k_i values used are too small and getting smaller, and it increases at a slow but consistent rate if the k_i values used are too large and getting larger.

Another indication of the rate of convergence toward an acceptable answer is the plot of the gap of the gate correcting errors at any particular time. The plots up to this time have been for k_i's which all feed into A_6. This was done for ease of comparison of the various plots. The plot of the gap for A_6 is shown in Figure 5.4.

The plot of the gap for A_6 is the contour that is expected. While the k_i values are too small, the gap is a negative value. As the k_i values increase, the negative gap increases toward zero. When the k_i values with the larger minimums reach that minimum value, the gap remains at zero until the k_i with the smaller minimum value gets back down in the range of its proper value. There is still an error at this point, but the error is very small. The program checks and there is still an error at the minimum. The error multiplying constant is increased until a minimum is found and the error is zero. At this final point the gap is a positive value.

The program written for Method II was run with several problems. The solutions found for the three problems chosen

for examples are shown in Table 5.1. Each problem was solved using an initial starting point where all k values were equal to 1. Then each problem was solved again using a starting point where all k values were greater than the answer originally found.

The separating functions for the example problems in Table 5.1 are:

$$\text{Prob. 1 : } f(p) = \langle 2\bar{X}_1 + \bar{X}_2 + X_3 + X_4 + 2 \langle 2X_1 + 2X_2 + X_3 + \bar{X}_4 \rangle_{5:4} + 2 \langle X_1 + X_2 \rangle_{2:1} \rangle_{4:3}$$

$$\text{Prob. 2 : } f(p) = \langle X_1 + X_2 + \bar{X}_3 + 2X_5 + 5 \langle 2\bar{X}_1 + 2\bar{X}_2 + X_3 + \bar{X}_4 + 2X_5 + 5 \langle X_1 + X_3 + X_4 + X_5 \rangle_{4:3} \rangle_{7:6} + 5 \langle \bar{X}_1 + X_2 + \bar{X}_3 + \bar{X}_5 \rangle_{4:3} \rangle_{5:4}$$

$$\text{Prob. 3: } f(p) = \langle 3.25X_1 + .25X_2 + 1.5X_3 + X_4 + 2X_5 + 2 \langle 2\bar{X}_1 + 3X_2 + 2\bar{X}_3 + \bar{X}_4 \rangle_{5:4} + 4 \langle \bar{X}_1 + \bar{X}_2 \rangle_{2:1} + 8 \langle 3.25\bar{X}_1 + .25X_2 + 1.5X_3 + X_4 + 2\bar{X}_5 + 2 \langle 2X_1 + 3X_2 + 2\bar{X}_3 + \bar{X}_4 \rangle_{5:4} + 4 \langle X_1 + \bar{X}_2 \rangle_{2:1} \rangle_{7.5:7} \rangle_{7.5:7}$$

The realization for Problem 2 is shown in Figure 1.5 and the realization for Problem 3 is shown in Figure 5.1.

TABLE 5.1

RESULTS OF THE METHOD II PROGRAM

PROB. NO.	INITIAL k VALUES OF GATE NO. 1,2,3,4,5,6	FINAL k VALUES OF GATE NO. 1,2,3,4,5,6	FINAL SUM OF k VALUES	FINAL GAP OF GATE NO. 1,2,3,4,5,6	RUN TIME IN MIN
1	1,1,1	5,5,1	11	1.0,1.0,0.2	1.16
1	9,9,1	5,5,1	11	1.0,1.0,0.2	1.16
2	1,1,1,1	6,6,6,1	19	1.0,0.1666,1.0,0.1666	1.22
2	11,11,11,1	6,6,6,1	19	1.0,0.1666,1.0,0.1666	1.22
3	1,1,1,1,1,1	17,9,25,12,23,1	87	1.0,1.0,0.0294,1.0,1.0,0.0061	1.20
3	27,15,37,15,37,1	17,9,29,9,18,1	83	1.0,1.0,0.0294,1.0,1.0,0.0019	1.32

The Method II separating functions required to correct one error of the problems in Table 5.1 are:

$$\text{Prob. 1 : } f(p) = \langle 2\bar{X}_1 + \bar{X}_2 + X_3 + X_4 + 2 \langle 2X_1 + 2X_2 + X_3 + \bar{X}_4 \rangle^{5}_{5:4} + 2 \langle X_1 + X_2 \rangle^{5}_{2:1} \rangle_{3.6:3.4}$$

$$\text{Prob. 2 : } f(p) = \langle X_1 + X_2 + \bar{X}_3 + 2X_5 + 5 \langle 2\bar{X}_1 + 2\bar{X}_2 + X_3 + \bar{X}_4 + 2X_5 + 5 \langle X_1 + X_3 + X_4 + X_5 \rangle^{6}_{4:3} \rangle^{6}_{7:6.83} + 5 \langle \bar{X}_1 + X_2 + \bar{X}_3 + \bar{X}_5 \rangle^{6}_{4:3} \rangle_{5:4.83}$$

$$\text{Prob. 3 : } f(p) = \langle 3.25X_1 + .25X_2 + 1.5X_3 + X_4 + 2X_5 + 2 \langle 2\bar{X}_1 + 3X_2 + 2\bar{X}_3 + \bar{X}_4 \rangle^{12}_{5:4} + 4 \langle \bar{X}_1 + \bar{X}_2 \rangle^{23}_{2:1} + 8 \langle 3.25\bar{X}_1 + .25X_2 + 1.5X_3 + X_4 + 2\bar{X}_5 + 2 \langle 2X_1 + 3X_2 + 2\bar{X}_3 + \bar{X}_4 \rangle^{9}_{5:4} + 4 \langle X_1 + \bar{X}_2 \rangle^{17}_{2:1} \rangle^{25}_{7.264:7.235} \rangle_{7.326:7.320}$$

Upon examining the data everything seems to be as expected in the case of Problems 1 and 2. The run times vary slightly with different starting points, but the minimum found is the same. In the case of Problem 3 the final results are not the same. Some of the individual k values are different and the sum of the final k values is not exactly the same for runs with different starting points. This appears to be an error in the Method II Program, but both of these answers may be checked and

both are minimums for the function. A minimum is defined as any solution so that the reduction of any k_i value will result in some constraint not being satisfied.

The results of Table 5.1 point out that with more complex realizations the cost function is not a contour having a single minimum. In problems that are more difficult to solve there appear to be "local minima" very close in value to the absolute minimum of the cost function. Gate A_6 is the only gate in Problem 3 which shows any variation in the k_i values feeding into it. The reason for this is the wide variation in the input weights to A_6 and the low relative gap. A low relative gap means that the gap is narrow compared to the distance it is away from the origin on the map of the separating function.

On the basis of the contours plotted and the data tabulated in Table 5.1, some conclusions may be drawn concerning the surface of the cost function and the performance of the Method II Program. These conclusions are:

1) The program for Method II may be started at any initial starting point for the k values, and it will find a minimum sum of k values.
2) As the complexity of the problem increases there is a higher probability that this could be a local minimum in the near vicinity of the absolute minimum. However, the sum of the k values for this minimum should be close to the sum of the k values for the absolute minimum.

3) By starting the program at widely varying initial values of k, it is possible to check, and see if the absolute minimum has been found.

4) Method II finds the minimum number of gates necessary to correct t errors using the weights that were read in for all the inputs in the realization.

Contours and Results of Method III

Method III attempts to improve the solution of Method II by reducing the number of gates required to correct the same number of errors. Method II takes a given realization and attempts to find the minimum number of gates necessary to correct a specified number of errors using multiplexing techniques. Method III finds a Method II solution and then changes the β_{ij} values in an attempt to minimize even further the number of gates necessary to correct the same number of errors. A Method III solution must realize the original output function. In effect, Method III performs a Method II search after each β_{ij} change except that it must calculate the new gap in the absence of errors for Gate A_j using the new β_{ij} values.

The Method III Program would have a cost function similar to that shown for Method II during each search for the minimum k values. The k values are searched after every change of each β_{ij}. A characteristic that helps point out how a solution is reached is a plot of the k_i values versus the number of iterations required. However, in Method III an iteration is considered to be each pattern move or completed exploratory move on the β_{ij} values.

Therefore, the k_i value plotted is the k_i value in the minimum sum of gates necessary to correct the specified number of errors for a particular set of β_{ij}'s.

In Method III a more meaningful plot is a plot of β_{ij} and its respective k_i with each iteration. In order to minimize the number of gates required, both k_i and β_{ij} for all $i \epsilon J$ must approach an optimum value in a reasonable number of iterations. Figure 5.5 shows the contour of a k_i and the respective β_{ij} for the realization shown in Figure 5.1 and previously referred to as Problem 3.

The contours plotted in Figure 5.5 show the variation of the k_i and β_i values with each iteration of the Method III Program. The values plotted are the best results found up to some particular iteration. Each iteration is a pattern move or a completed exploratory move of the β_{ij} values. It is shown by these plots that the program converges fairly rapidly to the final solution. The curves point to the fact that the correct answer may be searched out quite reliably.

For a comparison, it would be interesting to show a plot of the variation in the results of the k_i and β_{ij} values with each change of any β_{ij}. The plot of this curve for k_3 and β_{36} of Problem 3 is shown in Figure 5.6. This curve shows that the results found vary quite erratically for variations in β_{ij}. This curve shows the contour from which the program must select the correct values. Figure 5.6 may be compared to the corresponding values in Figure 5.5

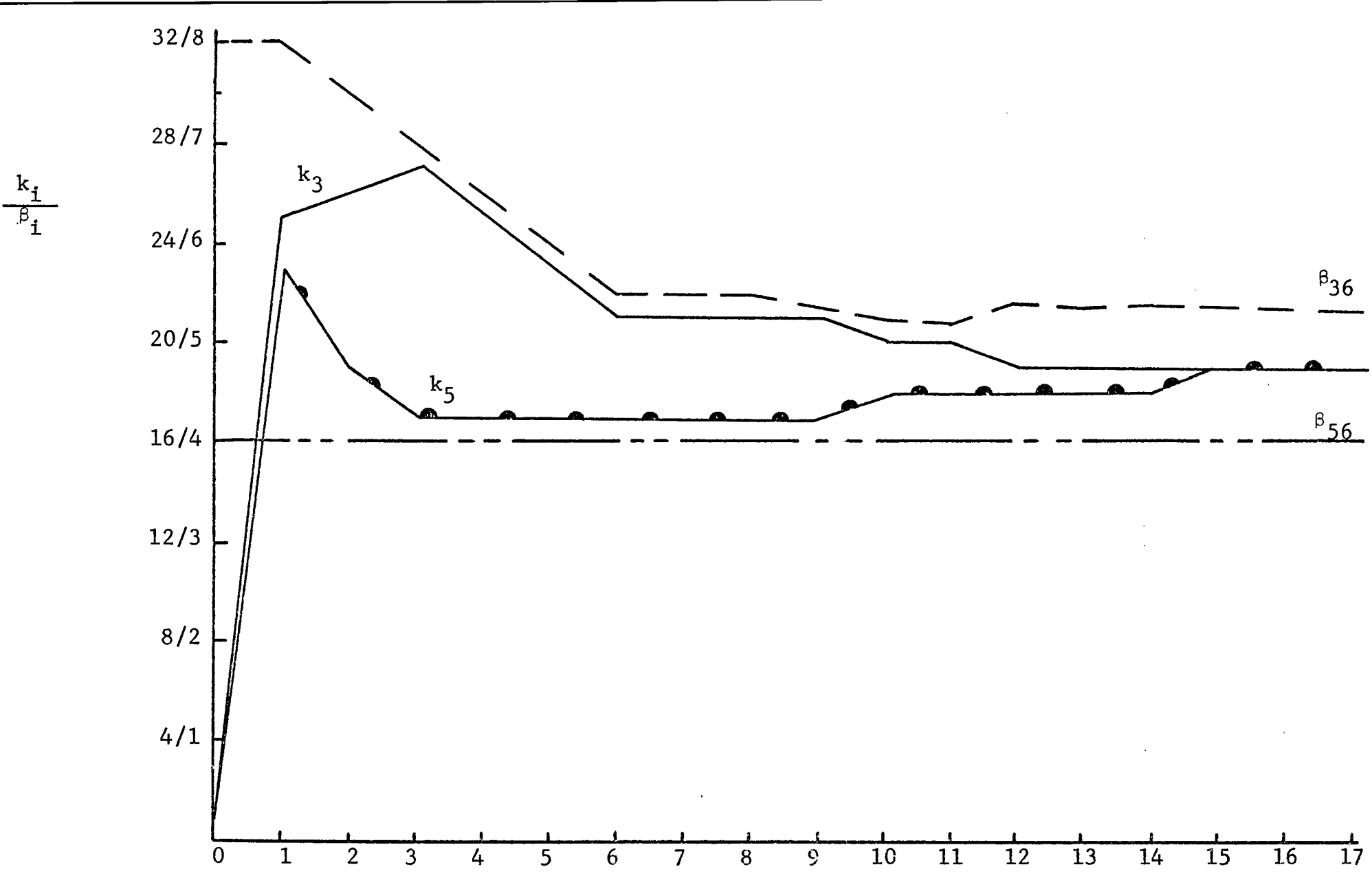


Figure 5.5. Contours of β_{36}, k_3, β_{56}, and k_5 versus the number of iterations to reach a solution of Problem 3. $(A_j=A_6)$

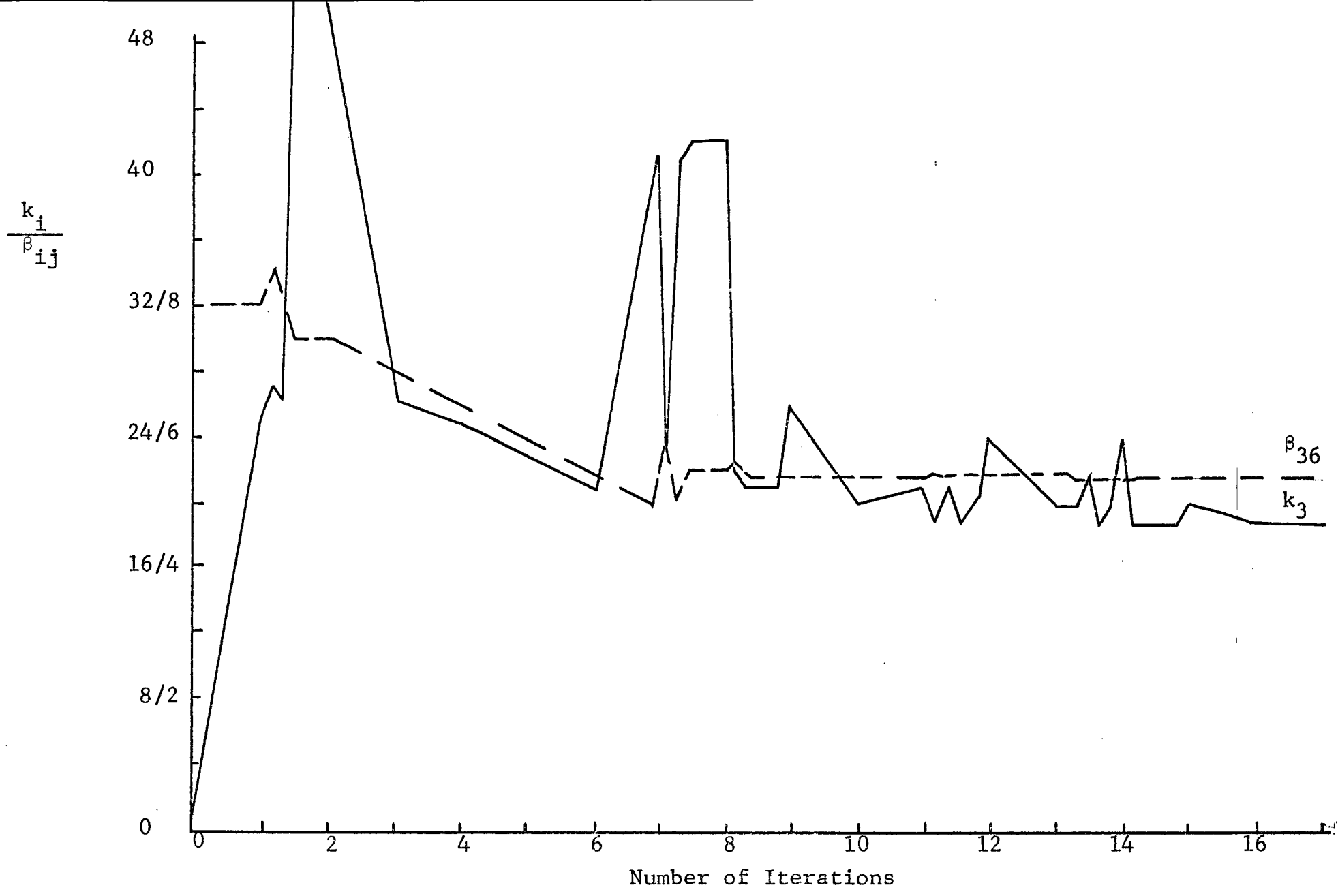


Figure 5.6. Plot showing the variations in k_3 and β_{36} for attempted β_{ij} values on Problem 3. ($A_j = A_6$)

The program written for Method III was run with the same problems that were run on Method II. The results of these runs for the three problems are shown in Table 5.2. Each problem was solved using an initial starting point of all k values equal to 1. Then each problem was solved again using the same starting point used in Method II where all k values were greater than the answer originally found. Shown in Table 5.2 are the values required for correcting one error of Problems 1, 2, and 3 by Method III.

The Method III solutions of the separating functions of Problems 1, 2, and 3 are as follows:

$$\text{Prob. 1 : } f(p) = \langle 2\bar{X}_1 + \bar{X}_2 + X_3 + X_4 + 2.668 \langle 2X_1 + 2X_2 + X_3 + \bar{X}_4 \rangle^{4}_{5:4} + 2.0 \langle X_1 + X_2 \rangle^{3}_{2:1} \rangle_{4:3.667}$$

$$\text{Prob. 2 : } f(p) = \langle X_1 + X_2 + \bar{X}_3 + 2X_5 + 4.0 \langle 2\bar{X}_1 + 2\bar{X}_2 + X_3 + \bar{X}_4 + 2X_5 + 4.804 \langle X_1 + X_3 + X_4 + X_5 \rangle^{6}_{4:3} \rangle^{5}_{7:6.801} + 4.0 \langle \bar{X}_1 + X_2 + \bar{X}_3 + \bar{X}_5 \rangle^{5}_{4:3} \rangle_{5:4.800}$$

$$\text{Prob. 3 : } f(p) = \langle 3.25X_1 + .25X_2 + 1.5X_3 + X_4 + 2X_5 + 2.2 \langle 2\bar{X}_1 + 3X_2 + 2\bar{X}_3 + X_4 \rangle^{8}_{5:4} + 4.0 \langle \bar{X}_1 + \bar{X}_2 \rangle^{19}_{2:1} + 5.3 \langle 3.25\bar{X}_1 + .25X_2 + 1.5X_3 + X_4 + 2\bar{X}_5 + 2.008 \langle 2X_1 + 3X_2 + 2\bar{X}_3 + \bar{X}_4 \rangle^{10}_{5:4} + 4.092 \langle X_1 + \bar{X}_2 \rangle^{14}_{2:1} \rangle^{19}_{7.299:7.292} \rangle_{7.2894:7.2789}$$

TABLE 5.2

RESULTS OF THE METHOD III PROGRAM

PROB. NO	INITIAL k VALUES FOR GATE NO. 1,2,3,4,5,6	FINAL k VALUES FOR GATE NO. 1,2,3,4,5,6	FINAL SUM OF k VALUES	FINAL GAP FOR GATE NO. 1,2,3,4,5,6	NEW BETA VALUES OTHER THAN 0	RUN TIME IN MIN.
1	1,1,1	4,3,1	8	1.0,1.0,0.3330	β_{13}=2.668, β_{23}=2.0	2.52
1	9,9,1	4,3,1	8	1.0,1.0,0.3330	β_{13}=2.668, β_{23}=2.0	2.65
2	1,1,1,1	6,5,5,1	17	1.0,0.19933,1.0, 0.2	β_{12}=4.804, β_{24}=4.0 β_{34} = 4.0	3.04
2	11,11,11,1	6,5,5,1	17	1.0,0.19933,1.0, 0.2	β_{12}=4.804, β_{24}=4.0 β_{34}=4.0	2.89
3	1,1,1,1,1,1	14,10,19,8,19,1	71	1.0,1.0,.00692,1.0, 1.0,0.01053	β_{13}=4.092, β_{23}=2.008 β_{36}=5.3, β_{46}=2.2 β_{56}=4.0	5.89
3	27,15,37,15, 37,1	14,10,19,8,19,1	71	1.0,1.0,0.00692,1.0 1.0,0.01053	β_{13}=4.092, β_{23}=2.008 β_{36}=5.3, β_{46}=2.2 β_{56}=4.0	5.89

Upon examining the data of Table 5.2, everything seems to be about as one would expect. In the case of Problem 3, it is found that the Method III Program is able to find the minimum at either starting point. The more difficult problem does not prevent the minimum from being found. It appears that the technique used in Method III not only allows the problems to be solved with fewer gates, but it apparently enables the absolute minimum to be found with better accuracy.

On the basis of the contours plotted and the data tabulated in Table 5.2, some conclusions may be drawn concerning the performance of the Method III Program. These conclusions are:

1) The program for Method III may be started at any initial starting point for the k values, and it will find a minimum sum of k values.

2) Method III improves on the Method II Program because it is allowed to change the β_{ij} values in order to improve the results. Method III performs a search on the β_{ij} values in an attempt to optimize them.

3) By starting the program at widely varying initial values of k it is possible to check and see if the absolute minimum has been found.

4) Method III finds the minimum number of gates necessary to correct t errors using the original realization which was read in, but it then allows the β_{ij}'s to take on an optimum value.

Results When Specifying Minimum Gaps

All of the previous results were solutions which had no minimum gap specified. Any gap that was found by the programs,

regardless of how small, was accepted. It is possible to specify a minimum gap and require all gates in the realization to have at least that wide a gap. This means a practical or a realizable limit may be specified.

Before any minimum gaps were specified, an error was made if any $v_i \leq h_q$ where $i,q \in J$. An error resulted if there was no required gap for the threshold T. An error was also made if $v_i \leq$ (the lower limit for the gap) or $h_i \geq$ (the upper limit for the gap).

In order to specify a minimum gap, the gap must be at least as wide as the gap specified. Therefore, v_i must be $\geq h_q$ + MINGAP where $i,q \in J$ and MINGAP is the minimum gap required. Also v_i must be $\geq$ (the lower limit for the gap + MINGAP and $h_i \leq$ (the upper limit for the gap + MINGAP). Some constraint will not be satisfied if the gap is not at least as wide as the minimum gap specified.

To specify a minimum gap for all the gates in a realization requires changing the same three cards in either program. The cards calculating ELL, EUL, and EFG must be modified in the SEARCH subroutine of either program. The three cards before the modification read as follows:

```
Method II;
      ELL = L(J)-V(I2)
      EUL = H(I2)-U(J)
      EFG = (Z(I3)+A(IG,J)*ERR)/KT(I3))-(W(J1)
           -(A(IH,J)*ERR)/KT(J1)).
```

Method III;

```
ELL = LNE(J)-V(I2)
EUL = H(I2)-UNE(J)
EFG = (Z(I3)+(BT(I3,J)*ERR)/KT(I3))-(W(J1)
      -(BT(J1,J)*ERR)/KT(J1)).
```

After the modification to specify the minimum gap, they read as follows:

Method II;

```
ELL= L(J)-V(I2)+MINGAP
EUL= H(I2)-U(J)+MINGAP
EFG= (Z(I3)+A(IG,J)*ERR)/KT(I3))-(W(J)
     -(A(IH,J)*ERR)/KT(J1)-MINGAP)
```

Method III;

```
ELL= LNE(J)-V(I2)+MINGAP
EUL= H(I2)-UNE(J)+MINGAP
EFG= (Z(I3)+BT(I3,J)*ERR)/KT(I3)-(W(J1)
     -(BT(J1,J)*ERR)/KT(J1)-MINGAP)
```

where MINGAP is the minimum gap specified as a floating point number.

For a basis of comparison of the problems with and without a minimum gap specified, a minimum gap of 10% of the smallest input weight to any gate was chosen. However, for Problems 1 and 2 the gap found by Methods II and III for any gate was already greater than 10% of the smallest weight into any gate. Then as a random value, a minimum gap of (.5) for both Problem 1 and 2 was chosen. The gap already found for Problem 3 was very

TABLE 5.3.

(a) RESULTS OF THE PROBLEMS USING SPECIFIED MINIMUM GAPS BY METHOD II.

(b) RESULTS OF THE PROBLEMS USING SPECIFIED MINIMUM GAPS BY METHOD III.

(a)

PROB. NO.	INITIAL k VALUES FOR GATE NO. 1,2,3,4,5,6	FINAL k VALUES FOR GATE NO. 1,2,3,4,5,6	SUM OF FINAL k VALUES	FINAL GAP FOR GATE NO. 1,2,3,4,5,6	MIN. GAP REQ.	RUN TIME IN MIN.
1	1,1,1	9,9,1	19	1.0,1.0,0.5555	.5	1.37
2	1,1,1,1	11,11,11,1	34	1.0,0.545,1.0,0.545	.5	1.17
3	1,1,1,1,1,1	17,9,25,13,26,1	91	1.0,1.0,0.0294,1.0, 1.0,0.0262	.025	1.20

(b)

PROB. NO.	INITIAL k VALUES FOR GATE NO. 1,2,3,4,5,6	FINAL k VALUES FOR GATE NO. 1,2,3,4,5,6	FINAL SUM OF k VALUES	FINAL GAP FOR GATE NO. 1,2,3,4,5,6	NEW BETA VALUES OTHER THAN 0	MIN. GAP REQ.	RUN TIME IN MIN.
1	1,1,1	6,5,1	12	1.0,1.0,0.6	$\beta_{13}=2.4, \beta_{23}=2.0$	.5	2.55
2	1,1,1,1	10,8,8,1	27	1.0,0.555,1.0 .5625	$\beta_{12}=4.448, \beta_{24}=3.5$ $\beta_{34}=3.5$	.5	3.04
3	1,1,1,1,1,1	15,10,23,9,17,1	75	1.0,1.0,.02773, 1.0,1.0,0.03294	$\beta_{13}=4.072$ $\beta_{23}=2.008$ $\beta_{36}=5.3239$ $\beta_{46}=2.0$ $\beta_{56}=4.004$	.025	7.53

close to the minimum gap specified but slightly lower. The results for the three problems run, with minimum gaps specified, are tabulated in Table 5.3 (a and b).

The results shown in Table 5.3 (a and b) confirm the results that are expected. A larger number of gates are required to make the gaps of the gates in the problem wider. Any reasonable minimum gap may be specified. It is not reasonable to ask the program to leave a wider minimum gap than the gap originally read in for any gate on the realization.

Summary of Conclusions

Two procedures have been developed, using two different methods, and the programs written for minimizing the number of gates required to correct t errors in a threshold realization using multiplexing techniques. Both of these methods require that some original realization be read in as input data. In obtaining these realizations it was found that:

1) Method II always finds a minimum number of gates required to correct t errors using the weights originally read in.

2) Due to "local minimums" it is possible in some situations for Method II to find a minimum sum of gates which is not the absolute minimum sum of gates. However, this minimum sum of gates is probably close to the actual minimum sum of gates.

3) Method III finds a minimum number of gates required to correct t errors by optimizing the β_{ij} values of the realization.

4) The minimum found by Method III should always be as good or better than the minimum found by Method II.

5) The cost function of both programs will always have a minimum. Therefore, there is a solution assuming the original realization read in was correct.

6) Either method allows a minimum gap to be specified for all gates in the realization found.

BIBLIOGRAPHY

1. Bargainer, J. D., Jr. and Coates, C. L., "Minimal Multiplexed Threshold Gate Realizations," I.E.E.E. Transactions on Computers, Vol. EC-17, No. 6, pp. 566-578, June, 1968.

2. Coates, C. L. and Lewis, P. M., "Linearly Separable Switching Functions," J. Franklin Inst., Vol. 272, pp. 366-419, November, 1961.

3. Coates, C. L. and Lewis, P. M., III, "Donut: A Threshold Gate Computer," I.E.E.E. Transactions on Electronic Computers, Vol. EC-13, No. 3, pp. 240-248, June, 1964.

4. Hooke, R. and Jeeves, T. A., "'Direct Search' Solutions of Numerical and Statistical Problems," Association for Computing Machinery Journal, Vol.8, pt. 2, April, 1961, pp. 212-229.

5. Lewis, P. M. and Coates, C. L., Threshold Logic. New York: Wiley, 1967.

6. Liu, C. L. and Liu, J. W., "On A Multiplexing Scheme for Threshold Logical Elements," Information and Control, Vol. 8, pp. 282-294, 1965.

7. Muroga, S., Toda, I., Ikasu, S., "The Theory of Majority Decision Elements," J. Franklin Institute, Vol. 272, pp. 376-419, May, 1961.

8. Von Newmann, J., "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components," in Automata Studies, Shannon, C. E. and McCarthy, J., Eds Princeton, N. J.: Princeton University Press, pp. 43-98, 1956.

9. Weisman, J. and Wood, C. F., "The Use of 'Optimal Search' for Engineering Design," Recent Advances in Optimization Techniques, pp. 219-228, Wiley, 1966.

10. Wilde, D. J., Optimum Seeking Methods, Englewood Cliff, N. J.: Prentice-Hall, Inc., 1964.

APPENDIX

VARIABLE LIST FOR METHOD II PROGRAM

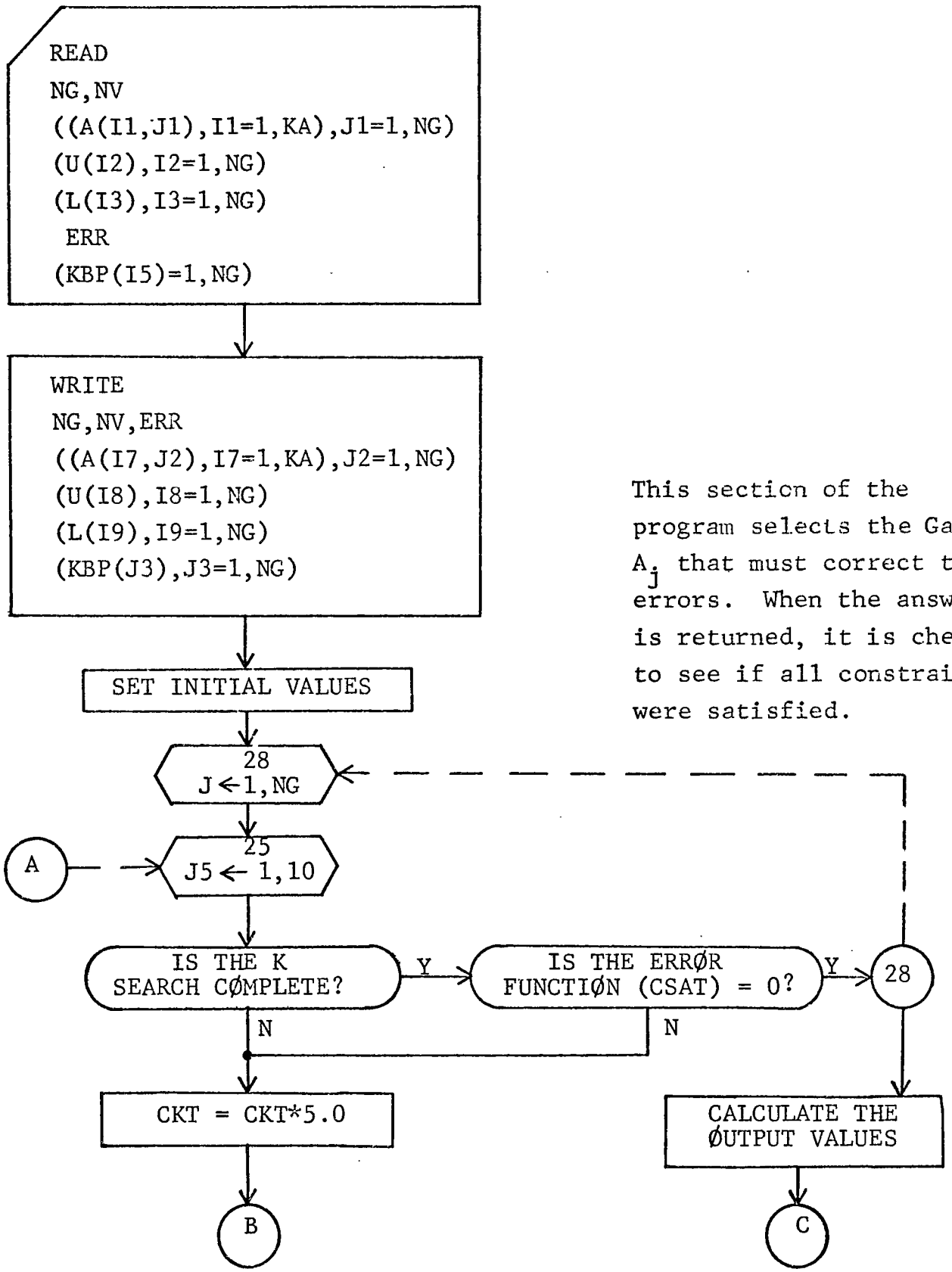
C PROGRAM DOCUMENTATION

VARIABLE LIST

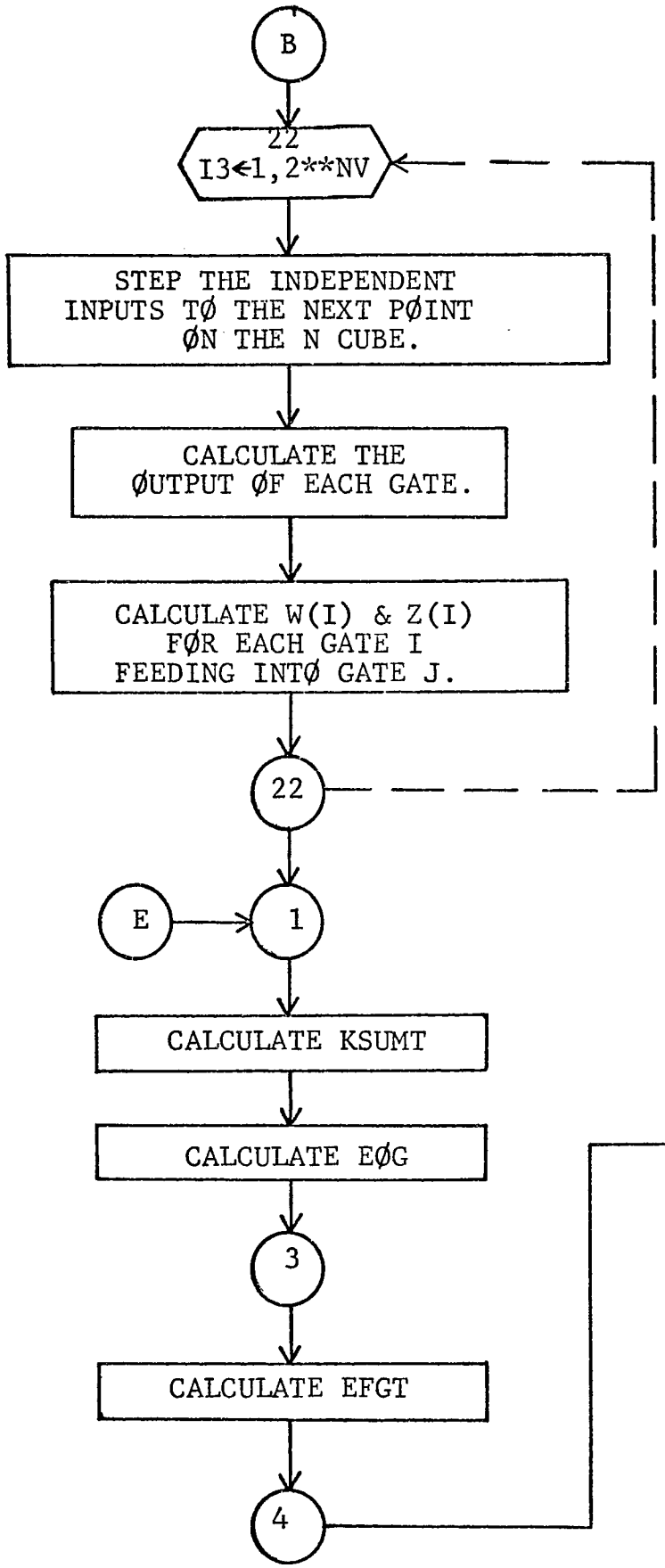
C F(I) = OUTPUT FUNCTION VALUE FOR GATE I.(1 OR 0)
C U(J) = UPPER LIMIT FOR THE THRESHOLD OF GATE J.
C L(J) = LOWER LIMIT FOR THE THRESHOLD OF GATE J.
C UN(J) = UPPER LIMIT FOR THE THRESHOLD OF GATE J NECESSARY TO CORRECT
C ERR ERRORS.
C LN(J) = LOWER LIMIT FOR THE THRESHOLD OF GATE J NECESSARY TO CORRECT
C ERR ERRORS.
C GAP(J) = ACTUAL GAP OF GATE J AT A PARTICULAR TIME.
C KT(I) = TEMPORARY K VALUES BEING USED IN AN ATTEMPT TO REDUCE THE
C COST FUNCTION.
C KBP(I) = THE BEST K VALUES FOUND IN PRECEEDING ITERATIONS.(BASE POINT
C K VALUES)
C KU(I) = TEMPORARY STORAGE FOR THE IMPROVED KT VALUES DURING AN
C EXPLORATORY SEARCH, BEFORE COMPARING THE KT VALUES TO THE
C KBP VALUES.
C Z(I) = THE MAXIMUM VALUE OF THE SEPARATING FUNCTION OF GATE J SUCH
C THAT F(J)=1 AND F(I)=1, WHERE THE OUTPUT OF GATE I IS AN
C INPUT TO GATE J.
C W(I) = THE MINIMUM VALUE OF THE SEPARATING FUNCTION OF GATE J SUCH
C THAT F(J)=0 AND F(I)=0, WHERE THE OUTPUT OF GATE I IS AN
C INPUT TO GATE J.
C X(I) = THE VALUE OF THE INDEPENDENT INPUT X(I) AT A PARTICULAR POINT
C ON THE N CUBE.(1 OR 0)
C XN(I) = THE VALUE OF THE INDEPENDENT INPUT XNOT(I) AT A PARTICULAR
C POINT ON THE N CUBE.(1 OR 0)
C NG = NO OF GATES IN THE REALIZATION.
C NV = NO OF INDEPENDENT VARIABLES IN THE REALIZATION.
C ERR = NO OF ERRORS THAT ARE TO BE CORRECTED.
C A(I,J) = MATRIX OF WEIGHTS FOR ALL INPUTS I FEEDING INTO GATE J. (I(1
C THRU NV): WEIGHTS FOR INDEPENDENT INPUTS X(1) THRU X(NV),
C I(NV+1 THRU 2NV): WEIGHTS FOR INDEPENDENT INPUTS XN(1) THRU
C XN(NV),I(2NV THRU 2NV+NG): WEIGHT FOR GATE I FEEDING INTO
C GATE J.
C VAR(I) = VALUE OF INPUT I FEEDING INTO GATE J.(1 OR 0)
C V(I) = MINIMUM VALUE OF THE SEPARATING FUNCTION OF GATE J WITH ERR
C ERRORS MADE IN THE I SET OF GATES, WHERE F(I)=1 AND F(J)=1.
C H(I) = MAXIMUM VALUE OF THE SEPARATING FUNCTION OF GATE J WITH ERR
C ERRORS MADE IN THE I SET OF GATES, WHERE F(I)=0 AND F(J)=0.
C CKT = THE TEMPORARY VALUE OF THE CONSTRAINTS ERROR MULTIPLYING
C FACTOR, ERRORS DUE TO THERE NOT BEING A GAP ARE MULTIPLIED
C BY CKT.
C CKP = THE PREVIOUS VALUE OF THE CONSTRAINTS ERROR MULTIPLYING

```
C          FACTOR.
C     KSC  =A FLAG INDICATING THE SEARCH FOR THE MINIMUM K VALUES IS
C          COMPLETE.
C     KSUMT=SUM OF KT VALUES AT ANY TIME.
C     EGG  =THE ERROR DUE TO THE GAP WITH ERR ERRORS NOT BEING
C          WITHIN THE GAP ESTABLISHED WHEN NO ERRORS ARE MADE.
C     EFGT =THE ERROR DUE TO THERE NOT BEING A GAP FOR THE THRESHOLD
C          WHEN ERR ERRORS ARE MADE.
C     KESP =A FLAG INDICATING THAT THE SEARCH ROUTINE IS PERFORMING AN
C          EXPLORATORY MOVE.
C     KPMP =A FLAG INDICATING THAT THE SEARCH ROUTINE IS PERFORMING A
C          PATTERN MOVE.
C     KPOS =A FLAG INDICATING WHETHER A KT VALUE WAS INDEXED POSITIVE OR
C          NEGATIVE IN THE LAST STEP.
C     IWSO =A FLAG INDICATING WHETHER ANY KT VALUE REDUCED THE COST
C          FUNCTION IN THE LAST EXPLORATORY MOVE.
C     KOSUM=SUM OF THE BEST K VALUES FOUND.
C     UOUT(J)=THE UPPER LIMIT FOR THE THRESHOLD OF GATE J USING THE BEST
C          K VALUES.
C     LOUT(J)=THE LOWER LIMIT FOR THE THRESHOLD OF GATE J USING THE BEST
C          K VALUES.
C     KOUT(I)=THE MINIMUM NUMBER OF GATES IN SET I REQUIRED TO CORRECT ERR
C          ERRORS IN THE GIVEN REALIZATION.
C
C     NOTE: GATES MUST BE NUMBERED IN ASCENDING ORDER TO THE OUTPUT GATE.
C           NO GATE CAN FEED INTO A GATE WITH A LOWER NUMBER THAN ITS OWN
C           NUMBER.
```

FLØW CHART FØR THE METHØD 2 PRØGRAM



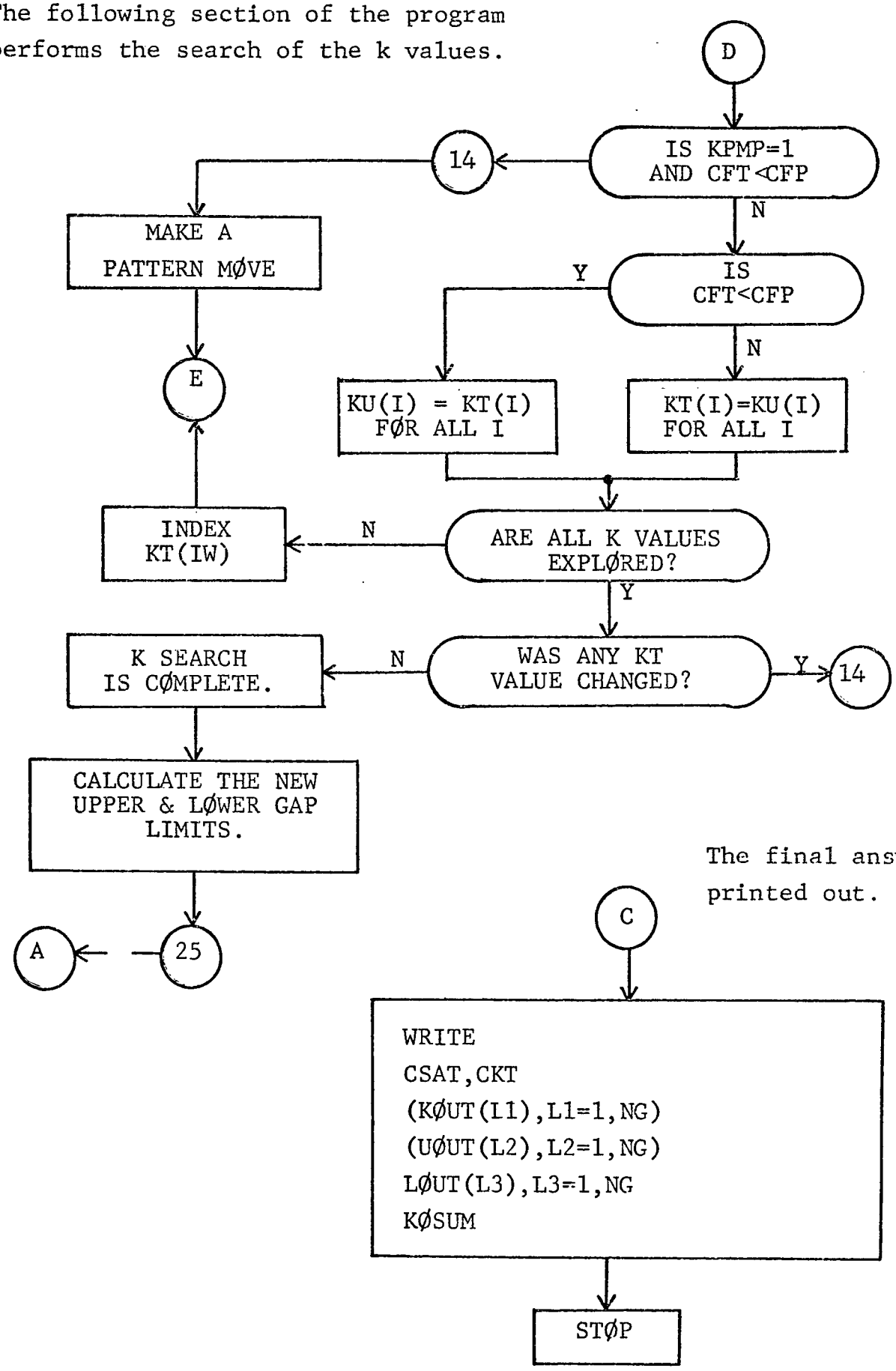
This section of the program selects the Gate A_j that must correct the errors. When the answer is returned, it is checked to see if all constraints were satisfied.



The following section of the program calculates the w_i and z_i values for each Gate A_i feeding into A_j.

The following section of the program (1 thru D) calculates the cost function and the error due to constraints not being satisfied.

The following section of the program performs the search of the k values.



The final answer is printed out.

METHOD II PROGRAM

```
C                    PROGRAM DOCUMENTATION
C
C
C                          VARIABLE LIST
C
C     F(I)    -OUTPUT FUNCTION VALUE FOR GATE I.(1 OR 0)
C     U(J)    -UPPER LIMIT FOR THE THRESHOLD OF GATE J.
C     L(J)    -LOWER LIMIT FOR THE THRESHOLD OF GATE J.
C     UN(J)   -UPPER LIMIT FOR THE THRESHOLD OF GATE J NECESSARY TO CORRECT
C              ERR ERRORS.
C     LN(J)   -LOWER LIMIT FOR THE THRESHOLD OF GATE J NECESSARY TO CORRECT
C              ERR ERRORS.
C     GAP(J)  - ACTUAL GAP OF GATE J AT A PARTICULAR TIME.
C     KT(I)   -TEMPORARY K VALUES BEING USED IN AN ATTEMPT TO REDUCE THE
C              COST FUNCTION.
C     KBP(I)  -THE BEST K VALUES FOUND IN PRECEEDING ITERATIONS.(BASE POINT
C              K VALUES)
C     KU(I)   -TEMPORARY STORAGE FOR THE IMPROVED KT VALUES DURING AN
C              EXPLORATORY SEARCH, BEFORE COMPARING THE KT VALUES TO THE
C              KBP VALUES.
C     Z(I)    -THE MAXIMUM VALUE OF THE SEPARATING FUNCTION OF GATE J SUCH
C              THAT F(J)=1 AND F(I)=1, WHERE THE OUTPUT OF GATE I IS AN
C              INPUT TO GATE J.
C     N(I)    -THE MINIMUM VALUE OF THE SEPARATING FUNCTION OF GATE J SUCH
C              THAT F(J)=0 AND F(I)=0, WHERE THE OUTPUT OF GATE I IS AN
C              INPUT TO GATE J.
C     X(I)    -THE VALUE OF THE INDEPENDENT INPUT X(I) AT A PARTICULAR POINT
C              ON THE N CUBE.(1 OR 0)
C     XN(I)   -THE VALUE OF THE INDEPENDENT INPUT XNOT(I) AT A PARTICULAR
C              POINT ON THE N CUBE.(1 OR 0)
C     NG      -NO OF GATES IN THE REALIZATION.
C     NV      -NO OF INDEPENDENT VARIABLES IN THE REALIZATION.
C     ERR     -NO OF ERRORS THAT ARE TO BE CORRECTED.
C     A(I,J)  -MATRIX OF WEIGHTS FOR ALL INPUTS I FEEDING INTO GATE J. (I(1
C              THRU NV): WEIGHTS FOR INDEPENDENT INPUTS X(1) THRU X(NV),
C              I(NV+1 THRU 2NV): WEIGHTS FOR INDEPENDENT INPUTS XN(1) THRU
C              XN(NV),I(2NV THRU 2NV+NG): WEIGHT FOR GATE I FEEDING INTO
C              GATE J.
C     VAR(I)  -VALUE OF INPUT I FEEDING INTO GATE J.(1 OR 0)
C     V(I)    -MINIMUM VALUE OF THE SEPARATING FUNCTION OF GATE J WITH ERR
C              ERRORS MADE IN THE I SET OF GATES, WHERE F(I)=1 AND F(J)=1.
C     P(I)    -MAXIMUM VALUE OF THE SEPARATING FUNCTION OF GATE J WITH ERR
C              ERRORS MADE IN THE I SET OF GATES, WHERE F(I)=0 AND F(J)=0.
C     CKT     -THE TEMPORARY VALUE OF THE CONSTRAINTS ERROR MULTIPLYING
C              FACTOR. ERRORS DUE TO THERE NOT BEING A GAP ARE MULTIPLIED
C              BY CKT.
C     CKP     -THE PREVIOUS VALUE OF THE CONSTRAINTS ERROR MULTIPLYING
```

```
C           FACTOR.
C  KSC      -A FLAG INDICATING THE SEARCH FOR THE MINIMUM K VALUES IS
C           COMPLETE.
C  KSUMT    -SUM OF KT VALUES AT ANY TIME.
C  ESG      -THE ERROR DUE TO THE GAP WITH ERR ERRORS NOT BEING
C           WITHIN THE GAP ESTABLISHED WHEN NO ERRORS ARE MADE.
C  EFGT     -THE ERROR DUE TO THERE NOT BEING A GAP FOR THE THRESHOLD
C           WHEN ERR ERRORS ARE MADE.
C  KEXP     -A FLAG INDICATING THAT THE SEARCH ROUTINE IS PERFORMING AN
C           EXPLORATORY MOVE.
C  KPMV     -A FLAG INDICATING THAT THE SEARCH ROUTINE IS PERFORMING A
C           PATTERN MOVE.
C  KPOS     -A FLAG INDICATING WHETHER A KT VALUE WAS INDEXED POSITIVE OR
C           NEGATIVE IN THE LAST STEP.
C  IWSG     -A FLAG INDICATING WHETHER ANY KT VALUE REDUCED THE COST
C           FUNCTION IN THE LAST EXPLORATORY MOVE.
C  KBSUM    -SUM OF THE BEST K VALUES FOUND.
C  UBUT(J)-THE UPPER LIMIT FOR THE THRESHOLD OF GATE J USING THE BEST
C           K VALUES.
C  LBUT(J)-THE LOWER LIMIT FOR THE THRESHOLD OF GATE J USING THE BEST
C           K VALUES.
C  KBUT(I)-THE MINIMUM NUMBER OF GATES IN SET I REQUIRED TO CORRECT ERR
C           ERRORS IN THE GIVEN REALIZATION.
C
C  NOTE: GATES MUST BE NUMBERED IN ASCENDING ORDER TO THE OUTPUT GATE.
C        NO GATE CAN FEED INTO A GATE WITH A LOWER NUMBER THAN ITS OWN
C        NUMBER.
C
C     THIS IS A PROGRAM WRITTEN TO MINIMIZE THE NUMBER OF THRESHOLD
C  GATES IN AN ERROR CORRECTING NETWORK, USING MULTIPLEXING TECHNIQUES.
C  THE REALIZATION IS FOUND USING THE OPTIMAL SEARCH TECHNIQUE, AND THE
C  COST FUNCTION IS THE NUMBER OF GATES REQUIRED PLUS AN ERROR FUNCTION
C  DUE TO CONSTRAINTS NOT BEING SATISFIED.
C
C
C                              PROGRAM LISTING
C
C     THE MAIN PROGRAM READS IN THE INPUT DATA AND SELECTS THE GATE J
C  WHICH MUST CORRECT THE SPECIFIED NUMBER OF ERRORS AT ANY POINT ON THE
C  N CUBE.  THE MAIN PROGRAM DETERMINES WHETHER THE ANSWER FURNISHED BY
C  THE SUBROUTINES SATISFIED THE CONSTRAINTS.  IF SO, IT PRINTS OUT THE
C  ANSWER.
C
C
C                              MAIN PROGRAM
C MAIN PROGRAM FOR MULTIPLEXING WITH THRESHOLD GATES USING METHOD 2
C
      DIMENSION KO(20),KBUT(20),UBUT(20),LBUT(20),GO(20)
      COMMON F(20),A(40,20),U(20),L(20),GAP(20),KT(20),KBP(20),Z(20),
     1X(20),V(20),W(20),X(10),XN(10),VAR(40),KU(20),UN(20),LN(20),NV,
     2NG,CKT,CSAT,ERR,CFT,CFP,KSC
      REAL L,LN,LBUT
      INTEGER F,X,XN
C  THE NEXT 17 CARDS READ IN THE INPUT DATA AND SET THE INITIAL OUTPUT
C  VALUES.
      READ(5,1) NG,NV
    1 FORMAT(2I5)
      KA=2*NV+NG
      NV2=2*NV
      READ(5,3) ((A(I1,J1),I1=1,KA),J1=1,NG)
    3 FORMAT(8F10.4)
      READ(5,4) (U(I2),I2=1,NG)
      READ(5,4) (L(I3),I3=1,NG)
```

```
    4 FORMAT(8F10.4)
    5 READ(5,6)ERR
    6 FORMAT(F10.2)
      DO 7 I4=1,NG
      KOUT(I4)=1
      UOUT(I4)=U(I4)
    7 LOUT(I4)=L(I4)
      READ(5,8) (KBP(I5),I5=1,NG)
    8 FORMAT(10I5)
      DO 9 I6=1,NG
      KO(I6)=KBP(I6)
    9 KT(I6)=KBP(I6)
C   THE NEXT 14 CARDS PRINT OUT THE INFORMATION READ IN.
      WRITE(6,10) NG,NV,ERR
   10 FORMAT(1HO,'NG=',I5,5X,'NV=',I5,5X,'NO. OF ERR=',F10.2)
      WRITE(6,11)
   11 FORMAT(1HO,' A MATRIX VALUES')
      WRITE(6,3)  ((A(I7,J2),I7=1,KA),J2=1,NG)
      WRITE(6,12)
   12 FORMAT(1HO,' U VALUES')
      WRITE(6,4) (U(I8),I8=1,NG)
      WRITE(6,13)
   13 FORMAT(1HO,' L VALUES')
      WRITE(6,4) (L(I9),I9=1,NG)
      WRITE(6,14)
   14 FORMAT(1HO,'KBP VALUES')
      WRITE(6,3) (KBP(J3),J3=1,NG)
C   THE DO LOOP TERMINATED BY STATEMENT 28 SELECTS THE GATE J THAT MUST
C      CORRECT THE ERRORS AT A PARTICULAR TIME.
      DO 28 J=1,NG
      CSAT=1.0E+15
      DO 15 J4=1,NG
      KU(J4)=KO(J4)
      KT(J4)=KO(J4)
   15 KBP(J4)=KO(J4)
      CKT=200.
      CKP=200.
      KSC=0
C   THE NEXT 11 CARDS SPECIFY THE VALUE OF CKT AND CALL SUBROUTINE CONST.
C      THEY ALSO CHECK TO SEE THAT ALL CONSTRAINTS WERE SATISFIED.  IF
C      CSAT WAS NOT 0, THEN CKT IS INCREASED AND CONST CALLED AGAIN.
      DO 25 J5=1,10
      IF(KSC.EQ.0) GO TO 16
      IF(CSAT.LE.0.) GO TO 26
   16 CKP=CKT
      CKT=CKT*5.
      CFT=1.0E+15
      CFP=1.0E+15
      KSC=0
      IW=1
      CALL CONST(J,IW,KA,NV2)
   25 CONTINUE
C   THE NEXT 7 CARDS FIND THE MAXIMUM K(I) REQUIRED BY ANY GATE J TO
C      CORRECT ERR ERRORS.  THESE CARDS ACCUMULATE THE OUTPUT INFORMATION.
   26 KBSUM=0
      DO 27 J6=1,NG
      IF(KBP(J6).GT.KOUT(J6)) KOUT(J6)=KBP(J6)
      KBSUM=KBSUM+KOUT(J6)
   27 CONTINUE
      IF(UN(J).LT.UOUT(J)) UOUT(J)=UN(J)
      IF(LN(J).GT.LOUT(J)) LOUT(J)=LN(J)
      GO(J)=UOUT(J)-LOUT(J)
```

```
   28 CONTINUE
C  THE NEXT 10 CARDS PRINT OUT THE FINAL VALUES FOR CORRECTING ERR ERRORS
C     IN THE REALIZATION.
   29 WRITE(6,30) CSAT,CKT
   30 FORMAT(1X,' CSAT=',F10.4,5X,' CKT=',F10.4)
      WRITE(6,31) (KOUT(L1),L1=1,NG)
   31 FORMAT(1HO,' FINAL K VALUES ARE',10X,10I5)
      WRITE(6,32) (UOUT(L2),L2=1,NG)
   32 FORMAT(1HO,' FINAL U VALUES ARE',10X,10F10.4)
      WRITE(6,33) (LOUT(L3),L3=1,NG)
   33 FORMAT(1HO,'FINAL L VALUES ARE',10X,10F10.4)
      WRITE(6,34) (GO(L4),L4=1,NG)
   34 FORMAT(/,'NEW GAPS ARE',10F10.5)
      WRITE(6,35) KOSUM
   35 FORMAT(/,'SUM OF FINAL K VALUES IS:',I5)
      STOP
      END
```

```
C     SUBROUTINE CONST DETERMINES WHETHER THE INPUTS TO GATE J SHOULD
C  BE 1 OR 0 AT EACH POINT ON THE N CUBE.  IT THEN CALCULATES THE MINI-
C  MUM W AND THE MAXIMUM Z VALUES FOR EACH GATE I FEEDING INTO GATE J.
C
C                    SUBROUTINE CONST
C SUBROUTINE CONST FOR CALCULATING VALUES FOR THE CONSTRAINTS
      SUBROUTINE CONST(J,IN,KA,NV2)
```

```
      DIMENSION IFL1(20),IFL2(20)
      COMMON F(20),A(40,20),U(20),L(20),GAP(20),KT(20),KBP(20),Z(20),
     1W(20),V(20),H(20),X(10),XN(10),VAR(40),KU(20),UN(20),LN(20),NV,
     2NG,CKT,CSAT,ERR,CFT,CFP,KSC
      REAL L,LN
      INTEGER F,X,XN
C  THE NEXT 6 CARDS SET INITIAL VALUES.
      DO 1 I1=1,NG
      V(I1)=U(J)
    1 W(I1)=L(J)
      DO 2 I2=1,NV
      X(I2)=0
    2 XN(I2)=0
C  THE DO LOOP TERMINATED BY STATEMENT 22 PERFORMS THE FUNCTION OF A
C     BINARY COUNTER INDEXING THE INDEPENDENT INPUTS THROUGH ALL POINTS
C     ON THE N CUBE.
      IPC=2**NV
      DO 22 I3=1,IPC
C  THE NEXT 30 CARDS ASSIGN THE PROPER VALUE TO INDEPENDENT INPUTS X
C     AND XN.
      IF(X(1).LE.1) GO TO 4
      X(1)=0
      X(2)=X(2)+1
      IF(X(2).LE.1) GO TO 4
      X(2)=0
      X(3)=X(3)+1
      IF(X(3).LE.1) GO TO 4
      X(3)=0
      X(4)=X(4)+1
      IF(X(4).LE.1) GO TO 4
      X(4)=0
      X(5)=X(5)+1
      IF(X(5).LE.1) GO TO 4
      X(5)=0
      X(6)=X(6)+1
      IF(X(6).LE.1) GO TO 4
      X(6)=0
      X(7)=X(7)+1
      IF(X(7).LE.1) GO TO 4
      X(7)=0
      X(8)=X(8)+1
      IF(X(8).LE.1) GO TO 4
      X(8)=0
      X(9)=X(9)+1
      IF(X(9).LE.1) GO TO 4
      X(9)=0
      X(10)=X(10)+1
    4 DO 5 I4=1,NV
      XN(I4)=1-X(I4)
    5 CONTINUE
C  THE NEXT 19 CARDS CALCULATE THE OUTPUT OF EACH GATE(1 OR 0), AND CON-
C     STRUCT THE MATRIX OF INPUT VARIABLES FOR EACH POINT ON THE N CUBE.
      DO 6 I5=1,KA
    6 VAR(I5)=0.
      DO 8 I6=1,NV
      IAA=I6+NV
      VAR(IAA)=XN(I6)
    7 VAR(I6)=X(I6)
    8 CONTINUE
      VAL=0.
      DO 12 I7=1,NG
      DO 9 J1=1,KA
```

```
    9 VAL=A(J1,I7)*VAR(J1)+VAL
      IF(VAL.LT.U(I7)) GO TO 10
      F(I7)=1
      GO TO 11
   10 F(I7)=0
   11 IAB=I7+2*NV
      VAR(IAB)=F(I7)
      VAL=0.
   12 CONTINUE
C  THE NEXT 25 CARDS CALCULATE THE MINIMUM W AND MAXIMUM Z FOR EACH GATE
C     I WHICH FEEDS INTO GATE J.
      DO 21 I8=1,NG
      IAC=2*NV+I8
      IF(A(IAC,J).LE.(.1)) GO TO 21
      IF(F(J).EQ.1.AND.F(I8).EQ.1) GO TO 16
      IF(F(J).EQ.1.OR.F(I8).EQ.1) GO TO 21
      SF2=0.
      DO 13 I9=1,KA
      SF2=A(I9,J)*VAR(I9)+SF2
   13 CONTINUE
      IF(IFL1(I8).EQ.1) GO TO 15
   14 IFL1(I8)=1
      Z(I8)=SF2
   15 IF(SF2.LE.Z(I8)) GO TO 21
      Z(I8)=SF2
      GO TO 21
   16 SF1=0.
      DO 17 J2=1,KA
      SF1=A(J2,J)*VAR(J2)+SF1
   17 CONTINUE
      IF(IFL2(I8).EQ.1) GO TO 19
   18 IFL2(I8)=1
      W(I8)=SF1
   19 IF(SF1.GE.W(I8)) GO TO 21
   20 W(I8)=SF1
   21 CONTINUE
      X(1)=X(1)+1
   22 CONTINUE
      DO 23 J3=1,NG
      IFL1(J3)=0
   23 IFL2(J3)=0
      CALL SEARCH(J,IN,KA,NV2)
C  THE NEXT 10 CARDS CALCULATE THE NEW UPPER AND LOWER LIMITS FOR THE
C     THRESHOLD OF GATE J IN ORDER TO CORRECT ERR ERRORS.
      UN(J)=U(J)
      LN(J)=L(J)
      DO 24 J4=1,NG
      MM=NV2+J4
      IF(A(MM,J).EQ.0.) GO TO 24
      V(J4)=W(J4)-(A(MM, J)*ERR)/KBP(J4)
      H(J4)=Z(J4)+(A(MM, J)*ERR)/KBP(J4)
      GAP(J)=UN(J)-LN(J)
      IF(V(J4).LT.UN(J)) UN(J)=V(J4)
      IF(H(J4).GT.LN(J)) LN(J)=H(J4)
   24 CONTINUE
      RETURN
      END
```

```
C      SUBROUTINE SEARCH DETERMINES THE V AND H VALUES FOR EACH GATE I
C   FEEDING INTO GATE J, USING THE GIVEN K VALUES, AND CALCULATES THE COST
C   FUNCTION.  IT THEN PERFORMS THE OPTIMAL SEARCH, CALCULATING THE COST
C   FUNCTION AFTER EACH STEP.
C
C                      SUBROUTINE SEARCH
      SUBROUTINE SEARCH(J,IN,KA,NV2)
      COMMON F(20),A(40,20),U(20),L(20),GAP(20),KT(20),KBP(20),Z(20),
     1A(20),V(20),H(20),X(10),XN(10),VAR(40),KU(20),UN(20),LN(20),NV,
     2NG,CKT,COAT,ERR,CFT,CFP,KSC
      REAL L,LN
C   THE NEXT 4 CARDS SET INITIAL VALUES.
      CFT=1.0E+15
      CFP=1.0E+15
      KBOS=0
      KBMP=0
    1 KSUMT=0
      DO 2 I1=1,NG
    2 KSUMT=KSUMT+KT(I1)
C   THE NEXT 15 CARDS CALCULATE V AND H VALUES, AND DETERMINE THE ERROR
C      DUE TO THE GAP WITH ERR ERRORS NOT BEING WITHIN THE GAP OF GATE J
C      WHEN NO ERRORS ARE MADE.
      EOG=0.
      DO 3 I2=1,J
      II=NV2+I2
      IF(A(II,J).LE.0.1) GO TO 3
      V(I2)=A(I2)-(A(II,J)*ERR)/KT(I2)
      H(I2)=Z(I2)+(A(II,J)*ERR)/KT(I2)
C   THE FOLLOWING 2 CARDS MUST BE MODIFIED IF A MINIMUM GAP IS TO BE
C   SPECIFIED.
      ELL=L(J)-V(I2)
```

```
      EUL=H(I2)-U(J)
      IF(ELL.EQ.0.0) ELL=1.0E-04
      IF(EUL.EQ.0.0) EUL=1.0E-04
      IF(ELL.LT.0.0) ELL=0.0
      IF(EUL.LT.0.0) EUL=0.0
      E0G=E0G+ELL+EUL
    3 CONTINUE
C   THE NEXT 13 CARDS CALCULATE THE ERROR DUE TO THERE NOT BEING A GAP,
C      IF ANY V VALUES ARE LESS THAN ANY H VALUE.
   26 EFGT=0.
      DO 4 I3=1,J
      IG=NV2+I3
      IF(A(IG,J).LE.0.1) GO TO 4
      DO 20 J1=1,J
      IH=NV2+J1
      IF(A(IH,J).LE.0.1) GO TO 20
C   THE FOLLOWING CARD MUST BE MODIFIED IF A MINIMUM GAP IS TO BE
C   SPECIFIED.
      EFG=(Z(I3)+(A(IG,J)*ERR)/KT(I3))-(W(J1)-(A(IH,J)*ERR)/KT(J1))
      IF(EFG.EQ.0.0) EFG=1.0E-04
      IF(EFG.LT.0.0) EFG=0.0
      EFGT=EFGT+EFG
   20 CONTINUE
    4 CONTINUE
C   THE NEXT 4 CARDS CALCULATE THE COST FUNCTION USING KT VALUES.
      E0G=E0G*1.0E+05
      EFGT=EFGT*CKT
      CSAT=E0G+EFGT
      CFT=CSAT+KSUMT
C   THE REMAINING STATEMENTS COMPRISE THE SEARCH TECHNIQUE FOR THE K
C      VALUES.
      IF(CFT.LT.CFP.AND.KPMP.EQ.1) GO TO 16
      IF(CFT.GE.CFP) GO TO 55
C   THE KU VALUES ARE SET EQUAL TO THE KT VALUES BECAUSE THE COST FUNC-
C      TION WAS REDUCED.
      DO 50 J4=1,NG
   50 KU(J4)=KT(J4)
      IF(KP0S.EQ.1) IW=IW+1
      KP0S=0
C   THE NEXT  4 CARDS START AN EXPLORATORY SEARCH IF ONE IS NOT ALREADY
C      IN PROGRESS.
   55 IF(KESP.NE.1) GO TO 5
      IF(IW.GT.NG) GO TO 12
    5 KESP=1
      KPMP=0
      IF(CFT.LT.CFP) GO TO 7
C   THE KT VALUES ARE RESTORED TO THE VALUE THEY HAD BEFORE THE STEP
C      BECAUSE THE COST FUNCTION INCREASED.
      DO 6 I4=1,NG
    6 KT(I4)=KU(I4)
      GO TO 8
C   THE BEST PREVIOUS COST FUNCTION IS SET EQUAL TO CFT BECAUSE THE SEARCH
C      WAS SUCCESSFUL.
    7 CFP=CFT
      IWS0=1
    8 IF(KP0S.EQ.1) GO TO 10
C   THE NEXT 2 CARDS INDEX KT(IW) POSITIVE BY 1.
    9 KT(IW)=KT(IW)+1
      KP0S=1
      GO TO 1
C   THE NEXT 4 CARDS INDEX THE KT(IW) NEGATIVE BY 1 AND INDEX IW FOR THE
C      NEXT PASS.
```

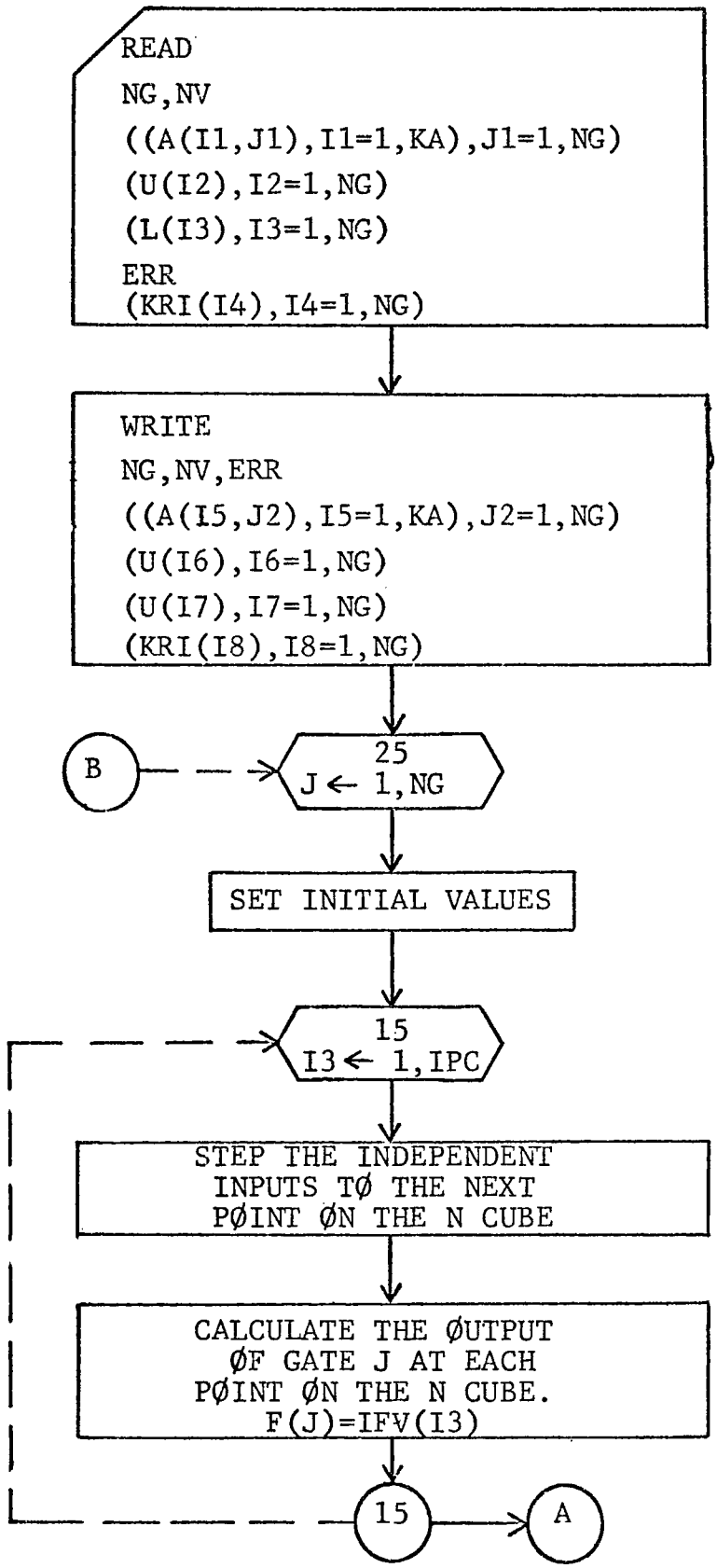
```
   10 KPWS=0
      IF(KT(IW).LE.1) GO TO 11
      KT(IW)=KT(IW)-1
      IW=IW+1
      GO TO 1
   11 KT(IW)=1
      IW=IW+1
      IF(IW.LE.NG) GO TO 9
   12 IW=1
      IF(IWS0.EQ.0) GO TO 13
      GO TO 14
   13 KSC=1
      RETURN
   14 IWS0=0
      KESP=0
      IF(CFT.LT.CFP) GO TO 16
C  THE NEXT 2 CARDS RESTORE THE KT VALUES TO THE VALUES THEY HAD BEFORE
C     THE LAST STEP, SINCE THE LAST STEP FAILED TO REDUCE THE COST
C     FUNCTION.
      DO 15 J2=1,NG
   15 KT(J2)=KU(J2)
C  THE NEXT 11 CARDS ACCOMPLISH THE PATTERN MOVE.
   16 DO 18 J3=1,NG
      IF(CFT.LT.CFP) CFP=CFT
      DEL=KT(J3)-KBP(J3)
      KBP(J3)=KT(J3)
      KU(J3)=KT(J3)
      IF(KT(J3).LE.1) GO TO 17
      KT(J3)=KT(J3)+DEL
      GO TO 18
   17 KT(J3)=1
   18 CONTINUE
      KPMP=1
      GO TO 1
      END
```

VARIABLE LIST FOR METHOD III PROGRAM

```
C                    PROGRAM DOCUMENTATION
C
C
C                              VARIABLE LIST
C  F(I)    -OUTPUT FUNCTION VALUE FOR GATE I.(1 OR 0)
C  U(J)    -UPPER LIMIT FOR THE THRESHOLD OF GATE J.
C  L(J)    -LOWER LIMIT FOR THE THRESHOLD OF GATE J.
C  UN(J)   -UPPER LIMIT FOR THE THRESHOLD OF GATE J NECESSARY TO CORRECT
C           ERR ERRORS.
C  LN(J)   -LOWER LIMIT FOR THE THRESHOLD OF GATE J NECESSARY TO CORRECT
C           ERR ERRORS.
C  GAP(J)  - ACTUAL GAP OF GATE J AT A PARTICULAR TIME.
C  KT(I)   -TEMPORARY K VALUES BEING USED IN AN ATTEMPT TO REDUCE THE
C           COST FUNCTION.
C  KBP(I)  -THE BEST K VALUES FOUND IN PRECEEDING ITERATIONS.(BASE POINT
C           K VALUES)
C  KU(I)   -TEMPORARY STORAGE FOR THE IMPROVED KT VALUES DURING AN
C           EXPLORATORY SEARCH, BEFORE COMPARING THE KT VALUES TO THE
C           KBP VALUES.
C  Z(I)    -THE MAXIMUM VALUE OF THE SEPARATING FUNCTION OF GATE J SUCH
C           THAT F(J)=1 AND F(I)=1, WHERE THE OUTPUT OF GATE I IS AN
C           INPUT TO GATE J.
C  W(I)    -THE MINIMUM VALUE OF THE SEPARATING FUNCTION OF GATE J SUCH
C           THAT F(J)=0 AND F(I)=0, WHERE THE OUTPUT OF GATE I IS AN
C           INPUT TO GATE J.
C  X(I)    -THE VALUE OF THE INDEPENDENT INPUT X(I) AT A PARTICULAR POINT
C           ON THE N CUBE.(1 OR 0)
C  XN(I)   -THE VALUE OF THE INDEPENDENT INPUT XNOT(I) AT A PARTICULAR
C           POINT ON THE N CUBE.(1 OR 0)
C  NG      -NO OF GATES IN THE REALIZATION.
C  NV      -NO OF INDEPENDENT VARIABLES IN THE REALIZATION.
C  ERR     -NO OF ERRORS THAT ARE TO BE CORRECTED.
C  A(I,J)  -MATRIX OF WEIGHTS FOR ALL INPUTS I FEEDING INTO GATE J. (I(1
C           THRU NV): WEIGHTS FOR INDEPENDENT INPUTS X(1) THRU X(NV),
C           I(NV+1 THRU 2NV): WEIGHTS FOR INDEPENDENT INPUTS XN(1) THRU
C           XN(NV),I(2NV THRU 2NV+NG): WEIGHT FOR GATE I FEEDING INTO
C           GATE J.
C  VAR(I)  -VALUE OF INPUT I FEEDING INTO GATE J.(1 OR 0)
C  V(I)    -MINIMUM VALUE OF THE SEPARATING FUNCTION OF GATE J WITH ERR
C           ERRORS MADE IN THE I SET OF GATES, WHERE F(I)=1 AND F(J)=1.
C  H(I)    -MAXIMUM VALUE OF THE SEPARATING FUNCTION OF GATE J WITH ERR
C           ERRORS MADE IN THE I SET OF GATES, WHERE F(I)=0 AND F(J)=0.
C  CKT     -THE TEMPORARY VALUE OF THE CONSTRAINTS ERROR MULTIPLYING
C           FACTOR. ERRORS DUE TO THERE NOT BEING A GAP ARE MULTIPLIED
C           BY CKT.
C  CKP     -THE PREVIOUS VALUE OF THE CONSTRAINTS ERROR MULTIPLYING
C           FACTOR.
```

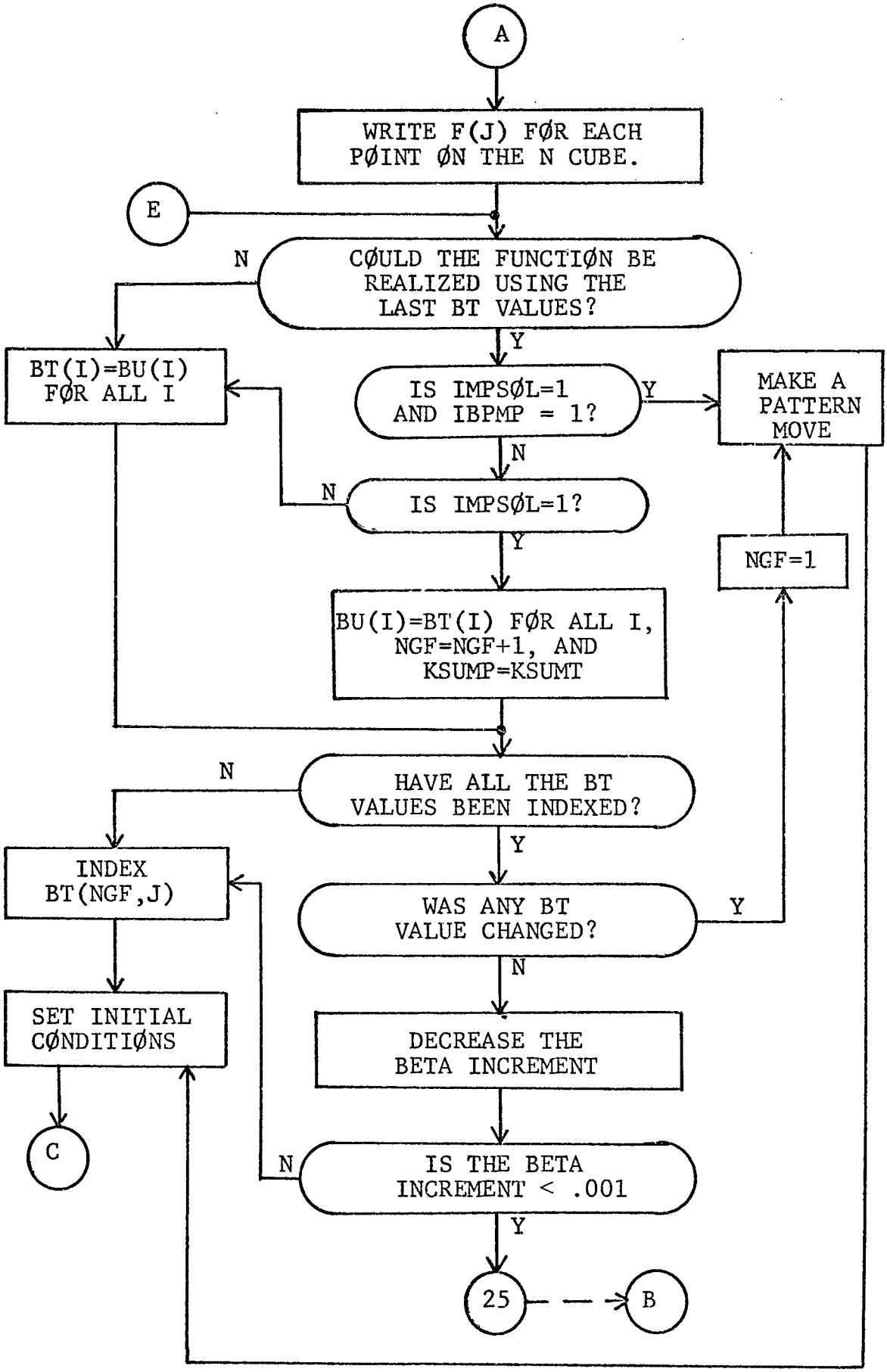
```
C  KSC     =A FLAG INDICATING THE SEARCH FOR THE MINIMUM K VALUES IS
C           COMPLETE.
C  KSUMT   =SUM OF KT VALUES AT ANY TIME.
C  EOG     =THE ERROR DUE TO THE GAP WITH ERR ERRORS NOT BEING
C           WITHIN THE GAP ESTABLISHED WHEN NO ERRORS ARE MADE.
C  EFGT    =THE ERROR DUE TO THERE NOT BEING A GAP FOR THE THRESHOLD
C           WHEN ERR ERRORS ARE MADE.
C  KESP    =A FLAG INDICATING THAT THE SEARCH ROUTINE IS PERFORMING AN
C           EXPLORATORY MOVE.
C  KPMP    =A FLAG INDICATING THAT THE SEARCH ROUTINE IS PERFORMING A
C           PATTERN MOVE.
C  KPOS    =A FLAG INDICATING WHETHER A KT VALUE WAS INDEXED POSITIVE OR
C           NEGATIVE IN THE LAST STEP.
C  IWSO    =A FLAG INDICATING WHEATHER ANY KT VALUE REDUCED THE COST
C           FUNCTION IN THE LAST EXPLORATORY MOVE.
C  KOSUM   =SUM OF THE BEST K VALUES FOUND.
C  UOUT(J)=THE UPPER LIMIT FOR THE THERSHOLD OF GATE J USING THE BEST
C           K VALUES.
C  LOUT(J)=THE LOWER LIMIT FOR THE THRESHOLD OF GATE J USING THE BEST
C           K VALUES.
C  KOUT(I)=THE MINIMUM NUMBER OF GATES IN SET I REQUIRED TO CORRECT ERR
C           ERRORS IN THE GIVEN REALIZATION.
C  AM(I,J)=SAME AS A(I,J) EXCEPT THAT IT CONTAINS THE INDEXED VALUES FOR
C           THE WEIGHT OF THE GATES I FEEDING INTO GATE J.
C  UNE(J) =UPPER LIMIT FOR THE THRESHOLD OF GATE J AFTER THE BT VALUES
C           HAVE BEEN CHANGED, AND WITHOUT ANY ERRORS MADE.
C  LNE(J) =LOWER LIMIT FOR THE THRESHOLD OF GATE J AFTER THE BT VALUES
C           HAVE BEEN CHANGED, AND WITHOUT ANY ERRORS MADE.
C  GNE(J) =ACTUAL GAP OF GATE J WITH NO ERRORS CORRECTED AFTER THE BT
C           VALUES HAVE BEEN CHANGED.
C BBP(I,J)=BASE POINT VALUES FOR THE WEIGHT OF GATE I FEEDING INTO GATE J.
C  BT(I,J)=TEMPORARY VALUES FOR THE WEIGHT OF GATE I FEEDING INTO GATE J.
C  BU(I,J)=TEMPORARY STORAGE OF THE BT VALUES DURING AN EXPLORATORY BETA
C           SEARCH.
C  SIG     =THE VALUE OF THE BETA INCREMENT AT ANY TIME.
C  NGF     =THE INPUT BETA VALUE FEEDING INTO GATE J THAT IS BEING INDEXED
C           AT THE TIME.
C  KSUMP   =THE BEST PREVIOUS SUM OF K VALUES.
C  IFV(N)  =MATRIX OF THE INITIAL FUNCTION VALUES OF THE GATE J AT EACH
C           POINT OF THE N CUBE.
C  CFNR    =A FLAG INDICATING THE PROPER OUTPUT FUNCTION IS NOT REA-
C           LIZED FOR EACH POINT ON THE N CUBE.
C  KRI(I)  =THE K VALUES READ IN.
C  IBV     =A FLAG INDICATING THAT THE REALIZATION IS BEING USED JUST AS
C           IT WAS READ IN.
C  IMPSOL  =A FLAG INDICATING THAT THE REALIZATION CALCULATED IS AN
C           IMPROVED SOLUTION.
C  ILLSOL  =A FLAG INDICATING THAT THE ORIGINAL OUTPUT FUNCTION IS NOT
C           REALIZED OR THE CONSTRAINTS WERE NOT SATISFIED WHEN ERRORS
C           WERE MADE.
C
C  NOTE: GATES MUST BE NUMBERED IN ASCENDING ORDER TO THE OUTPUT GATE.
C        NO GATE CAN FEED INTO A GATE WITH A LOWER NUMBER THAN ITS OWN
C        NUMBER.
```

FLØW CHART FØR THE METHØD 3 PRØGRAM

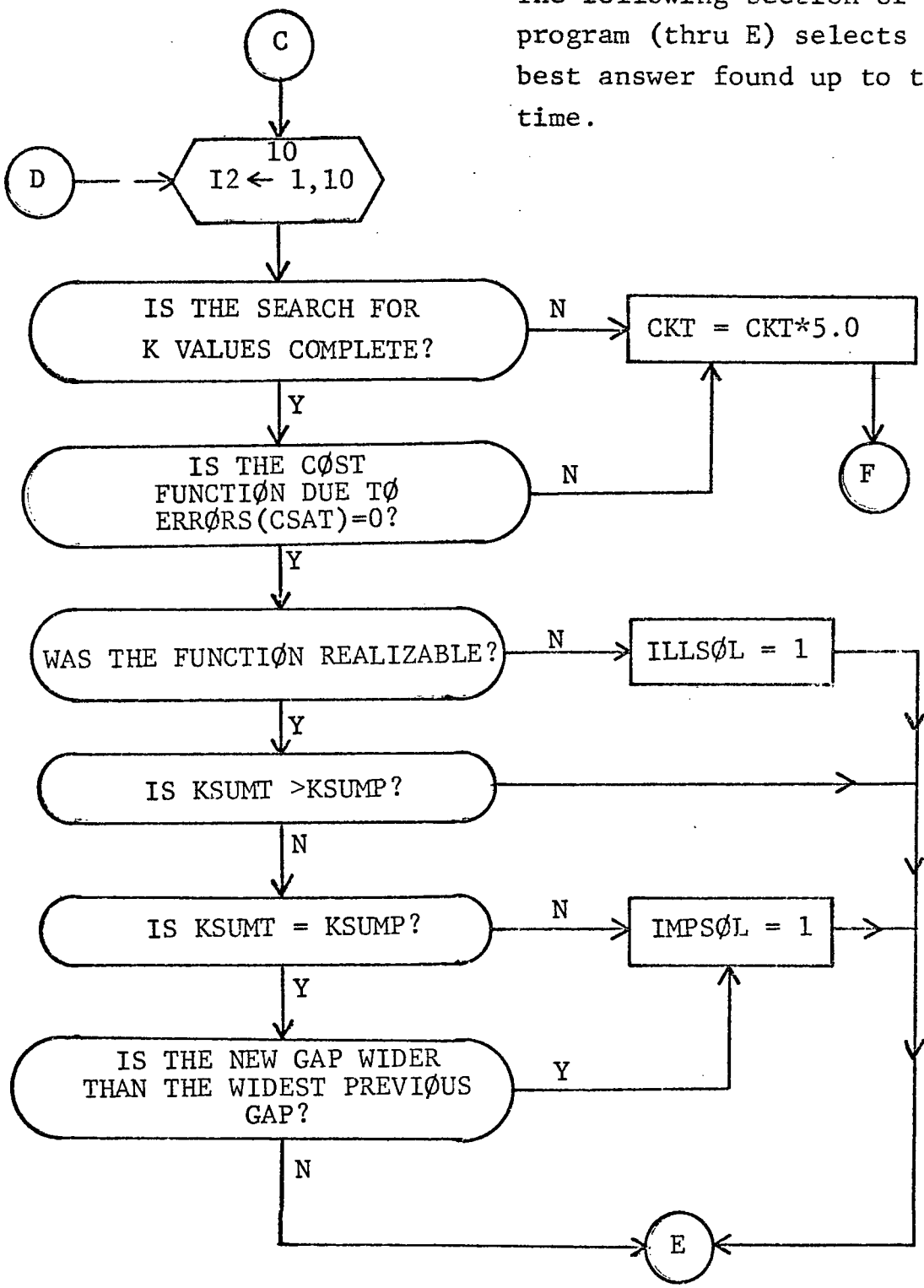


The next section of the program (thru A) calculate the output A_j should have for each point on the n cube.

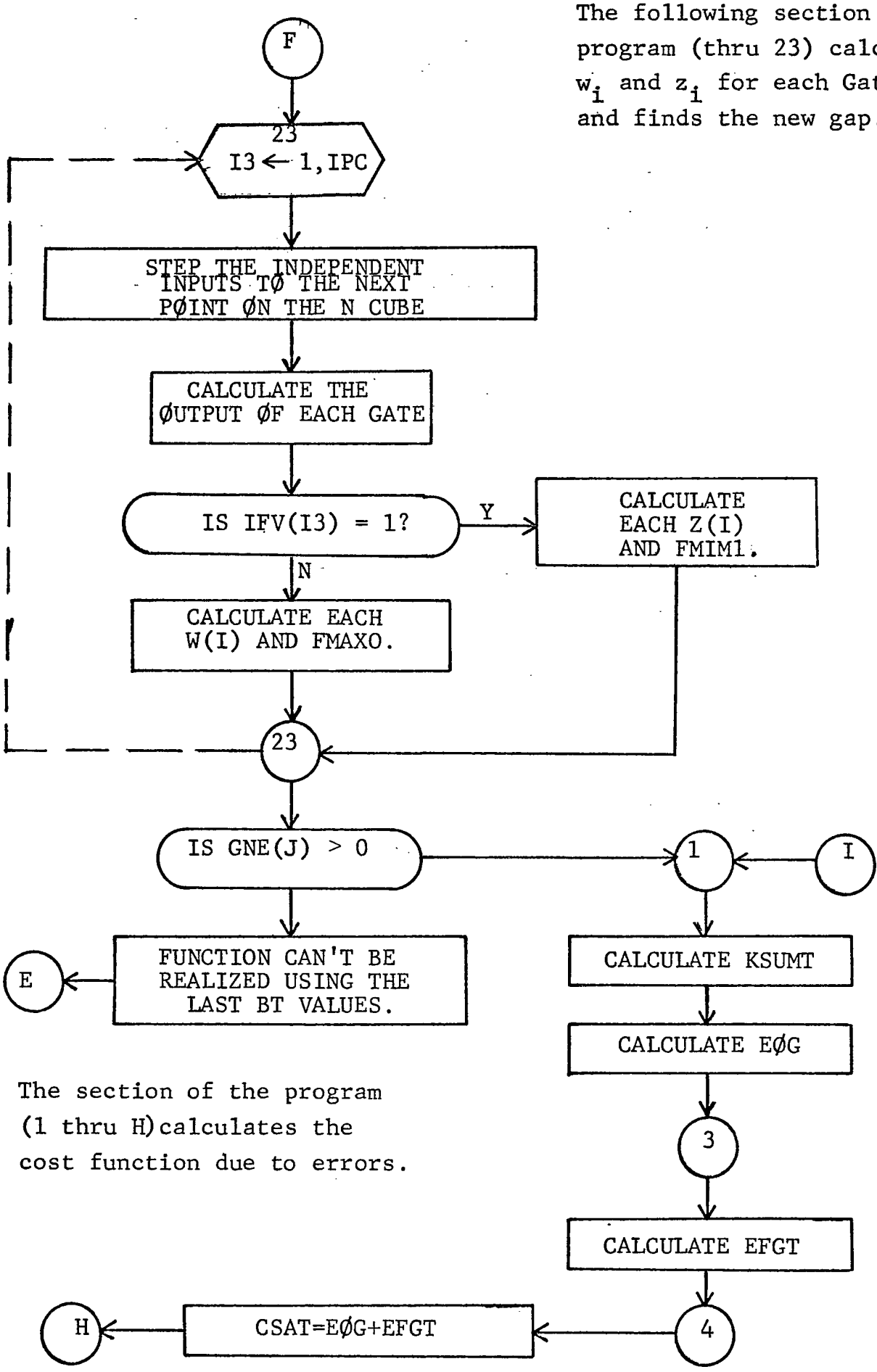
The following section of the program performs the search of the β_{ij} values.



The following section of the program (thru E) selects the best answer found up to that time.

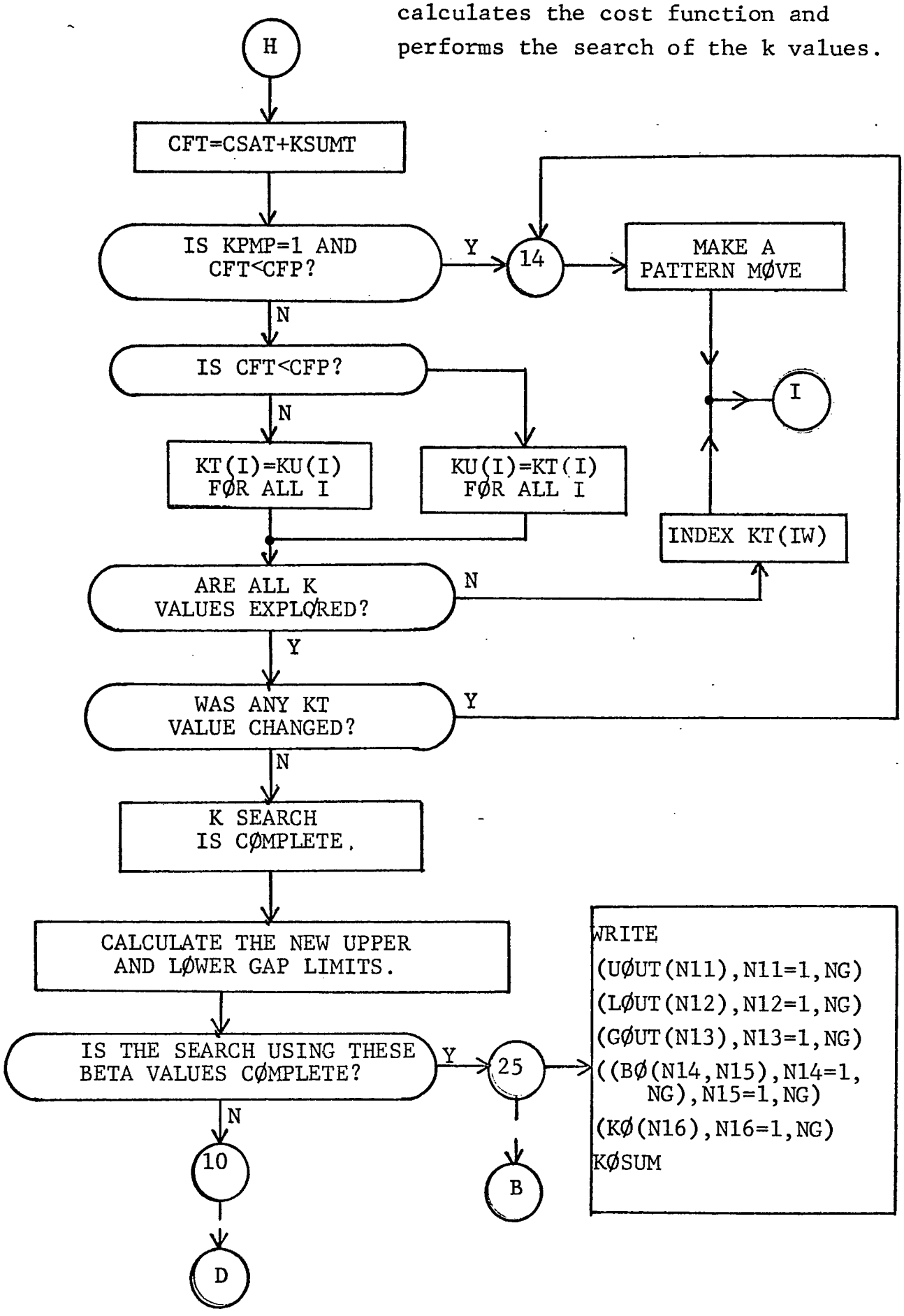


The following section of the program (thru 23) calculate w_i and z_i for each Gate A_i, and finds the new gap.



The section of the program (1 thru H) calculates the cost function due to errors.

The following section of the program calculates the cost function and performs the search of the k values.



METHOD III PROGRAM

```
C                    PROGRAM DOCUMENTATION
C
C
C
C                              VARIABLE LIST
C   F(I)    -OUTPUT FUNCTION VALUE FOR GATE I.(1 OR 0)
C   U(J)    -UPPER LIMIT FOR THE THRESHOLD OF GATE J.
C   L(J)    -LOWER LIMIT FOR THE THRESHOLD OF GATE J.
C   UN(J)   -UPPER LIMIT FOR THE THRESHOLD OF GATE J NECESSARY TO CORRECT
C            ERR ERRORS.
C   LN(J)   -LOWER LIMIT FOR THE THRESHOLD OF GATE J NECESSARY TO CORRECT
C            ERR ERRORS.
C   GAP(J)  - ACTUAL GAP OF GATE J AT A PARTICULAR TIME.
C   KT(I)   -TEMPORARY K VALUES BEING USED IN AN ATTEMPT TO REDUCE THE
C            COST FUNCTION.
C   KBP(I)  -THE BEST K VALUES FOUND IN PRECEEDING ITERATIONS.(BASE POINT
C            K VALUES)
C   KU(I)   -TEMPORARY STORAGE FOR THE IMPROVED KT VALUES DURING AN
C            EXPLORATORY SEARCH, BEFORE COMPARING THE KT VALUES TO THE
C            KBP VALUES.
C   Z(I)    -THE MAXIMUM VALUE OF THE SEPARATING FUNCTION OF GATE J SUCH
C            THAT F(J)=1 AND F(I)=1, WHERE THE OUTPUT OF GATE I IS AN
C            INPUT TO GATE J.
C   W(I)    -THE MINIMUM VALUE OF THE SEPARATING FUNCTION OF GATE J SUCH
C            THAT F(J)=0 AND F(I)=0, WHERE THE OUTPUT OF GATE I IS AN
C            INPUT TO GATE J.
C   X(I)    -THE VALUE OF THE INDEPENDENT INPUT X(I) AT A PARTICULAR POINT
C            ON THE N CUBE.(1 OR 0)
C   XN(I)   -THE VALUE OF THE INDEPENDENT INPUT XNOT(I) AT A PARTICULAR
C            POINT ON THE N CUBE.(1 OR 0)
C   NG      -NO OF GATES IN THE REALIZATION.
C   NV      -NO OF INDEPENDENT VARIABLES IN THE REALIZATION.
C   ERR     -NO OF ERRORS THAT ARE TO BE CORRECTED.
C   A(I,J)  -MATRIX OF WEIGHTS FOR ALL INPUTS I FEEDING INTO GATE J. (I(1
C            THRU NV): WEIGHTS FOR INDEPENDENT INPUTS X(1) THRU X(NV),
C            I(NV+1 THRU 2NV): WEIGHTS FOR INDEPENDENT INPUTS XN(1) THRU
C            XN(NV),I(2NV THRU 2NV+NG): WEIGHT FOR GATE I FEEDING INTO
C            GATE J.
C   VAR(I)  -VALUE OF INPUT I FEEDING INTO GATE J.(1 OR 0)
C   V(I)    -MINIMUM VALUE OF THE SEPARATING FUNCTION OF GATE J WITH ERR
C            ERRORS MADE IN THE I SET OF GATES, WHERE F(I)=1 AND F(J)=1.
C   H(I)    -MAXIMUM VALUE OF THE SEPARATING FUNCTION OF GATE J WITH ERR
C            ERRORS MADE IN THE I SET OF GATES, WHERE F(I)=0 AND F(J)=0.
C   CKT     -THE TEMPORARY VALUE OF THE CONSTRAINTS ERROR MULTIPLYING
C            FACTOR. ERRORS DUE TO THERE NOT BEING A GAP ARE MULTIPLIED
C            BY CKT.
C   CKP     -THE PREVIOUS VALUE OF THE CONSTRAINTS ERROR MULTIPLYING
C            FACTOR.
```

```
C  KSC     -A FLAG INDICATING THE SEARCH FOR THE MINIMUM K VALUES IS
C           COMPLETE.
C  KSUMT   -SUM OF KT VALUES AT ANY TIME.
C  EOG     -THE ERROR DUE TO THE GAP WITH ERR ERRORS NOT BEING
C           WITHIN THE GAP ESTABLISHED WHEN NO ERRORS ARE MADE.
C  EFGT    -THE ERROR DUE TO THERE NOT BEING A GAP FOR THE THRESHOLD
C           WHEN ERR ERRORS ARE MADE.
C  KESP    -A FLAG INDICATING THAT THE SEARCH ROUTINE IS PERFORMING AN
C           EXPLORATORY MOVE.
C  KPMP    -A FLAG INDICATING THAT THE SEARCH ROUTINE IS PERFORMING A
C           PATTERN MOVE.
C  KPOS    -A FLAG INDICATING WHETHER A KT VALUE WAS INDEXED POSITIVE OR
C           NEGATIVE IN THE LAST STEP.
C  IWSO    -A FLAG INDICATING WHEATHER ANY KT VALUE REDUCED THE COST
C           FUNCTION IN THE LAST EXPLORATORY MOVE.
C  KOSUM   -SUM OF THE BEST K VALUES FOUND.
C  UOUT(J)-THE UPPER LIMIT FOR THE THERSHOLD OF GATE J USING THE BEST
C           K VALUES.
C  LOUT(J)-THE LOWER LIMIT FOR THE THRESHOLD OF GATE J USING THE BEST
C           K VALUES.
C  KOUT(I)-THE MINIMUM NUMBER OF GATES IN SET I REQUIRED TO CORRECT ERR
C           ERRORS IN THE GIVEN REALIZATION.
C  AM(I,J)-SAME AS A(I,J) EXCEPT THAT IT CONTAINS THE INDEXED VALUES FOR
C           THE WEIGHT OF THE GATES I FEEDING INTO GATE J.
C  UNE(J) -UPPER LIMIT FOR THE THRESHOLD OF GATE J AFTER THE BT VALUES
C           HAVE BEEN CHANGED, AND WITHOUT ANY ERRORS MADE.
C  LNE(J) -LOWER LIMIT FOR THE THRESHOLD OF GATE J AFTER THE BT VALUES
C           HAVE BEEN CHANGED, AND WITHOUT ANY ERRORS MADE.
C  GNE(J) -ACTUAL GAP OF GATE J WITH NO ERRORS CORRECTED AFTER THE BT
C           VALUES HAVE BEEN CHANGED.
C BBP(I,J)-BASE POINT VALUES FOR THE WEIGHT OF GATE I FEEDING INTO GATE J.
C  BT(I,J)-TEMPORARY VALUES FOR THE WEIGHT OF GATE I FEEDING INTO GATE J.
C  BU(I,J)-TEMPORARY STORAGE OF THE BT VALUES DURING AN EXPLORATORY BETA
C           SEARCH.
C  SIG     -THE VALUE OF THE BETA INCREMENT AT ANY TIME.
C  NGF     -THE INPUT BETA VALUE FEEDING INTO GATE J THAT IS BEING INDEXED
C           AT THE TIME.
C  KSUMP   -THE BEST PREVIOUS SUM OF K VALUES.
C  IFV(N)  -MATRIX OF THE INITIAL FUNCTION VALUES OF THE GATE J AT EACH
C           POINT OF THE N CUBE.
C  CFNR    -A FLAG INDICATING THE PROPER OUTPUT FUNCTION IS NOT REA-
C           LIZED FOR EACH POINT ON THE N CUBE.
C  KPI(I)  -THE K VALUES READ IN.
C  IBV     -A FLAG INDICATING THAT THE REALIZATION IS BEING USED JUST AS
C           IT WAS READ IN.
C  IMPSBL  -A FLAG INDICATING THAT THE REALIZATION CALCULATED IS AN
C           IMPROVED SOLUTION.
C  ILLSBL  -A FLAG INDICATING THAT THE ORIGINAL OUTPUT FUNCTION IS NOT
C           REALIZED OR THE CONSTRAINTS WERE NOT SATISFIED WHEN ERRORS
C           WERE MADE.
C
C  NOTE: GATES MUST BE NUMBERED IN ASCENDING ORDER TO THE OUTPUT GATE.
C        NO GATE CAN FEED INTO A GATE WITH A LOWER NUMBER THAN ITS OWN
C        NUMBER.
C
C     THE MAIN PROGRAM READS IN THE INPUT DATA AND SELECTS THE GATE J
C  WHICH MUST CORRECT ERR ERRORS AT ANY POINT ON THE N CUBE.  THE MAIN
C  PROGRAM ALSO DETERMINES WHETHER THE ANSWER OBTAINED BY THE SUBROU-
C  TINES IS THE BEST ANSWER OBTAINED UP TO THAT TIME, AND CONTINUES TO
C  CALL THE SUBROUTINES TO TRY AND FIND A BETTER ANSWER.  AFTER ALL GATES
C  J HAVE BEEN USED TO CORRECT ERRORS, THE FINAL RESULTS ARE PRINTED OUT.
C
```

```
C                              MAIN PROGRAM
C MAIN PROGRAM FOR MULTIPLEXING USING METHOD 3
C
      DIMENSION BB(20,20),KB(20),UB(20),LB(20),KD(20),GB(20)
      COMMON AM(40,20),A(40,20),U(20),L(20),GAP(20),UNE(20),LNE(20),
     1GNE(20),UN(20),LN(20),Z(20),W(20),V(20),H(20),X(10),XN(10),EMAX
     2(20,20),VAR(40),BBP(20,20),BT(20,20),BU(20,20),NG,NV,CKT,CKP,CSAT,
     3SIG,NGF,KSJMP,KSUMT,ERR,IFV(1200),CFNR,KA,NV2,KT(20),KRI(20),
     4KBP(20),J,IPC,IBV,KU(20)
      REAL L,LN,LTEST,LB,LNE
      INTEGER X,XN
C  THE NEXT 14 CARDS READ IN THE INPUT DATA.
      READ(5,1) NG,NV
    1 FORMAT(2I5)
      KA=2*NV+NG
      NV2=2*NV
      IPC=2**NV
      READ(5,2) ((A(I1,J1),I1=1,KA),J1=1,NG)
    2 FORMAT(8F10.4)
      READ(5,3) (U(I2),I2=1,NG)
      READ(5,3) (L(I3),I3=1,NG)
    3 FORMAT(8F10.2)
    4 READ(5,5) ERR
    5 FORMAT(F10.2)
      READ(5,6) (KRI(I4),I4=1,NG)
    6 FORMAT(10I5)
C  THE NEXT 13 CARDS WRITE OUT THE INPUT DATA THAT WAS READ.
      WRITE(6,7) NG,NV,ERR
    7 FORMAT(1H0,' NG=',I5,5X,'NV=',I5,5X,'NO. OF ERR.=',F10.2)
      WRITE(6,8)
    8 FORMAT(1H0,' A MATRIX VALUES')
      WRITE(6,2) ((A(I5,J2),I5=1,KA),J2=1,NG)
      WRITE(6,9)
    9 FORMAT(1H0,' U VALUES')
      WRITE(6,3) (U(I6),I6=1,NG)
      WRITE(6,10)
   10 FORMAT(1H0,' L VALUES')
      WRITE(6,3) (L(I7),I7=1,NG)
      WRITE(6,11) (KRI(I8),I8=1,NG)
   11 FORMAT(1H0,' KRI VALUES',5X,10I5)
C  THE NEXT 23 CARDS SET CERTAIN INITIAL CONDITIONS.
      DO 12 J6=1,NG
      KD(J6)=1
   12 KB(J6)=1
      DO 25 J=1,NG
      DO 50 I8=1,J
      DO 50 J8=1,J
      IAD=J8+NV2
      BBP(J8,I8)=A(IAD,I8)
      BT(J8,I8)=BBP(J8,I8)
      BU(J8,I8)=BBP(J8,I8)
   50 CONTINUE
      DO 51 J8=1,NG
      KBP(J8)=KRI(J8)
      DO 51 K3=1,KA
      AM(K3,J8)=A(K3,J8)
   51 CONTINUE
      SIG=0.5
      NGF=1
      IBV=1
      KSUMP=1000
      NBC=0
```

```
      CALL OUTFUN
      GO TO 16
   14 CALL BETA(MBC,ILLSOL,IMPSOL)
   15 IF(MBC.EQ.1) GO TO 21
   16 CALL CONTRL
C  THE NEXT 26 CARDS PICK THE BEST RESULTS FROM ALL THE TRIALS MADE.
      IF(CSAT.EQ.0..AND.CFNR.EQ.0.) GO TO 17
      ILLSOL=1
      GO TO 14
   17 IF(KSUMT.GT.KSUMP) GO TO 14
      IF(KSUMT.EQ.KSUMP) GO TO 19
      UO(J)=UN(J)
      LO(J)=LN(J)
      GO(J)=UO(J)-LO(J)
      IMPSOL=1
      DO 18 J4=1,NG
      KO(J4)=KBP(J4)
   18 BO(J4,J)=BT(J4,J)
      GO TO 14
   19 UTEST=UN(J)
      LTEST=LN(J)
      GTEST=UTEST-LTEST
      GO(J)=UO(J)-LO(J)
      IF(GTEST.LE.GO(J)) GO TO 14
      UO(J)=UTEST
      LO(J)=LTEST
      GO(J)=UO(J)-LO(J)
      DO 20 J5=1,NG
      KO(J5)=KBP(J5)
   20 BO(J5,J)=BT(J5,J)
      IMPSOL=1
      GO TO 14
C  THE NEXT 5 CARDS DETERMINE THE PROPER OUTPUT DATA.
   21 KOSUM=0
      DO 24 J7=1,NG
      IF(KO(J7).GT.KO(J7)) KO(J7)=KO(J7)
      KOSUM=KOSUM+KO(J7)
   24 CONTINUE
   25 CONTINUE
C  THE NEXT 14 CARDS PRINT OUT THE FINAL OUTPUT VALUES.
   30 WRITE(6,31)
   31 FORMAT(1HO,30X,' FINAL VALUES')
      WRITE(6,32) (UO(N11),N11=1,NG)
   32 FORMAT(/,'NEW U VALUES',10F10.5)
      WRITE(6,33) (LO(N12),N12=1,NG)
   33 FORMAT(/,'NEW L VALUES',10F10.5)
      WRITE(6,34) (GO(N13),N13=1,NG)
   34 FORMAT(/,'NEW GAPS ARE',10F10.5)
      WRITE(6,35) ((BO(N14,N15),N14=1,NG),N15=1,NG)
   35 FORMAT(/,'NEW BETA VALUES ARE',10F10.5)
      WRITE(6,36) (KO(N16),N16=1,NG)
   36 FORMAT(/,'FINAL K VALUES ARE:',10I5)
      WRITE(6,37) KOSUM
   37 FORMAT(/,'SUM OF FINAL K VALUES IS:',I5)
      STOP
      END
```

```
C     SUBROUTINE OUTFUN CALCULATES THE CORRECT OUTPUT FOR GATE J AT EACH
C  POINT ON THE N CUBE AND STORES THE INFORMATION FOR COMPARISON LATER.
C
C                    SUBROUTINE  OUTFUN
C
      SUBROUTINE OUTFUN
      DIMENSION F(20)
      COMMON AM(40,20),A(40,20),U(20),L(20),GAP(20),UNE(20),LNE(20),
     1GNE(20),UN(20),LN(20),Z(20),W(20),V(20),H(20),X(10),XN(10),EMAX
     2(20,20),VAR(40),BBP(20,20),BT(20,20),BU(20,20),NG,NV,CKT,CKP,CSAT,
     3SIG,NGF,KSUMP,KSUMT,ERR,IFV(1200),CFNR,KA,NV2,KT(20),KRI(20),
     4KBP(20),J,IPC,IBV,KU(20)
      REAL L,LN,LNE
      INTEGER F,X,XN
C  THE NEXT 3 CARDS SET INITIAL VALUES.
      DO 2 I2=1,NV
      X(I2)=0
    2 XN(I2)=0
C  THE NEXT 32 CARDS SET THE VALUES OF THE INDEPENDENT INPUTS X AND XN
C     TO THE PROPER VALUES FOR EACH POINT ON THE N CUBE.
      DO 15 I3=1,IPC
      IF(X(1).LE.1) GO TO 4
      X(1)=0
      X(2)=X(2)+1
      IF(X(2).LE.1) GO TO 4
      X(2)=0
```

```
      X(3)=X(3)+1
      IF(X(3).LE.1) GO TO 4
      X(3)=0
      X(4)=X(4)+1
      IF(X(4).LE.1) GO TO 4
      X(4)=0
      X(5)=X(5)+1
      IF(X(5).LE.1) GO TO 4
      X(5)=0
      X(6)=X(6)+1
      IF(X(6).LE.1) GO TO 4
      X(6)=0
      X(7)=X(7)+1
      IF(X(7).LE.1) GO TO 4
      X(7)=0
      X(8)=X(8)+1
      IF(X(8).LE.1) GO TO 4
      X(8)=0
      X(9)=X(9)+1
      IF(X(9).LE.1) GO TO 4
      X(9)=0
      X(10)=X(10)+1
    4 DO 5 I4=1,NV
      XN(I4)=1-X(I4)
    5 CONTINUE
C   THE NEXT 19 CARDS CALCULATE THE OUTPUT FUNCTION OF EACH GATE AND
C      CONSTRUCT THE VARIABLE MATRIX AT EACH POINT ON THE N CUBE.
      DO 6 I5=1,KA
    6 VAR(I5)=0.
      DO 8 I6=1,NV
      IAA=I6+NV
      VAR(IAA)=XN(I6)
    7 VAR(I6)=X(I6)
    8 CONTINUE
      VAL=0.
      DO 12 I7=1,J
      DO 9 J1=1,KA
    9 VAL=A(J1,I7)*VAR(J1)+VAL
      IF(VAL.LT.U(I7)) GO TO 10
      F(I7)=1
      GO TO 11
   10 F(I7)=0
   11 IAB=I7+2*NV
      VAR(IAB)=F(I7)
      VAL=0.
   12 CONTINUE
C   THE INITIAL FUNCTION VALUE OF F(J) IS CALCULATED AT EACH POINT ON
C      THE N CUBE FOR GATE J.
      IFV(I8)=F(J)
      X(1)=X(1)+1
   15 CONTINUE
      RETURN
      END
```

```
C     SUBROUTINE BETA SEARCHES THE BETA VALUES (WEIGHTS FROM THE OUTPUT
C   OF ONE GATE TO THE INPUT OF THE OTHER) TO TRY AND FIND A REALIZATION
C   REQUIRING FEWER GATES.
C
C                    SUBROUTINE BETA
      SUBROUTINE BETA(MBC,ILLSAL,IMPSAL)
      COMMON AM(40,20),A(40,20),U(20),L(20),GAP(20),UNE(20),LNE(20),
     1GNE(20),UN(20),LN(20),Z(20),W(20),V(20),H(20),X(10),XN(10),EMAX
     2(20,20),VAR(40),BBP(20,20),BT(20,20),BU(20,20),NG,NV,CKT,CKP,CSAT,
     3SIG,NGF,KSUMP,KSUMT,ERR,IFV(1200),CFNR,KA,NV2,KT(20),KRI(20),
     4KBP(20),IG,IPC,IBV,KU(20)
      REAL L,LN,LNE
      INTEGER X,XN
      IF(ILLSAL.EQ.1) GO TO 5
    1 IF(IMPSAL.EQ.1.AND.IBPMP.EQ.1) GO TO 16
C   IF AN EXPLORATORY SEARCH IS NOT IN PROGRESS ONE SHOULD BE STARTED.
C   SINCE THE PREVIOUS BETA STEP WAS UNSUCCESSFUL THE BEST PREVIOUS VALUES
C     ARE RESTORED.
      IF(IBESP.NE.1) GO TO 2
      IF(NGF.GE.IG) GO TO 13
    2 IBESP=1
      IBPMP=0
    3 IF(BT(NGF,IG).GT.0.) GO TO 4
      BT(NGF,IG)=0.
      NGF=NGF+1
      IF(NGF.GE.IG) GO TO 13
      GO TO 3
    4 IF(IMPSAL.EQ.1) GO TO 7
    5 DO 6 I1=1,NG
    6 BT(I1,IG)=BU(I1,IG)
      IF(BT(NGF,IG).LE.0.) GO TO 3
      GO TO 9
C   THE NEXT 6 CARDS CHANGE THE BU VALUES TO THOSE OF BT, SINCE THE SEARCH
C     WAS SUCCESSFUL.  THEY ALSO SET INITIAL CONDITIONS AGAIN.
    7 KSUMP=KSUMT
```

```
      DO 8 J2=1,NG
    8 BU(J2,IG)=BT(J2,IG)
      IF(IBPOS.EQ.0) GO TO 30
      IBPOS=0
      NGF=NGF+1
   30 IF(IBV.EQ.0) ICOB=1
    9 IF(IBPOS.EQ.1) GO TO 11
   10 BT(NGF,IG)=BT(NGF,IG)+SIG
      IBPOS=1
      GO TO 19
   11 IBPOS=0
      BT(NGF,IG)=BT(NGF,IG)-SIG
      IF(BT(NGF,IG).LE.0.) BT(NGF,IG)=0.
      NGF=NGF+1
      GO TO 19
   13 IF(IMPSOL.EQ.0) GO TO 33
      KSUMP=KSUMT
      DO 32 J4=1,NG
   32 BU(J4,IG)=BT(J4,IG)
   33 NGF=1
      IF(ICOB.EQ.1) GO TO 14
      SIG=SIG/5.0
      IF(SIG.LT..001) GO TO 20
      IBPOS=0
      GO TO 3
   14 ICOB=0
      IBESP=0
      IF(IMPSOL.EQ.1) GO TO 16
      DO 15 I2=1,NG
   15 BT(I2,IG)=BU(I2,IG)
      KSUMT=KSUMP
C  THE NEXT 10 CARDS ACCOMPLISH THE PATTERN SEARCH OF THE BETA VALUES.
   16 DO 18 J3=1,NG
      KSUMP=KSUMT
      DEL=BT(J3,IG)-BBP(J3,IG)
      BBP(J3,IG)=BT(J3,IG)
      BU(J3,IG)=BT(J3,IG)
      IF(BT(J3,IG).LE.0.0) GO TO 17
      BT(J3,IG)=BT(J3,IG)+DEL
      GO TO 18
   17 BT(J3,IG)=0.0
   18 CONTINUE
      IBPMP=1
   19 IMPSOL=0
      ILLSOL=0
      IBV=0
      RETURN
   20 NRC=1
      RETURN
      END
```

```
C                    SUBROUTINE CONTRL
C     SUBROUTINE CONTRL SETS INITIAL VALUES AND THE CONSTRAINTS ERROR
C   MULTIPLYING FACTOR.  THIS SUBROUTINE DETERMINES WHETHER THE
C   CONSTRAINTS HAVE BEEN SATISFIED, AND IF SO, RETURNS TO THE MAIN
C   PROGRAM.
C
      SUBROUTINE CONTRL
C
      COMMON AM(40,20),A(40,20),U(20),L(20),GAP(20),UNE(20),LNE(20),
     1UNE(20),UN(20),LN(20),Z(20),W(20),V(20),H(20),X(10),XN(10),EMAX
     2(20,20),VAR(40),BBP(20,20),BT(20,20),BU(20,20),NG,NV,CKT,CKP,CSAT,
     3SIG,NGF,KSUMP,KSUMT,ERR,IFV(1200),CFNR,KA,NV2,KT(20),KRI(20),
     4KBP(20),J,IPC,IBV,KU(20)
      REAL L,LN,LNE
      INTEGER X,XN
C   THE NEXT 12 CARDS SET INITIAL CONDITIONS.
      DO 1 I1=1,NG
      KU(I1)=KBP(I1)
    1 KT(I1)=KBP(I1)
      CSAT=1.0E+10
      CFNR=0.0
      KSC=0
      CKT=200.
      CKP=200.
      DO 6 K3=1,NG
      IAB=K3+NV2
      AM(IAB,J)=BT(K3,J)
    6 CONTINUE
C   THE NEXT 12 CARDS SET THE CONSTRAINTS ERROR FACTOR.  WHEN
C     CONTROL IS RETURNED TO CONTRL, THE RESULTS ARE CHECKED TO SEE THAT
C     THE CONSTRAINTS WERE SATISFIED.
      DO 10 I2=1,10
      IF(KSC.EQ.0) GO TO 8
      IF(CSAT.LE.0.) GO TO 11
    8 CKP=CKT
      CKT=CKT*5.
      CFT=1.0E+15
      CFP=1.0E+15
```

```
      KSC=0
      IW=1
      CALL CONST(IW)
      IF(CFNR.EQ.1.0) GO TO 11
   10 CONTINUE
   11 RETURN
      END
```

```
C     SUBROUTINE CONST DETERMINES WHETHER THE INPUTS TO GATE J SHOULD
C  BE 1 OR 0 AT EACH POINT ON THE N CUBE.  IT THEN CALCULATES THE MINIMUM
C  W AND THE MAXIMUM Z VALUES FOR EACH GATE I FEEDING INTO GATE J.
C
C                    SUBROUTINE CONST
C
      SUBROUTINE CONST(IW)
      DIMENSION IFL1(20),IFL2(20),F(20)
      COMMON A(40,20),A(40,20),U(20),L(20),GAP(20),UNE(20),LNE(20),
     1UNE(20),UN(20),LN(20),Z(20),W(20),V(20),H(20),X(10),XN(10),EMAX
     2(20,20),VAR(40),BBP(20,20),BT(20,20),BU(20,20),NG,NV,CKT,CKP,CSAT,
     3SIG,NOF,KSUMP,KSUMT,ERR,IFV(1200),CFNR,KA,NV2,KT(20),KRI(20),
     4KBP(20),J,IPC,IBV,KU(20)
      REAL L,LN,LNE
      INTEGER F,X,XN
C  THE NEXT 10 CARDS SET INITIAL CONDITIONS.
      IFL3=0
      IFL4=0
      IFL5=0
      IFL6=0
```

```
      DO 1 I1=1,NG
      A(I1)=0.0
    1 Z(I1)=0.0
      DO 2 I2=1,NV
      X(I2)=0
    2 XN(I2)=0
C   THE DO LOOP TERMINATED BY STATEMENT 23 PERFORMS THE FUNCTION OF A
C     BINARY COUNTER INDEXING THE INDEPENDENT INPUTS THROUGH ALL POINTS
C     ON THE N CUBE.
      DO 23 I3=1,IPC
C THE NEXT 31 CARDS ASSIGN THE PROPER VALUE TO INEDPENDENT INPUTS X AND
C     XN.
      IF(X(1).LE.1) GO TO 4
      X(1)=0
      X(2)=X(2)+1
      IF(X(2).LE.1) GO TO 4
      X(2)=0
      X(3)=X(3)+1
      IF(X(3).LE.1) GO TO 4
      X(3)=0
      X(4)=X(4)+1
      IF(X(4).LE.1) GO TO 4
      X(4)=0
      X(5)=X(5)+1
      IF(X(5).LE.1) GO TO 4
      X(5)=0
      X(6)=X(6)+1
      IF(X(6).LE.1) GO TO 4
      X(6)=0
      X(7)=X(7)+1
      IF(X(7).LE.1) GO TO 4
      X(7)=0
      X(8)=X(8)+1
      IF(X(8).LE.1) GO TO 4
      X(8)=0
      X(9)=X(9)+1
      IF(X(9).LE.1) GO TO 4
      X(9)=0
      X(10)=X(10)+1
    4 DO 5 I4=1,NV
      XN(I4)=1-X(I4)
    5 CONTINUE
C   THE NEXT 20 CARDS CALCULATE THE OUTPUT OF EACH GATE (1 OR 0), AND CON-
C     STRUCT THE MATRIX OF INPUT VARIABLES FOR EACH POINT ON THE N CUBE.
      DO 6 I5=1,KA
    6 VAR(I5)=0.
      DO 8 I6=1,NV
      IAA=I6+NV
      VAR(IAA)=XN(I6)
    7 VAR(I6)=X(I6)
    8 CONTINUE
      VAL=0.
      JM1=J-1
      DO 12 I7=1,JM1
      DO 9 J1=1,KA
    9 VAL=A(J1,I7)*VAR(J1)+VAL
      IF(VAL.LT.U(I7)) GO TO 10
      F(I7)=1
      GO TO 11
   10 F(I7)=0
   11 IAB=I7+NV2
      VAR(IAB)=F(I7)
```

```
      VAL=0.
   12 CONTINUE
      IF(IFV(I3).EQ.1) GO TO 17
C  THE NEXT 22 CARDS CALCULATE THE MAXIMUM Z FOR EACH GATE I WHICH FEEDS
C     INTO GATE J.  THE MAXIMUM FOR F(I)=1 AND F(J)=0 IS ALSO CALCULATED.
      DO 16 J5=1,J
      IF(BT(J5,J).LE.0.0) GO TO 16
      IF(F(J5).EQ.1) GO TO 40
      SF2=0.
      DO 13 I9=1,KA
      SF2=AM(I9,J)*VAR(I9)+SF2
   13 CONTINUE
      IF(IFL1(J5).EQ.1) GO TO 15
   14 IFL1(J5)=1
      Z(J5)=SF2
   15 IF(SF2.LE.Z(J5)) GO TO 16
      Z(J5)=SF2
      GO TO 16
   40 SF5=0.
      DO 41 K2=1,KA
      SF5=AM(K2,J)*VAR(K2)+SF5
   41 CONTINUE
      IF(IFL5.EQ.1) GO TO 43
   42 IFL5=1
      FMAX0=SF5
   43 IF(FMAX0.LT.SF5) FMAX0=SF5
   16 CONTINUE
      GO TO 22
C  THE NEXT 22 CARDS CALCULATE THE MINIMUM W FOR EACH GATE I WHICH FEEDS
C     INTO GATE J.  THE MINIMUM FOR F(I)=0 AND F(J)=1 IS ALSO CALCULATED.
   17 DO 21 J6=1,J
      IF(BT(J6,J).LE.0.0) GO TO 21
      IF(F(J6).EQ.0) GO TO 50
      SF1=0.
      DO 18 J7=1,KA
      SF1=AM(J7,J)*VAR(J7)+SF1
   18 CONTINUE
      IF(IFL2(J6).EQ.1) GO TO 19
      IFL2(J6)=1
      W(J6)=SF1
   19 IF(SF1.GE.W(J6)) GO TO 21
      W(J6)=SF1
      GO TO 21
   50 SF6=0.
      DO 51 K1=1,KA
      SF6=AM(K1,J)*VAR(K1)+SF6
   51 CONTINUE
      IF(IFL6.EQ.1) GO TO 53
   52 IFL6=1
      FMIN1=SF6
   53 IF(FMIN1.GT.SF6) FMIN1=SF6
   21 CONTINUE
   22 X(1)=X(1)+1
   23 CONTINUE
C  THE NEXT 26 CARDS SPECIFY THE NEW MINIMUM AND MAXIMUM FOR THE GAP OF
C     GATE J.
      DO 28 J2=1,J
      IF(BT(J2,J).LE.0.0) GO TO 28
      IF(IFL3.EQ.1) GO TO 25
   24 IFL3=1
      ZMAX=Z(J2)
   25 IF(ZMAX.LT.Z(J2)) ZMAX=Z(J2)
```

```
      IF(IFL4.EQ.1) GO TO 27
   26 IFL4=1
      WMIN=W(J2)
   27 IF(WMIN.GT.W(J2)) WMIN=W(J2)
   28 CONTINUE
      IF(IFL5.EQ.0) FMAX0=ZMAX
      IF(IFL6.EQ.0) FMIN1=WMIN
      IF(FMIN1.GT.WMIN) FMIN1=WMIN
      IF(FMAX0.LT.ZMAX) FMAX0=ZMAX
      IF(ISV.EQ.1) GO TO 31
      GNE(J)=FMIN1-FMAX0
   29 LNE(J)=FMAX0
      UNE(J)=FMIN1
      DO 30 J3=1,NG
      IFL1(J3)=0
      IFL2(J3)=0
      V(J3)=WMIN
      H(J3)=ZMAX
      UN(J3)=FMIN1
   30 LN(J3)=FMAX0
      IF(GNE(J).GT.0.0) GO TO 34
C  SINCE THE REALIZATION WILL NOT REALIZE THE FUNCTION WITHOUT ERRORS
C     THERE IS NO NEED TO GO FURTHER.  CFNR IS SET EQUAL TO 1
C     AND IT RETURNS TO CONTRL.
      CFNR=1.0
      RETURN
C  THE NEXT 10 CARDS CALCULATE NEW UPPER AND LOWER LIMITS FOR THE NEW
C     GAP OF GATE J IN THE REALIZATION.
   31 LNE(J)=L(J)
      UNE(J)=U(J)
      GNE(J)=U(J)-L(J)
      DO 33 J4=1,NG
      IFL1(J4)=0
      IFL2(J4)=0
      V(J4)=U(J)
      H(J4)=L(J)
      UN(J4)=U(J)
   33 LN(J4)=L(J)
   34 CALL SEARCH(IW)
C  THE NEXT 8 CARDS CALCULATE THE NEW UPPER AND LOWER LIMITS FOR THE
C     GAP OF GATE J USING THE BEST K VALUES FOUND, BEFORE RETURNING TO
C     CONTRL.
      DO 35 K1=1,NG
      IF(BT(K1,J).LE.0.) GO TO 35
      V(K1)=W(K1)-(BT(K1,J)*ERR)/KBP(K1)
      H(K1)=Z(K1)+(BT(K1,J)*ERR)/KBP(K1)
      IF(V(K1).LT.UN(J)) UN(J)=V(K1)
      IF(H(K1).GT.LN(J)) LN(J)=H(K1)
      GAP(J)=UN(J)-LN(J)
   35 CONTINUE
      RETURN
      END
```

```
C      SUBROUTINE SEARCH DETERMINES THE V AND H VALUES FOR EACH GATE I
C   FEEDING INTO GATE J, USING THE GIVEN K VALUES, AND CALCULATES THE COST
C   FUNCTION.  IT THEN PERFORMS THE OPTIMAL SEARCH, CALCULATING THE COST
C   FUNCTION AFTER EACH STEP.
C
C                      SUBROUTINE SEARCH
C SEARCH SUBROUTINE USING OPTIMAL SEARCH TECHNIQUE
      SUBROUTINE SEARCH(IW)
      DIMENSION UD(20),LD(20)
      COMMON AM(40,20),A(40,20),U(20),L(20),GAP(20),UNE(20),LNE(20),
     1GNE(20),UN(20),LN(20),Z(20),W(20),V(20),H(20),X(10),XN(10),EMAX
     2(20,20),VAR(40),BBP(20,20),BT(20,20),BU(20,20),NG,NV,CKT,CKP,CSAT,
     3SIG,NGF,KSUMP,KSUMT,ERR,IFV(1200),CFNR,KA,NV2,KT(20),KRI(20),
     4KRP(20),J,IPC,IBV,KU(20)
      REAL L,LN,LNE
      INTEGER X,XN
C  THE NEXT 5 CARDS SET INITIAL VALUES.
      CFT=1.0E+15
      CFP=1.0E+15
      KPOS=0
      KPMP=0
    1 KSUMT=0
      DO 2 I1=1,NG
      UD(I1)=UNE(I1)
      LD(I1)=LNE(I1)
    2 KSUMT=KSUMT+KT(I1)
C  THE NEXT 15 CARDS CALCULATE V AND H VALUES, AND DETERMINE THE ERROR
C      DUE TO THE GAP WITH ERR ERRORS NOT BEING WITHIN THE GAP OF GATE J
C      WHEN NO ERRORS ARE MADE.
      EOG=0.
      DO 3 I2=1,J
```

```
      IF(BT(I2,J).LE.0.0) GO TO 3
      V(I2)=W(I2)-(BT(I2,J)*ERR)/KT(I2)
      H(I2)=Z(I2)+(BT(I2,J)*ERR)/KT(I2)
      IF(V(I2).LT.UD(J)) UD(J)=V(I2)
      IF(H(I2).GT.LD(J)) LD(J)=H(I2)
C  THE FOLLOWING 2 CARDS MUST BE MODIFIED IF A MINIMUM GAP IS TO BE
C  SPECIFIED.
      ELL=LNE(J)-V(I2)
      EUL=H(I2)-UNE(J)
      IF(ELL.EQ.0.0) ELL=1.0E-04
      IF(EUL.EQ.0.0) EUL=1.0E-04
      IF(ELL.LT.0.0) ELL=0.0
      IF(EUL.LT.0.0) EUL=0.0
      EOG=EOG+ELL+EUL
    3 CONTINUE
C  THE NEXT 11 CARDS CALCULATE THE ERROR DUE TO THERE NOT BEING A GAP,
C      IF ANY V VALUES ARE LESS THAN ANY H VALUE.
      EFGT=0.
      DO 4 I3=1,J
      IF(BT(I3,J).LE.0.0) GO TO 4
      DO 20 J1=1,J
      IF(BT(J1,J).LE.0.0) GO TO 20
C  THE FOLLOWING CARD MUST BE MODIFIED IF A MINIMUM GAP IS TO BE
C  SPECIFIED.
      EFG=(Z(I3)+(BT(I3,J)*ERR)/KT(I3))-(W(J1)-(BT(J1,J)*ERR)/KT(J1))
      IF(EFG.EQ.0.0) EFG=1.0E-04
      IF(EFG.LT.0.0) EFG=0.0
      EFGT=EFGT+EFG
   20 CONTINUE
    4 CONTINUE
C  THE NEXT 4 CARDS CALCULATE THE COST FUNCTION USING KT VALUES.
      EOG=EOG*1.0E+05
      EFGT=EFGT*CKT
      CSAT=EOG+EFGT
      CFT=CSAT+KSUMT
C  THE REMAINING STATEMENTS COMPRISE THE SEARCH TECHNIQUE FOR THE K
C      VALUES.
      IF(CFT.LT.CFP.AND.KPMP.EQ.1) GO TO 16
      IF(CFT.GE.CFP) GO TO 55
C  THE KU VALUES ARE SET EQUAL TO THE KT VALUES BECAUSE THE COST FUNC-
C      TION WAS REDUCED.
      DO 50 J4=1,NG
      UN(J4)=UD(J4)
      LN(J4)=LD(J4)
   50 KU(J4)=KT(J4)
      IF(KPOS.EQ.1) IW=IW+1
      KPOS=0
C  THE NEXT  4 CARDS START AN EXPLORATORY SEARCH IF ONE IS NOT ALREADY
C      IN PROGRESS.
   55 IF(KESP.NE.1) GO TO 5
      IF(IW.GT.NG) GO TO 12
    5 KESP=1
      KPMP=0
      IF(CFT.LT.CFP) GO TO 7
C  THE KT VALUES ARE RESTORED TO THE VALUE THEY HAD BEFORE THE STEP
C      BECAUSE THE COST FUNCTION INCREASED.
      DO 6 I4=1,NG
    6 KT(I4)=KU(I4)
      GO TO 8
C  THE BEST PREVIOUS COST FUNCTION IS SET EQUAL TO CFT BECAUSE THE SEARCH
C      WAS SUCCESSFUL.
    7 CFP=CFT
```

```
      IWS0=1
    8 IF(KPOS.EQ.1) GO TO 10
C   THE NEXT 2 CARDS INDEX KT(IW) POSITIVE BY 1.
    9 KT(IW)=KT(IW)+1
      KPOS=1
      GO TO 1
C   THE NEXT 4 CARDS INDEX THE KT(IW) NEGATIVE BY 1 AND INDEX IW FOR THE
C      NEXT PASS.
   10 KPOS=0
      IF(KT(IW).LE.1) GO TO 11
      KT(IW)=KT(IW)-1
      IW=IW+1
      GO TO 1
   11 KT(IW)=1
      IW=IW+1
      IF(IW.LE.NG) GO TO 9
   12 IW=1
      IF(IWS0.EQ.0) GO TO 13
      GO TO 14
   13 KSC=1
      RETURN
   14 IWS0=0
      KESP=0
      IF(CFT.LT.CFP) GO TO 16
C   THE NEXT 2 CARDS RESTORE THE KT VALUES TO THE VALUES THEY HAD BEFORE
C      THE LAST STEP, SINCE THE LAST STEP FAILED TO REDUCE THE COST
C      FUNCTION.
      DO 15 J2=1,NG
   15 KT(J2)=KU(J2)
C   THE NEXT 11 CARDS ACCOMPLISH THE PATTERN MOVE.
   16 DO 18 J3=1,NG
      IF(CFT.LT.CFP) CFP=CFT
      DEL=KT(J3)-KBP(J3)
      KBP(J3)=KT(J3)
      KU(J3)=KT(J3)
      IF(KT(J3).LE.1) GO TO 17
      KT(J3)=KT(J3)+DEL
      GO TO 18
   17 KT(J3)=1
   18 CONTINUE
      KPMP=1
      GO TO 1
      END
```