COMPUTER AIDED OPTIMISATION OF COMBINATIONAL LOGIC

Christopher William NETTLE, B.Sc., B.E.(Hons.)

March 1979

Being a thesis submitted to
the University of Adelaide
for the degree of
DOCTOR OF PHILOSOPHY
in the
Department of Electrical Engineering

TABLE OF CONTENTS

COMPUTER AIDED OPTIMISATION OF COMBINATIONAL LOGIC

SUMMARY

This thesis is concerned with the problem of Boolean minimisation with a view to its practical application in optimising logic networks. In this context, an optimal circuit is one which has the lowest possible cost according to a user-defined cost function, which may specify minimum numbers of terms, literals, gates or packages, or linear combinations of these quantities. Only two level AND-OR circuits are considered, since NAND-NAND, OR-AND and NOR-NOR realisations may readily be obtained by suitable logic manipulation of the function and its solution.

To put this study into perspective, there is a brief look at the history of this problem and its various "solutions". Particular emphasis is placed on the works of H.R. Hwa (1974) who outlined an efficient method for generating prime implicants, and T.C. Bartee (1961) who first described the "tag method" for multiple output networks.

The bulk of the thesis is devoted to the development and explanation of an algorithm to solve the covering problem and its implementation as a computer program (which is reproduced in full in the appendix). The method takes as its starting point the prime implicant table, and proceeds to construct a tree in which nodes represent a partial cover of the table. The branches from a particular node represent the choice of rows available to cover one of the uncovered columns of that node. The size of the tree is restricted by employing an ordering on the rows, and redundant or unused parts of it are deleted whenever necessary to economise on time and storage.

The program is designed to run on a CDC 6400 computer, and is written in FORTRAN Extended, with a few minor assembly language routines for bit manipulation.

Although principally intended for single-output networks, the technique may be extended to cater for multiple-output problems, and this is discussed and provided in the program.

The effectiveness of this incompletely-specified tree approach by comparison with existing methods is demonstrated by the enlarged class of problems which the program is capable of solving. A consideration of the performance of this program and of store requirements in general leads to the conclusion that complete optimisation of large problems by computer is unlikely to ever prove viable. For the majority of applications, however, the combinational stage of design may be segmented into problems having fewer than eleven variables, in which case this program can provide a ready solution.

STATEMENT

This thesis contains no material which has been presented for the award of any other degree or diploma in any university. To the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except when due reference is made in the text of the thesis.

C. NETTLE

ACKNOWLEDGMENTS

The author gratefully acknowledges the assistance offered by Prof. H.T. Nagle of Auburn University, Prof. P.S. Noe of Texas A & M University and Prof. D.L. Dietmeyer of the University of Wisconsin, all of whom supplied information for comparative purposes. The helpful advice and comments offered by J.P. Roth of IBM Corporation and Dr. A. Dunworth of the University of New South Wales are also greatly appreciated.

Finally, the author expresses sincere appreciation to his supervisor, Dr. D.A. Pucknell, for his patience, guidance and support throughout this research.

1. INTRODUCTION

1.1 General

Digital networks of various forms are being used with increasing frequency in current technology. As the size and complexity of such systems increase, so we come to rely more heavily on computers as tools to aid in the design process. The advent of "computer aided design" has highlighted the need for developing new techniques, ones which make more efficient use of the computer's capabilities.

Boolean minimisation has long been a thorn in the side of design engineers, because although it is a necessary step in planning most logic networks, only a limited class of problems can be handled effectively by existing methods. General techniques exist, but because of their excessive demands on computer time and storage for even medium sized problems, their utility is limited. A need therefore exists for a method which makes efficient use of computer resources, which is generally applicable, and which produces a reduced solution in cases where optimisation requires too much time. This study is concerned with the development of just such an algorithm.

It is worth noting at this stage the distinctions between the following terms:

"optimisation" means obtaining a truly minimum cost according to specific criteria,

"minimisation" means obtaining an expression having the least number of terms, and having the smallest number of literals if there is a choice,

"simplification" usually means reducing the network cost to a local but not necessarily absolute minimum, and

"reduction" (or "approximation") refers to any means of reducing the cost which does not aim at a minimum value.

A successful method of optimising such systems would have very wide application, because they form part of synchronous, asynchronus and static logic, as well as appearing in a disguised format in computer software itself.

The present work attempts to find a general method for optimising all possible combinational circuits, the associated computer program, however, is not so ambitious; indeed the indications are that such a goal is unobtainable, but it should prove to be an invaluable aid in areas where current methods are inadequate.

1.2 Problem Specification

The problem to be considered is that of optimising a 2-level combinational logic network. The assumptions made are as follows:

(1) The resulting circuit shall contain no more than two levels of gates. This restriction exists because the problem of optimising multi-level circuits was found to be intractable. An advantage of this configuration is that the signals undergo minimum delay, which is normally desirable.

(2) All input variables are available in both complemented and uncomplemented form. This is rendered necessary by the above assumption, as otherwise a three level circuit is required in general. Such circuits are referred to in the literature as TANT networks (Gimpel, 1967). In many applications, input variables are derived from other logic devices such as flip-flops, and so such an assumption is not unrealistic.

(3) There is no "FAN-IN" or "FAN-OUT" limitation on gates or input variables: such restrictions generally are impossible to satisfy in two levels, but for most practical cases this is not a real limitation.

(4) only a sum-of-products (AND-OR) realisation will be described explicitly, but such results may readily be interpreted as NAND-NAND (by complementing any single literal terms). An OR-AND (or NOR-NOR) expression may be obtained from the solution of the complementary problem.

(5) Only single output networks are considered except where otherwise indicated. Multiple output networks may be handled by straightforward modification of single output techniques, as will be described later.

Recent improvements in medium and large scale integrated circuit technology have led to the ready availability of more complex devices capable of performing many functions on a single chip. A PLA (or FPLA) for example consists of an array of gates specifically for realising multiple output combinational logic functions of the type to be considered here. If the function desired is small enough to be accommodated in an available commercial package, and

if programming facilities are available, then naturally there is no need to optimise the circuit for discrete gates. The cost of an FPLA chip may be ten times greater than using individual gates, however, which may be a deciding factor. In cases where PLAs are to be used, but the function contains more terms than are provided for in the device, a procedure such as described here is required to reduce the problem until it fits.

Multiplexers may also be used to realise combinational logic networks having a small number of variables, but for larger problems, auxilliary gating is required, adding to the cost and propagation delay of the final circuit.

A very straight-forward way to create a multiple output circuit is to use a ROM (read-only memory) in which each address bit corresponds to an input variable, and each data bit corresponds to an output variable. No minimisation is necessary, as the ROM is programmed with the canonical expansion of the function. Unfortunately the penalties for using this approach are a much greater propagation delay and greater cost (usually) than using discrete gates.

It appears, then, that a need still exists for combinational logic minimisation, even though it is used in conjunction with or even competes with devices such as those described above.

1.3 Definitions

Familiarity with the normal operations of Boolean algebra, conjunction (OR), disjunction (AND), negation (NOT) is assumed.

A literal is a single occurrence of a variable in either its true or negated state, such as a or $\bar{a}$.

A term is a disjunction of literals.

A fundamental formula is a term in which no variable appears more than once.

A function is said to be in normal form if it is expressed as a conjunction of fundamental formulae.

An irredundant normal form is one in which no term may be removed without changing the function represented.

The number of binary variables of a function F is represented by n.

A minterm of function F is a fundamental formula of F containing exactly n literals.

An irredundant normal form composed entirely of minterms is known as a canonical expansion (also referred to in the literature as a developed or expanded normal form).

The complementary function $\bar{F}$ of function F is given by the conjunction of all minterms of the unity function specifically excluded from F.

A dontcare term of function F is a minterm of the unity function not contained in F or $\bar{F}$ (and is therefore also a dontcare term of $\bar{F}$).

A term I is said to be an implicant of F if F is true whenever I is true.

A prime implicant P of function F is an implicant of F such that the term obtained by deleting any literal from P is not an implicant of F. Notice that P must hence be a fundamental formula. The abbreviation PI is used to represent "prime implicant".

An irredundent cover of function F is a set of prime implicants (PIs) whose conjunction is an irredundent normal form of F.

An essential prime implicant (EPI) of function F is a prime implicant of F which appears in all irredundant covers of F.

The cost of a prime implicant is defined as the weighted sum of various attributes of that PI.

The cost function is a set of weighting factors for determining the cost of a PI. Number of terms, literals, gates or packages in a circuit realisation may be minimised by suitable choice of this function.

A consensus operation may be performed on two fundamental formulae which have the property that precisely one variable appears affirmed in one and negated in the other. The resulting consensus term is the fundamental formula resulting from the conjunction of all other literals in the two generating terms. For example, the consensus of (acd) and ($ac\bar{d}e$) is (ace).

A term A is said to subsume term B if B is an implicant of A and A is not an implicant of B. For example, (ac) subsumes (abc).

2. STATE OF THE ART

2.1 Brief History

The application of truth-function logic to electric switching circuits was demonstrated by Shannon in 1938, but no real headway was made in the field of logic minimisation until the classic paper published by Quine in 1952. In this paper, Quine established the fact that any minimal sum of products expression must be composed of prime implicants. He espoused a very rudimentary form of the consensus method for generating these PIs, which he subsequently refined in his paper of 1955. Meanwhile, Samson and Mills (1954) devised much the same approach, based on Quine's earlier paper. There is reference however to the consensus operation havings its origin in a Ph.D. dissertation by Blake in 1937.

McCluskey (1956) developed a different approach based on Quine's papers, his being the tabular method for generating PIs which has since become known as the "Quine-McCluskey" method. These two methods have formed the backbone for PI generation ever since, although other algorithms have been developed. Unfortunately the problem of using these terms to generate an optimal irredundant sum has proved to be a much more demanding task.

Karnaugh (1953) proposed a map method which to this day is widely favoured for hand calculation of small problems, however it becomes too complex for the powers of human pattern recognition for problems of about six input variables. Attempts to program such an approach on a computer (or even to build a special purpose computer) usually result in some form of "heaviest-first" method,

which in general cannot be relied upon to yield an optimal solution.

A means of deriving all irredundant sums of PIs was detailed by Petrick (1956), but while it is conceptually feasible, in practice the number of such sums becomes prohibitively large for problems in excess of about six variables. McCluskey (1956) showed that the selection of PIs became a "covering problem" on the prime implicant table, and attempted to simplify the problem by reducing the table. Gimpel (1965), Grasselli and Luccio (1968) and others have contributed ways of further reducing the table, but such techniques are only worthwhile when they yield a reduced table which is manageable with Petrick's (or some similar) method: such is not the case for problems of eight or more variables, where the relative reduction in table size is usually quite small.

This lack of a general-purpose method for minimising even modest sized problems has led other researchers such as Bowman and McVey (1970) to look for approximate solutions which may be computed much more quickly than truly optimal solutions. The resulting networks are usually termed "minimal", as distinct from "optimal", and the term "solution" is often expanded to include virtually any network realisation of the function, rather than only networks having the smallest cost.

One of the earliest papers on multiple output problems was by Bartee (1961), in which he developed the concept of "multiple output prime implicants", and discussed a means for generating such terms. Mage (1970) pointed out a relatively minor flaw in the method, but it still remains

one of the best available because it is systematic and readily programable.

2.2 Summary of Other Methods

2.2.1 Quine-McCluskey

This algorithm was described with full details and examples by McCluskey (1956), but is presented here for the sake of completeness.

The function to be minimised is first expanded if necessary into its canonical form. The minterms and dontcares are then listed in groups so that within each group every term has the same weight (number of true literals). Consensus terms are then formed between members of adjacent groups, yielding a second list containing all (n-1) literal implicants. This list is grouped according to weight also, and all used terms of the first list are marked. The second list is used to obtain a third, in the same way, and so on until no further lists are possible. At this stage, all unmarked terms must be prime implicants.

The prime implicant table (PI table) is then set up, having one row for each PI and one column for each minterm. An X is entered in the table for every element (i,j) such that column j is covered by row i (minterm j is an implicant of term i).

A set of rows (PIs) is then selected such that each column is covered (contains an X in at least one selected row), and such that no set having

fewer members exists. To this end a certain amount of table reduction is possible by selecting essential rows which cover a column containing only one cross; deleting dominated rows and dominating columns. Gimpel (1965) showed how a still smaller table may be formed by removing a column of weight two and making a conditional row selection.

When no further reduction is possible, McCluskey describes the table as "cyclic" and offers no better suggestion for proceeding than simple trial and error.

2.2.2 Petrick

In any PI table, a particular column contains crosses in a small subset of rows only, and one of this subset of rows must be selected to form a covering set. Petrick (1956) expressed this fact with logic symbols, in which each variable represented the act of selecting one particular row. The constraint imposed by each column becomes a sum of (true) variables, and since every column must be covered, the whole table may be replaced by a product of sums expression. When this expression is multiplied out to a sum of products form, each product represents a covering set of PIs, and the optimum solution is obtained by choosing the set having the lowest cost.

This method of realising all the irredundant normal forms of a Boolean function is simple and elegant, but as mentioned before is in most cases impractical.

2.2.3 Karnaugh

This method involves mapping the desired minterms and dontcares into the cells of a matrix. Adjacent cells are arranged to represent minterms which differ in only one literal. Implicants are represented by rectangular arrays containing 2^i cells, where i is any integer, and a minimal cover is obtained by choosing the smallest number of the largest possible implicant groups (PIs) such that each minterm cell is included. The Karnaugh map method is well suited to hand calculation for small problems, but because it is virtually a "heaviest-first" approach, it can not reliably optimise larger problems, particularly as the maps required are difficult to represent.

2.2.4 Residues method

This is a technique for generating PIs, and appears in the literature in various forms under such names as "partitioned lists" (Morreale, 1968) and "clause tables" (Schmidt and Druffel, 1974). Broadly speaking, all are based on the "expansion theorem" of Scheinman (1962), which expresses any function of n variables in terms of two "residue functions" involving (n-1) variables. In essence

each method relies on multiplying out or otherwise expanding the canonical product-of-sums expansion in a well regimented fashion. Some of these approaches may generate non-prime implicants, and indeed may not generate every single prime implicant.

2.2.5 Integer Programming

As its name suggests, this approach seeks to express the covering problem as a set of mathematical constraints so that integer programming techniques can be applied to it. Luccio (1966) describes such a method, but as may be anticipated, changing the format of the problem does not lead to an easier solution: trial-and-error branching is still required.

2.2.6 Branch and Bound

It appears that the search for a deterministic solution to the covering problem has generally been abandoned in favour of some form of branch-and-bound approach. The "extraction method" described by Roth and Wagner in 1959 and again in 1961 with Ewing actually contained little innovation over McCluskey's tabular approach, but their advocation of a branching method opened up the avenues by which larger problems might be tackled. By computing PIs only as they are required, the "local extraction method" (Su, 1969) is useful for problems having a large number of variables but a small number of terms in the solution.

Essentially, this method procedes by selecting EPIs and removing redundant (dominated) PIs until a choice is required. The choice made is either to select or reject a chosen PI, and then to procede as before. It is similar to the algorithm described in the next chapter, but differs in several important respects:

(1) The depth of branching is governed by the number of PIs rather than the number of minterms, which is usually less, thus the tree is generally bigger.

(2) The strategies for PI selection and backtracking are far less efficient.

(3) Searches for extremals (EPIs) and redundant PIs are required, rather than incorporating such work in the branching process itself.

Davidson (1969) uses a tree structure in which each node represents a class of feasible solutions, and branching is only continued as long as cheaper solutions are obtained. He mentions the critical dependence of his approach on the heuristic strategies which are used in making decisions within the tree. This is also a feature of the present algorithm to be described in chapter 3, and is probably true of any such trial and error approach.

A similar approach is outlined by Bowman and McVey (1970), but their's is a simplified form, relying heavily on their strategies, and does not necessarily yield an optimum solution.

2.3 Present limitations

It seems, then, that eight, nine or ten variable combinational networks can not in general be optimised by use of the classical methods. Certainly some examples may be handled, but these approaches are better suited to hand rather than machine computation: Karnaugh's method requires pattern recognition, Petrick's requires a huge store, and implementations of the Quine-McCluskey method normally do not cater for "cyclic tables".

A further deficiency, one which has acquired increasing practical significance in recent times, is that previous techniques did not normally take into consideration the logic packages to be used or their cost. With the advent of LSI technology in fact, a need has emerged for a very general cost-function, since the significant parameters may be circuit area or interconnections, rather than the classical "number of diodes".

Branch and Bound techniques have exhibited their worth by extending the frontier of solvable problems well past ten variables. In fact programs have been written which could minimise some 35 variable problems (Ewing et al, 1961), but simple calculation indicates that no present computer could possible even store the problem definition, let alone optimise, an arbitrary 35 variable problem. Minterms and dontcares are selected from a set of 2^{35} members (canonical terms), which requires $\log_2 3 \times 2^{35}$ bits of store for perfect coding, and 2^{36} bits if standard coding is used: no computer storage system has 70 billion bits of store. In fact, the machine available for this study was not big enough for all

arbitrary 10 variable problems, and so this was felt to be a realistic upper limit for programming purposes.

With these points in mind, an attempt was made to develop a general-purpose approach: one which could utilise the power of a computer (namely repetitious trial and error), which was flexible enough to permit the designer to specify the criteria by which a network is judged, and which could provide a workable approximate solution if optimisation could not be obtained. The resulting algorithm, the subject of this dissertation, satisfies these aims, and is described in the following chapter.

3. THE ALGORITHM

3.1 Broad Outline

As in the classical approach (Quine - McCluskey), the problem of obtaining an optimal network is tackled by first generating all the PIs of the function. The PIs are assigned costs according to a (linear) cost function, and then an irredundant covering set is selected such that the sum of costs is a minimum. Notice that because PIs may have differing costs, it is not possible in general to remove from consideration a PI corresponding to a dominated row of the PI table, and so the total PI set must be considered.

Once this set is determined, the PI table is drawn up, and then the solution tree may be constructed. Within the tree, each node represents a partial cover of the table, and each link represents the act of selecting a row of the table.

The operation of the strategies involved is such that a terminal node is reached as quickly as possible, which means that an approximate solution is readily available. Successive approximate solutions have decreasing costs, and so the algorithm still remains useful in cases where to obtain a truly optimum solution would be too time consuming.

To summarise then, the steps involved in using this method to optimise a network are:

(1) Expand all implicants so that the function is expressed in canonical form.

(2) Generate the complete set of prime implicants from the minterms and dontcares.

(3) Construct the PI table having one row per PI and one column per minterm.

(4) Assign a cost to each PI according to user-defined coefficients.

(5) Represent the choices available in covering the PI table by means of a tree.

(6) Extract a solution set of PIs by following the path from the cheapest terminal node back to the root node.

Step 1 is elementary Boolean algebra and requires no further comment. Step 2 may be accomplished by using any of the existing techniques, but the method due to Hwa was found to be the most efficient, and is described in the following section. Step 4 determines the characteristics of the final network, and is discussed in section 3.3. Precise details of the algorithm itself, step 5 are presented in section 3.4.

3.2 Generating the Prime Implicants

This method was briefly outlined by Hwa (1974), and what follows is an expanded interpretation of that paper. Basically, this is an application of the consensus method (Quine, 1955), but one in which the comparisons and tests proceed in a highly ordered fashion. The result is a three-fold advantage over standard consensus in that (a) the number of unproductive comparisons is greatly reduced, (b) some redundant consensus terms are not generated, and (c) less checking is required for one term subsuming another.

Firstly, the list of minterms and dontcares is divided into groups of equal weight (number of uncomplemented literals). Starting with the group having the lowest

weight, each member is compared with the members of the next group (having a weight greater by one). If a consensus term can be formed having the same number of complemented literals as members of the second group, it is added to that group. Any subsumed terms in the second group are deleted immediately, but subsumed terms in the first group are not deleted until they have been used in all possible comparisons. The procedure is continued by comparing second group members with the next group, and so on until all terms have been considered.

Notice that this is essentially a single-pass method, although a certain amount of back-tracking is required to check for deletions. New terms are kept together with their generating term of the same group, so as to form a subgroup. It happens that a new term may only be subsumed by a member of its own subgroup, and may only subsume members of its own subgroup or of the previous main group, which leads to a large reduction in the number of checking operations to be performed.

At the conclusion of this procedure, all remaining terms constitute the complete set of PIs of the function. By way of illustration, consider the function whose

minterms have the following decimal representation:
(6,7,16,17,18,19,20,21,22,23,24,25).

1.1	1	0	0	0	0
2.1	0	0	1	1	0
2.2	1	0	0	0	1
2.3	1	0	0	1	0
2.4	1	0	1	0	0
2.5	1	1	0	0	0
3.1	0	0	1	1	1
3.2	1	0	0	1	1
3.3	1	0	1	0	1
3.4	1	0	1	1	0
3.5	1	1	0	0	1
4.1	1	0	1	1	1

TABLE 3.1 Minterms arranged in ascending weight.

Table 3.1 shows the minterms arranged into ascending weight - individual terms are labelled for didactic reasons only. Term 1.1 is compared with term 2.1, but a consensus term is not possible. A consensus term is generated however from terms 1.1 and 2.2: the new term (1000-) subsumes 2.2 and so replaces it. The new 2.2 subsumes 1.1, and so 1.1 is marked with a cross for later deletion. Terms 2.3, 2.4 and 2.5 are used and replaced in the same way, after which 1.1 is deleted, leaving the situation shown in table 3.2

2.1	0	0	1	1	0
2.2	1	0	0	0	-
2.3	1	0	0	-	0
2.4	1	0	-	0	0
2.5	1	-	0	0	0
3.1	0	0	1	1	1
3.2	1	0	0	1	1
3.3	1	0	1	0	1
3.4	1	0	1	1	0
3.5	1	1	0	0	1
4.1	1	0	1	1	1

TABLE 3.2 After comparing groups 1 and 2.

Term 2.1 is now compared with each of the terms in group 3, causing 3.1 and 3.4 to be replaced. Next 2.2 is compared with the terms of group 3, causing 3.2, 3.3 and 3.5 to be replaced, but since none of the new terms subsumed 2.2, it remains uncrossed: table 3.3 shows the terms at this stage.

2.1	0	0	1	1	0	X
2.2	1	0	0	0	-	
2.3	1	0	0	-	0	
2.4	1	0	-	0	0	
2.5	1	-	0	0	0	
3.1	0	0	1	1	-	
3.2	1	0	0	-	1	
3.3	1	0	-	0	1	
3.4	-	0	1	1	0	
3.5	1	-	0	0	1	
4.1	1	0	1	1	1	

TABLE 3.3 After comparing 2.2 with group 3.

Combining 2.3 with 3.2 yields (100- -) which replaces 3.2 and subsumes 2.2 and 2.3 which must now be crossed. Consider the comparison between 2.3 and 3.3:

2.3	1	0	0	-	0	X
3.3	1	0	-	0	1	
	1	0	0	0	-	?

The consensus term is possible, but it is ignored because it contains three 0's, and may not be entered in group 3 which must contain only terms containing two 0's. The new term obtained from 2.3 and 3.4 (10-10) does not subsume 3.4, and so it is added to the list as a new entry, 3.4.1.

2.4 (and 2.5) must be compared with both 3.4 and 3.4.1: in fact 2.4 and 3.4 combine to yield a third member of subgroup 3.4, namely 3.4.2 (101-0). Combining 2.4 and 3.4.1 however gives (10- - 0), which replaces 3.4.1 and causes 3.4.2 to be deleted.

3.1	0	0	1	1	-
3.2	1	0	0	-	-
3.3	1	0	-	0	-
3.4	-	0	1	1	0
3.4.1	1	0	-	-	0
3.5	1	-	0	0	-
4.1	1	0	1	1	1

TABLE 3.4 After comparing groups 2 and 3.

Table 3.4 shows the situation just prior to comparing groups 3 and 4, and applying the rules again yields table 3.5 in which only PIs remain.

3.5	1	-	0	0	-
4.1	-	0	1	1	-
4.1.1	1	0	-	-	-

TABLE 3.5 The prime implicants at completion of Hwa's method.

3.3 The Cost Function

The only restrictions which the algorithm places on the cost function are that the addition of an extra term must increase the cost of a cover, and that prime implicants are not more expensive than implicants which they subsume. The latter feature is true of all practical instances and is universally assumed throughout the literature.

It is convenient to express the cost of a set of PIs as the sum of the costs of the individual PIs. This facilitates the operation of computing the cost of such a set obtained by the addition of one PI to an existing set. Costs are assigned to PIs according to the formula:

$$C = Txt + Lxl + Gxg + PxV(1)$$

where t is the number of terms which this PI represents (always one for a single output circuit)

l is the number of literals in the PI

g is the number of gates required to implement the PI (assumed to be zero for a single-literal PI, one otherwise, but the general formula allows for 2 or more if gate expanders are used for example).

V is a general purpose user-defined function of the number of literals, but which has one obvious application in specifying the package requirement for this PI.

and T,L,G,P are the user-defined cost coefficients.

By appropriate choice of these latter four coefficients, the designer is able to vary the priority with which various attributes of the network are minimised. For example, (1,0,0,0) minimises the size of the secondary gate (smallest number of terms), (0,1,0,0) leads to the smallest number of literals, and (0,1,1,0) represents "diode" minimisation (least number of primary plus secondary inputs).

Classical minimisation (least terms, least literals if a choice exists) is obtained with (4608,1,0,0). (Proof that 4608 is great enough for 10 variables is given in Appendix E.)

For standard TTL packages, V might be defined as follows:

Literals:	1	2	3	4	5	6	7	8	9	10
Package :	0	•25	•33	•5	1	1	1	1	-	-
V:	0	3	4	6	12	12	12	12	100	100

PI costs (and hence V) are constrained to be integral for programming expediency, thus a normalising factor (12) has been used above. This in no way affects the final result because it is relative cost rather than absolute cost which is important in comparing solutions. The V values may also be used to reflect economic cost: if for example 9 and 10-input gates are extremely expensive, V(9) and V(10) are assigned on appropriately high value (100 above).

The algorithm relies on PIs having strictly positive costs, and so care must be exercised to ensure that the assignment of coefficients can not lead to a PI having negative or zero cost. For example, (0,0,0,1) will effectively minimise packages, but a single-literal PI would have zero cost if the above V function were used, and so a better set would be (1,0,-1,10), where P is chosen to be large enough that the component of cost due to single-literal PIs only becomes significant where package costs are identical. In fact the cost coefficients (1,0,0,512) are more useful, in that the solution obtained is one having the minimum number of packages and the smallest secondary gate if a choice exists. 512 is chosen to ensure that variations in package costs swamp variations in term costs.

The assignment of package values as suggested may lead to an error as great as one in the worst case where the unfilled packages consist of:

1	x	4	I/P	NAND	=	0.50	Packages
2	x	3	I/P	NAND	=	0.67	Packages
3	x	2	I/P	NAND	=	0.75	Packages
Total						1.92	Packages

In fact 3 ICs are required, not just 2. It is felt, however, that this degree of inaccuracy may readily be tolerated in view of the simplicity of the cost manipulation.

The example illustrated in table 3.6 is a case where the classical minimum solution requires more packages than necessary. Solution (B,D,E) contains 3 terms and 13 literals and requires 1.83 packages. Solution (A,B,C)

has one more literal but only requires 1.67 packages - in fact the total circuit fits into 2 NAND ICs whereas the "minimal" solution requires 3 standard 14-pin ICs.

PI	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
A -------000	x	x	x	x	x	x	x			
B ------0-00	x	x	x	x	x	x		x		
C 00000000--									x	x
D -----000-0	x	x	x	x	x				x	
E 0000---00-					x	x	x			x

TABLE 3.6 Package optimisation example

This example also serves to illustrate one technique by which a problem may be constructed from any PI table. The procedure for creating an n-variable problem from an n-column PI table may be obtained by inspection of table 3.6. If a dominated row exists add a further variable to the problem by appending a dash to the corresponding PI and a zero to all others, which corresponds to the addition of a dontcare term having the same form (2^i) as the minterms. Thus it is valid to choose examples for illustration by choosing any matrix of blanks and crosses having at least one cross per column, since at least one function having that matrix as its PI table must exist.

The secondary gate is not included in the foregoing cost calculations for the following reasons. It will normally require an extra package, and if not, minimising its size will nearly always coincide with the optimum package solution. This gate must exist for non-trivial functions, and thus it is reasonable to exclude it from the gate count when comparing solutions. The sole attribute under the control of the designer is the number of inputs, which is readily minimised by assigning an appropriately high T coefficient.

3.4 Solving the Covering Problem

Quine-McCluskey have shown that the problem of Boolean minimisation reduces to that of obtaining a minimum cover for the associated PI table. Unfortunately, although the objective may be simply expressed, no method has yet been found for proceeding directly to such a cover other than some form of complete enumeration. The rules they (and others) present for selecting particular rows and for reducing the table are quite effective and often sufficient for combinational networks of up to 6 variables. As the number of variables increases however the relative reduction in table size due to the application of these rules reduces, resulting in quite large PI tables for which there is no recourse other than trial and error.

At any stage in the application of this approach, there are choices to be made, each of which leads to further choices, and so the concept of using a tree structure to represent the layout of these choices readily presents itself. The method to be described is just such an approach: a tree is constructed in which each node represents a particular partial cover of the table, and each link represents the choice of a particular row. Every PI is assigned a cost, and hence each node has a cost which is given by the cost of its predecessor node plus that of the PI on the link joining them. An optimal solution is then quite readily obtained by choosing from all nodes with a complete cover, one having the minimum cost: the solution set is given by those PIs on the path back to the root node. The power of such an approach in yielding an answer as efficiently as possible is heavily dependent upon

the way it is implemented and on the strategies used. These are discussed in the following sections but first it is necessary to define some of the terms used.

The first node in the tree is called the root, and is defined to have zero cost and empty covering set. The path to a node is the set of nodes and links connecting that node to the root. The PIs appearing on the path to a node constitute a cover of a set of columns in the table. The associated minterm set itself constitutes a cover which is termed node cover, which is therefore defined as the set of columns (or minterms) of the PI table which are covered by the rows (or PIs) on the path of that node. Average cost of a node is simply the cost of that node divided by the number of columns it covers. A terminal node is one for which all columns are covered. As construction of the tree progresses, those nodes for which successors have been determined are said to be ticked. Notice that once a particular unticked node has been selected to continue branching, a complete set of successor nodes is determined.

By way of illustration, consider the following PI table:

P.I.	Cost	1	2	3	4
A	1	X	X		
B	2	X		X	
C	2		X		X
D	4		X	X	X

TABLE 3.7 Sample PI table

For simplicity, the PIs are represented by letters, the columns are numbered, and costs are arbitrarily assigned. Notice that although this is not a "cyclic" table by

McCluskey's (1956) definition, row C may not be deleted due to domination by row D because C is cheaper than D.

The basic operation of the method is quite straightforward: once the root node is set up, one column to be covered (e.g. 1) is chosen from the table. One new branch is set up corresponding to each row of the table which covers this column (A and B in this case). Each of these branches terminates on a new node, having the same cost and cover as the row concerned (since by definition, the root node has zero cost and empty cover). When all such branching is complete, the root node is ticked, and one of the new nodes is selected to continue the process (e.g. node 1). Figure 3.1 shows the tree at this stage.

A column is chosen which is not covered by this node (e.g. column 3) and new branches are set up corresponding to the rows of the table which cover this column (B and D). Each branch leads to a new node having a node cost and cover as previously defined. The cost of node 3 for instance is equal to the cost of node 1 (1) plus the cost of row B (2), namely 3. The cover of node 4 is given by the conjunction of node 1 cover (1100) and row D cover (0111) to give (1111). Node 4 is therefore a terminal node, since all columns are covered. This situation is shown in Figure 3.2.

As branching from each node finishes, it is ticked, and a new unticked node is selected until none remains. The solution set may then be obtained by examining the path to the cheapest terminal node. From Figure 3.3, node 6 is the cheapest terminal node, and so the solution set is (B,C).

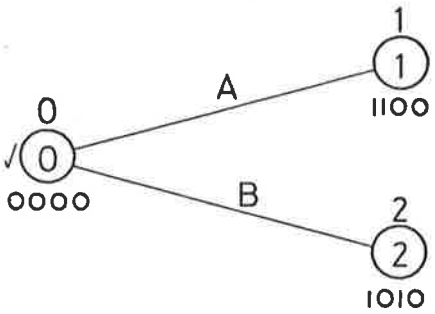


Fig 3.1 START OF TREE

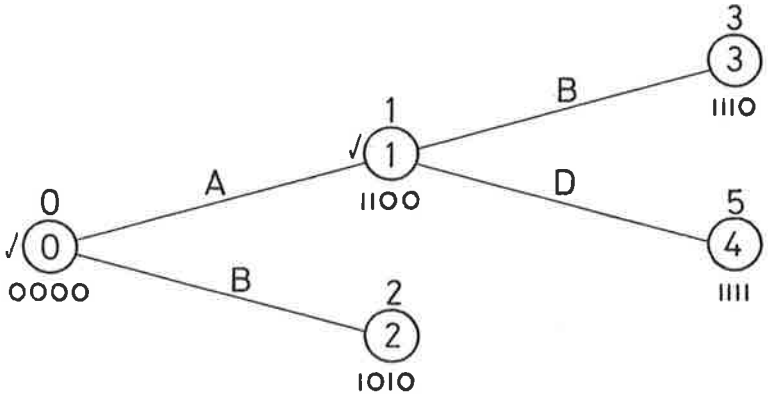


Fig 3.2 THE TREE CONTINUED

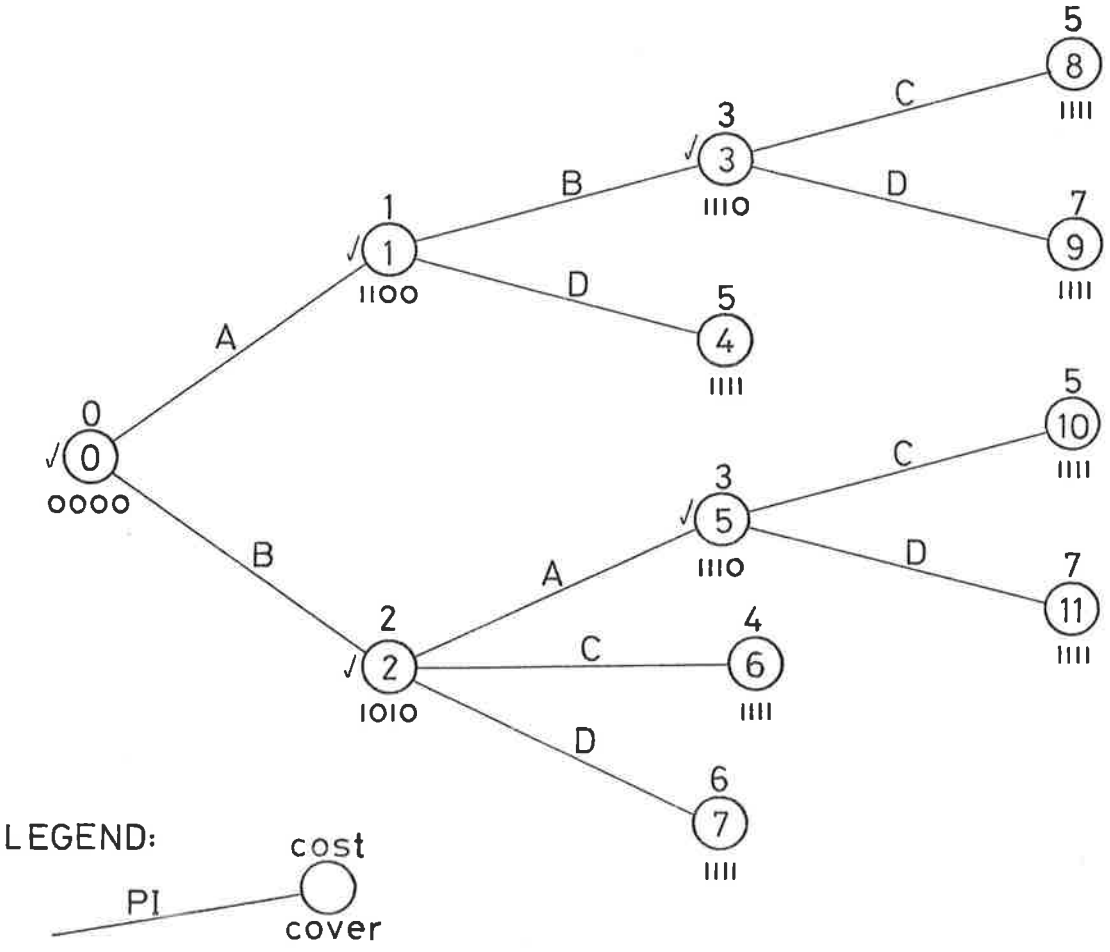


Fig 3.3 THE FINAL TREE

By way of contrast, the extraction method described by Roth (1959), Ewing et al (1961), Breuer (1972) and others differs in the following respects:

(1) It is 2-choice branching (a PI is selected or deleted) with cost remaining constant on one branch, and the depth of branching governed by the number of PIs. This is multi-choice branching with cost strictly increasing on all branches, and depth governed by the number of minterms, which for problems in 9 or more variables is usually a lower number (Meo, 1968).

(2) The criteria to be described for determining how to continue branching and for backtracking appear to be more efficient than those normally described for the extraction method. In particular, EPIs do not require any special consideration.

(3) The means for estimating a minimum cost along a new path and therefore being able to terminate redundant branching at an early stage is not described elsewhere, but is covered in the next section.

3.4.1 Incomplete Branching

It is obviously pointless to finish constructing the tree when it is not possible for a cheaper terminal node to be generated. This leads us to the concept of incomplete branching, in which some sections of the tree which can be shown to be redundant need never be formed.

Consider two nodes n_1 and n_2 such that node n_1 dominates node n_2 (i.e. the columns forming the cover of n_2 are a proper subset of those forming the cover of n_1), and n_1 costs less than n_2. Any set of PIs on a path leading from n_2 to a terminal node may be used to obtain a terminal node having no greater cost by using them on a path constructed from n_1, and so the sub-tree having n_2 as its root node is redundant.

To illustrate this point, consider the following example:

PI	cost	1	2	3	4	5	6
A	2	X	X				
B	1	X		X			
C	1		X		X		
D	1					X	X
E	1				X		X

TABLE 3.8 Second Example

The fact that this table is reducible by classical means is not relevant to the general technique. Applying the methods described above to table 3.8 yields the tree shown in figure 3.4. Examining nodes 3 and 5, we see that 5 dominates 3 (since 111000 is contained in 111100) and that 5 costs less than 3. Thus the set of rows (C,D) leading from 3 to terminal node 11 may be used on a path from node 5 to yield a cheaper terminal node.

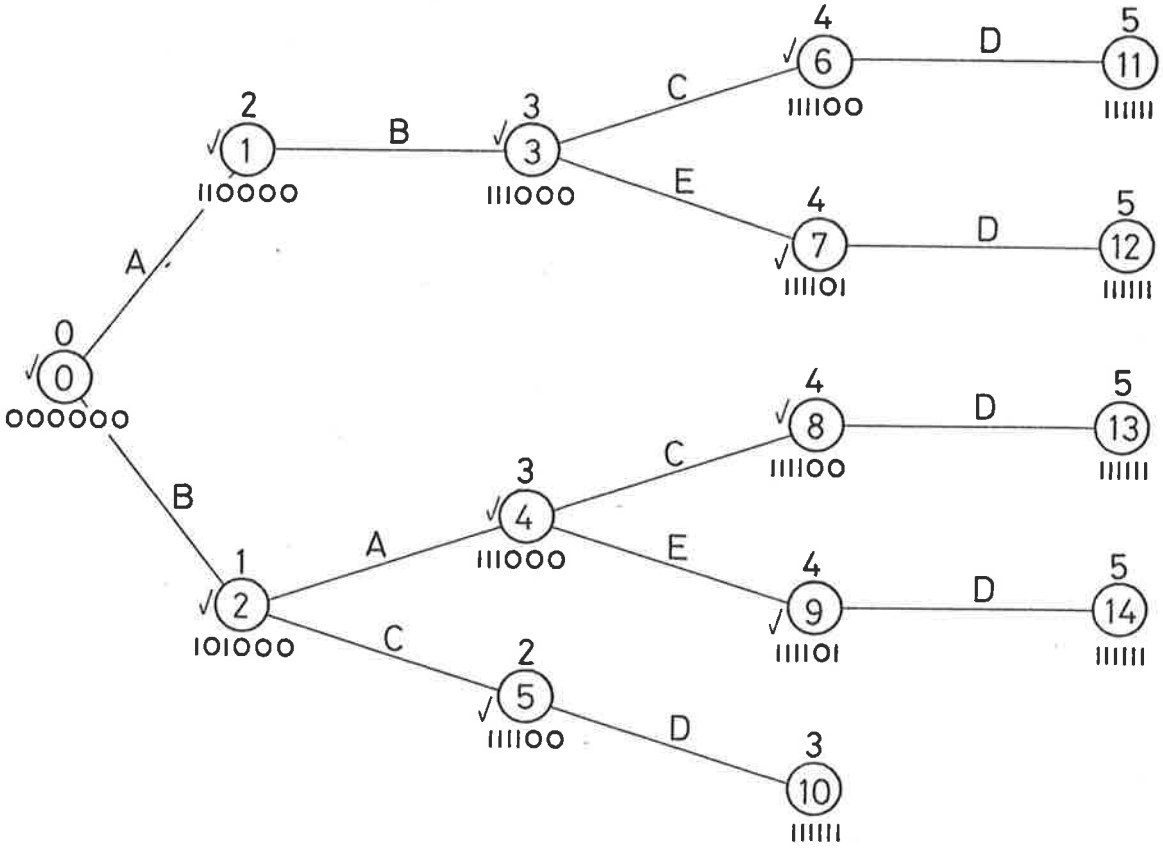


Fig 3.4 SECOND EXAMPLE

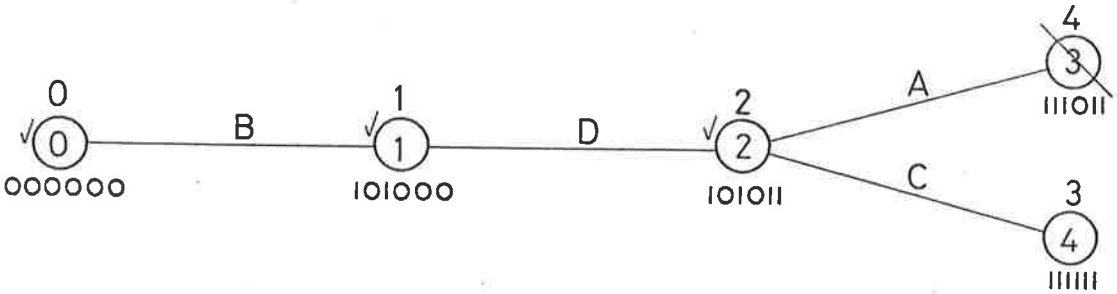


Fig 3.5 SAME PROBLEM WITH COLUMN ORDERING

In fact row C is not required, and terminal node 10 is obtained by using row D alone, and the solution set is (B,C,D).

This discussion leads to the following rule:

Rule 1 Delete a node n_2 (and its subtree if any) when there exists another node n_1 in the tree such that n_1 dominates or equals n_2 and the cost of n_1 is less than or equal to the cost of n_2.

Application of this rule in the present examples means that nodes 3 and 4 are deleted. It is possible that the application of Rule 1 can lead to a particular node having no successors (such as node 1): such a node can clearly not be on a solution path, which suggests the next rule:

Rule 2 Delete any node having no successors.

3.4.2 Ordering the Columns

Now every unused PI could be used for branching from a node, but of course such a tree would be enormous. A subset of PIs must therefore be used, but each such set must not preclude any possible cover which may happen to be the solution. The subset used for branching is chosen to be the set of PIs covering any uncovered column of the node concerned. This satisfies the above criterion since all covers must contain at least one of this set of PIs. This choice also ensures that the number of uncovered columns for each node must decrease monotonically, thereby ensuring that the algorithm terminates. Thus it is sufficient to select any uncovered column, but the question then arises as to which one; the answer to which is suggested by the following discussion.

Consider a PI table in which columns 1 to 4 are covered by 1,2,3 and 4 rows respectively. Let us assume that the rows are all totally independent so that all 24 partial covers are required in the tree. Now if the columns to be covered by the branches are chosen in ascending order, there will be one node at the first level, 2 at the second, 6 at the third, and 24 nodes at the fourth level (each containing columns 1 to 4 in its cover but representing a different combination of rows used to achieve this). Thus far, the tree will have 1+2+6+24 = 33 nodes, while if the columns were covered in reverse order, the number would be 4+12+24+24 = 64 nodes. Any other combination of column ordering yields a number between these two bounds, and so it seems logical to cover first the columns having the lowest weight.

This choice does indeed prove to be the best in general, and an attempt was made to refine it still further by taking into consideration the weights of covering rows with a view to causing the highest possible coverage in a successor node. The improvement afforded by such an approach was far outweighted by the extra complexity and computation time needed to implement it, and thus the order in which columns are to be chosen is simply expressed as:

Rule 3 Columns are to be covered in ascending order of weight.

Notice that this automatically forces essential PIs to be selected first, and so EPIs need receive no special consideration. The improvement afforded by application of this rule to the example of table 3.8 is illustrated by the tree of figure 3.5.

3.4.3 Ordering the Prime Implicants

In implementing a tree technique such as this, it is very desirable to restrict the size of the structure in such a way that computer storage is used as efficiently as possible. One such method was described in the previous section, in which the need to limit the number of branches in the early stages was demonstrated. A second way is to restrict the number of PIs available for branching by defining a hierachy in such a way that the earliest nodes in use are the ones most likely to be on the solution path.

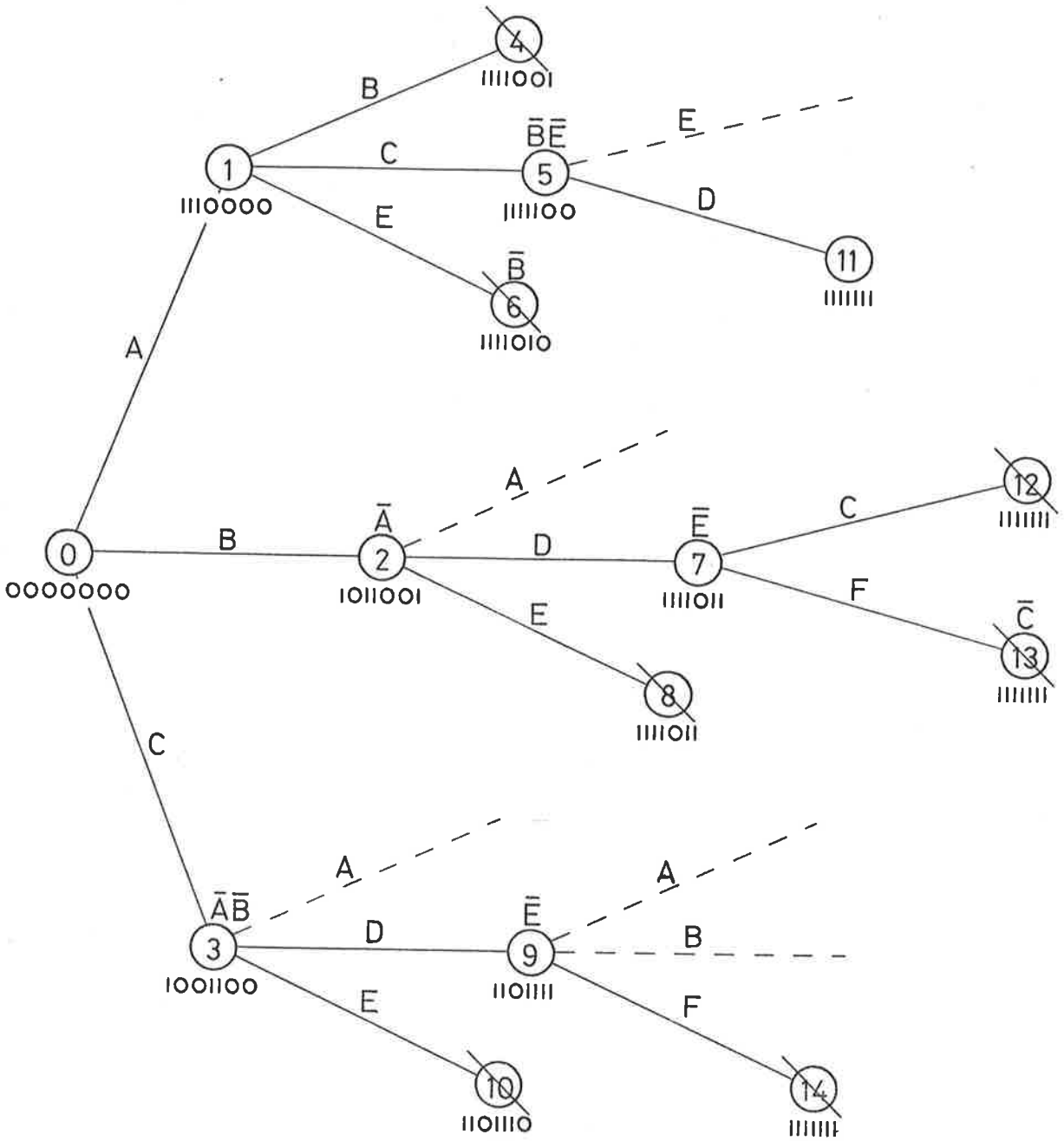
As a demonstration, consider the following example:

	1	2	3	4	5	6	7
A	X	X	X				
B	X		X	X			X
C	X			X	X		
D		X				X	X
E		X		X		X	
F			X		X		

TABLE 3.9 Third Example (all PI costs equal)

The resulting tree is shown in figure 3.6 which indicates the branching which may be avoided by suitably ordering the PIs.

This is achieved as follows: whenever a node is used for branching, such as node 1, the PI on the link from its predecessor (A) becomes unavailable for use in any sub-tree based on remaining unused nodes having the same predecessor, nodes 2 and 3 in this case. In this way, if the predecessor node appears on the solution path and the solution is to include the first-used PI (A was "used" before B or C in that node 1 was used before nodes 2 or 3),



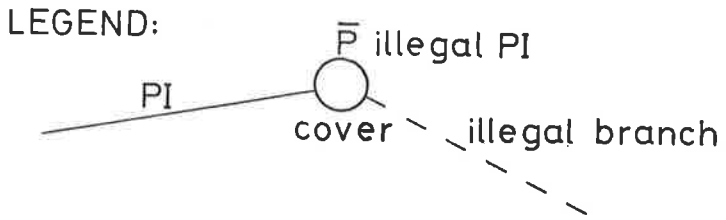


Fig 3.6 THIRD EXAMPLE

the path is forced to include the first-used node.

The same comments apply to the second-used PI, namely B, except that B is not illegal in the sub-tree based on A, and so paths including B may pass through nodes 1 or 2. Notice that nodes may be "used" in another way: when they are deleted. Nodes 4 and 6 for example exist only until node 7 is created, at which time they are deleted by applying Rule 1, and their associated PIs (B and E) become illegal on paths through node 5.

The operational rule may thus be expressed as:

Rule 4a Whenever an unticked node is ticked or deleted, add its predecessor PI to the illegal list of all other unticked nodes having the same predecessor node.

Rule 4b No PI may be used on a link if it appears in the illegal list of any node on its predecessor path.

3.4.4 Node Selection Strategy

The question now arises as to which unticked node to select to continue branching such that a solution is obtained as quickly as possible. Notice that the strategy used heavily influences the amount of storage required: if we choose nodes closest to the root, it is very likely that the tree will grow to fill the whole store before a solution is obtained; whereas if nodes furthest from the root are chosen, the tree will rarely consist of more than simply a path from root to terminal node. When the minimum cost solution is obtained, the tree degenerates into this form - all other nodes having been deleted by reason of domination or lack of successors. Thus if a large space is required its use will be only temporary, and so it is feasible to tailor the selection strategy in such a way as

to ensure that (a) the tree will fit into available store at all times, and (b) the solution is reached as quickly as possible.

Considerable effort was expended in deriving this strategy: various combinations of node attributes were considered as possibilities for a suitable selection function. It was discovered that if each node were examined in isolation, about the best selection rule was to choose the unticked node having the smallest average cost. Unfortunately, this (and any other similar strategy) is readily baulked if an ill-conditioned PI table occurs such that one particular column causes a jump in the selection function (average cost), no matter which PI is used to cover it. This means that a node in which this particular column is covered will not be selected until almost all earlier branching is completed, which is very inefficient.

Clearly the way around this predicament is to compare nodes so that the above situation can be detected. The method chosen is as follows:

Rule 5a Maintain a list, one entry per column, consisting of the cost of the cheapest node created which used this column for branching.

Rule 5b When selecting an unticked node, temporarily modify each node cost by subtracting the list cost corresponding to the column to be covered by this node. Select the one having the lowest modified cost, and if there is a choice, choose one with the lowest average cost.

Due to limitations in computer storage, this node selection strategy needs to be adaptive, and so when space is at a premium, the strategy becomes:

Rule 6 Select the node whose cover has the highest weight (number of covered columns).

The transition between the two strategies is accomplished by simply having a storage threshold, and monitoring the amount of store in use at any time. Notice that rule 6 ensures that once a particular node is selected, one of its successors must be chosen next, since they will each have covers with weights strictly greater than the selected one. This means that a single path to a terminal node will be set up, after which the amount of memory in use will remain substantially constant while nodes are deleted and added, until a sizeable sub-tree is deleted, and the first strategy can be re-employed. The level of the threshold may thus be determined by ensuring that once it is passed, there is adequate space left to construct a single path to a terminal node.

3.4.5 Further Refinements

It was found to be advantageous not to delete entirely nodes having no successors. This is because such nodes are often found to dominate nodes generated subsequently, and thereby inhibit redundant branching. The only information required to be saved is node cover and cost, and it is stored in a separate list. Naturally entries in the list may be obliterated by reason of domination by a later node.

It is possible to incorporate with each node a tally of input literals used so far. In this way, some control may be exercised over input literal fan-out so as to prevent excessive fan-out on any one literal for example. It was felt, however, that such considerations were not warranted, because in general fan-out limitations cannot be satisfied in two levels, and such a facility would have little practical importance.

3.5 Theorem and Proof

THEROREM: The algorithm as presented must terminate and produce an irredundant cover having the lowest cost.

PROOF: The number of branches from each node is equal to the number of crosses in a column of the PI table, which is less than or equal to the number of rows (r). The number of columns covered by each node is strictly greater than the number covered by its predecessor, and thus the number of links in any path from the home node is less than or equal to the number of columns (c). Hence the number of nodes in the tree is finite, being limited at least by $(1 + r + \ldots r^c)$, which means the algorithm must terminate.

If rules 1,2,4 are not applied, all irredundant covers must be generated. To verify this, assume that one such cover (the set of rows, S, say) is not generated. Since S is a cover, it must contain at least one element covering any given column. This means that the PIs on the branches from any node must include at least one element of S, since there is one branch for every row having a cross in the column being covered. It must therefore be possible to proceed from home node along a path consisting only of elements from S. Now since all paths finish on terminal

nodes, this particular one must end on a terminal node having a cover S' consisting of elements of S. But S is an irredundant cover which means that no proper subset of S is a cover. Hence S' is identical to S which contradicts the above assumption that S is not generated.

All irredundant covers are generated, therefore, and in particular one or more of these must have the least cost. To show that one such set remains to become the optimum solution when rules 1 - 6 are applied, consider the following.

Rules, 3,5,6 clearly do not affect the above arguments, since they only serve to constrain the order in which operations are performed, not the operations themselves. Rule 4 cannot inhibit the generation of any particular irredundant cover, S, by the following reasoning.

Consider the branches eminating from the home node. If the first-used row is in S, follow that link to the next node, where there are no illegal successor rows. If the first-used row is not in S, either the second-used row is in S or it isn't. Again if it is, follow that link to its successor node where the only illegal successor row is not in S. If it isn't continue in this way until a suitable link and successor node is found. (At least one such link must exist because elements of S cover every column). Having selected the next node, use the same process to select the following one. In all cases, the illegal successor rows include no member of S, by construction, and the path so formed contains only members of S. If the final node is not a terminal, it must contain an uncovered column; but then an element of S could be selected to cover this column, creating

a new node - thus the final node must be terminal, and represents a solution set S' consisting solely of elements in S. Because S is irredundant, S' must be identical to S, and so rule 4 does not inhibit the generation of any irredundant cover.

Let S be a solution set (irredundant cover having minimum cost), and consider the associated path eminating from home node and constructed as above. If one of the nodes (n_1) on this path is deleted by another node (n_2) by application of rule 1, define the following quantities:

A = set of rows on path to n_1

B = set of rows on path to n_2

C = S - A = set of rows in S and not in A

S' = B $\cup$ C

cost(X) = the cost of the set of rows X

M(X) = the set of columns covered by the set of rows X

Now $M(A) \subseteq M(B)$ and cost (A) $\geq$ cost (B) since n_2 deletes n_1 (rule 1). $M(S) = M(A) \cup M(C) \subseteq M(B) \cup M(C) = M(S')$ and hence S' is a cover. cost (S) = cost (A) + cost (C) $\geq$ cost (B) + cost(C) = cost(S') but cost(S) $\leq$ cost (all covers), and so cost (S) = cost(S'). Therefore the application of rule 1 can inhibit the generation of a solution set S if and only if there exists an alternative solution set S'. This means that a solution set must be produced.

Finally, rule 2 cannot affect this result, since no nodes in the solution path qualify for deletion under this rule.

Q.E.D.

4. MULTIPLE OUTPUT CIRCUITS

4.1 Broad Outline

The discussion so far has concerned combinational logic networks with only a single output, but it will now be shown how the optimisation technique developed may readily be extended to circuits having more than one output.

In this section it will be assumed that it is desired to minimise a 2-level combinational circuit consisting of NAND gates only. Realisations employing NOR gates only are readily obtained by considering dual problems, but mixed gate representations (eg NAND/NOR or AND/OR) are specifically excluded from the discussion. The reason for this restriction is to ensure that the classical assumption that "there exists at least one minimum-cost two-stage network in which the corresponding expressions for the output functions are all sums of multiple-output prime implicants" is in fact correct (McCluskey, 1965). Counter examples to show that such an assumption is not valid for mixed gate circuits was demonstrated in 1968 by Weiner and Dwyer, but this point has often been overlooked in the literature.

Multiple output optimisation is a more complex problem than simply optimising the network for the various outputs separately, because it may be possible for two or more outputs to make use of common primary gates. It also happens that the primary gates in the optimum solution need not represent prime implicants of any of

the output functions, although they normally correspond to prime implicants of a function formed by taking the conjunction of any number of the output functions. Bartee (1961) described a convenient way of computing all of these terms which for convenience are called multiple output prime implicants. Once they are found, an extended PI table is set up and solved in precisely the same manner as previously described.

Unfortunately the solution so obtained is only optimal with respect to primary gates: no consideration is given to the secondary gates. The size of the secondary gates required may be reduced by detecting and removing redundant connections. This is achieved by using the same method to solve a reformulated problem in which the PI list is modified and used to construct a new PI table. This second-pass is always very much quicker than the first, and the solution it generates has the minimum number of connections to secondary gates while having the same primary gate cost.

4.2 Generating the Prime Implicant Table

The procedure for dealing with multiple output prime implicants is adequately described by Bartee (1961), and a special class of problem in which care must be exercised is pointed out by Mage (1970). The following brief description is included as an indication of how the present author has chosen to implement the method.

Consider a multiple output problem in which there are n input variables and m output variables. We construct two lists of terms ("minterms" and "dontcares") such that every term consists of (n+m) literals. Each actual minterm of a given output function is represented in the "minterm" list by a term in which the first n bits are a replica of that minterm, and in which there is a single 1 in the remaining m bits corresponding to this particular output - all other (m-1) bits being 0. Actual dontcare terms are represented in a similar fashion in the "dontcare" list. Next, all terms in either list having their first n bits identical are used to create new terms for addition to the "dontcare" list. These new terms are such that the first n bits remain the same, but every combination of 1's (including none at all) appears in the relevant positions in the m bit section. These two lists are then treated as though they were the minterm and dontcare lists of an (n+m) input variable single output problem - the PIs are generated and the PI table is set up exactly as before.

Notice that the way in which the "dontcare" list is augmented ensures that the lower m positions of every PI will be composed of zeros and dashes only. Each dash corresponds to an output function for which the upper n positions represents an implicant.

As an example, consider the following problem having 3 inputs (a,b,c) and 3 outputs (X,Y,Z)

inputs			outputs		
a	b	c	X	Y	Z
0	1	0	1	-	1
1	0	0	0	1	0
1	0	1	0	-	1

TABLE 4.1 Multiple Output Example

The four 1's in the output specifications lead to the generation of the following 4 terms in the "minterm" list:

1. 010001
2. 010100
3. 100010
4. 101001

The two dontcare specifications (-'s) lead to the terms 010010 and 101010 in the "dontcare" list. Now extracting the terms starting with 010, we find that there are three, and so the following five terms must be added to the "dontcares":

010000
010011
010101
010110
010111

Similarly 100010 forces the addition of 100000, and the two terms starting with 101 yield the last two, so the full "dontcare" list becomes:

010000
010010
010011
010101
010110
010111
100000
101000
101010
101011

From these two sets of terms, the PIs are found to be

A. 0 1 0 - - -
B. 1 0 - 0 - 0
C. 1 0 1 0 - -

and the resulting PI table is:

	1.	2.	3.	4.
A.	X	X		
B.			X	
C.				X

4.3 The First-Pass Solution

The solution to the extended PI table consists of a set of multiple output PIs, and is obtained in the same way as previously described. Notice however that in assigning PI costs, only the first n positions are examined - they represent the presence or otherwise of actual literals. Each term in the solution represents one primary gate of up to n literals, and a circuit may be realised by simply wiring each such gate to each

secondary gate having a dash in the corresponding position of the lower m bits of that term.

Thus in the previous example, term A ($\bar{a}b\bar{c}$) will form an input to each of the three output gates, term B is used for output Y only, and term C is used for the two outputs Y and Z. The solution may be written as:

$$X = \bar{a}b\bar{c}$$
$$Y = \bar{a}b\bar{c} + a\bar{b} + ab\bar{c}$$
$$Z = \bar{a}b\bar{c} + ab\bar{c}$$

This is the first-pass solution, in which the cost of the primary gates is optimised, but clearly all three of the terms appearing in the expression for Y are not required. Such redundancies may not always be eliminated by simply examining the solution expressions term by term. Consider the following example (modified from one presented by Bartee):

$x = \sum_m 1,6,7,9,11,14,15 \quad y = \sum_m 4,5,6,7,10,12,13,14,15$

$z = \sum_m 4,5,10,11,12,13,14,15 + \sum_d 1,9.$

The first-pass solution is found to be:

$$x = bc + \bar{b}\bar{c}d + acd$$
$$y = bc + b\bar{c} + ac\bar{d}$$
$$z = acd + b\bar{c} + ac\bar{d} + \bar{b}\bar{c}d$$

The optimal solution involves replacing the terms (bc) and ($b\bar{c}$) in the expression for y by the new term b : this type of simplification is more difficult to detect, but is readily accomplished by the method to be described in the following section.

4.4 The Final Solution

Once a set of multiple output prime implicants is obtained as a first-pass solution, all others in the list are removed except those with zero cost. Each term remaining in the list is divided into component terms having exactly one dash in the lower m positions. In the previous example, where n = 4, m = 3, (-001-0-) is replaced by two terms, (-001-00) and (-00100-),

This new "PI" list is used to construct a second "PI" table, which is solved to yield an irredundant cover containing the minimum number of terms. Notice that the way in which the PI list is altered ensures that each term now represents one input to a secondary gate, and so minimising rows will minimise the number of secondary inputs.

The terms forming the final solution may be obtained from the second-pass solution set by recombining component terms having their first n positions identical. Dashes in the lower m positions are interpreted as before, as representing which output gates are wired to each primary gate.

Because the algorithm relies on PI costs being strictly positive, it is found convenient to leave in the second PI list any term whose cost is one rather than zero (usually single-literal terms). Such terms should only be the ones which the designer regards as having zero cost, so that their addition to the first-pass solution set does not disturb the optimality of the solution.

Assuming that minimisation is required, the solution to the preceding example becomes:

$$X = bc + \bar{b}\bar{c}d + acd$$
$$Y = b + ac\bar{d}$$
$$Z = acd + b\bar{c} + \bar{b}\bar{c}d$$

It is conceded that this approach to multiple output circuits may not always yield the optimal solution - particularly if the criterion is to minimise packages, but the solution obtained will always be very close to the optimum. The principal reason for presenting it is to demonstrate the versatility of the main algorithm, in that by suitably modifying the input data and interpreting the results it can handle more than simply single-output networks.

5. COMPUTER PROGRAM

5.1 Introduction and Environment

The computer program listing which appears in the appendix was not written as a single entity, but rather is the result of a great many extensions, modifications and deletions made during the development and testing of various ideas. As a consequence, the coding is sometimes clumsy and circuitous: no attempt was made to rewrite or streamline it. The reason for including it and this description are to illustrate a practical implementation and to provide designers with a useful aid.

The installation at which this work was carried out consisted of a Control Data Corporation 6400 computer with batch and low speed interactive facilities. The memory available was up to 32K (32,768) words of 60 bits each. Extensive use is made of the long word length to pack related information into one location, and this would be one of the main restrictions in transposing the software to a computer system having a shorter word length.

The language used is the CDC version of FORTRAN Extended, which contains some minor departures from the standardised ANSI FORTRAN. For reasons of efficiency and expediency, some bit and character manipulation routines are written in assembly language (COMPASS), but at the expense of execution time, the whole program could be converted to standard FORTRAN.

In fact the program is readily portable between CDC 6000 series or CYBER computers without amendment: An earlier version was used successfully on the CYBER computer at Melbourne University.

FORTRAN was chosen because it is widely used and understood, and is reasonably machine independent. It is in general much more efficient than other high level languages in this type of application: in fact CDC FORTRAN Extended using the "Optimisation" parameter has been shown to be second only to assembly language in execution efficiency (Adelaide University, Computer Centre Newsletter B22, 1975). This version also provides in-line shifting and mask generating functions which are used extensively in this program.

5.2 Facilities and Limitations

This program is intended for interactive use, where the user controls the execution depending upon interim results and other information, but if it is used to perform well defined tasks not requiring user decisions, it may simply be run as a batch job.

The number of input variables for a single output problem is limited to ten, and for multiple output problems the total number of input and output variables may not exceed ten. This limitation was imposed solely for programming expediency: one computer word may represent ten display characters. In fact it would have been pointless to raise this limit, since the program exists mainly to demonstrate the method, and machine limitations severely restrict the type of large problems which can be optimised.

The main facilities provided are as follows:

(1) Data may be input either on-line or from an existing file, and may be in a variety of formats including partly minimised form.

(2) Random problems may be generated for testing purposes.

(3) A sum-of-products or product-of-sums expression may be expanded into its canonical form.

(4) Prime implicants may be computed using either Hwa's method or the consensus method, and the PI table printed if required.

(5) A quick covering set may be obtained using a "heaviest first" (Karnaugh) method with two options on selection priority.

(6) The previously described tree algorithm may be invoked to compute an optimum solution, but may be terminated prematurely to provide an irredundant cover if an approximate solution only is desired.

(7) Extensive error checking may be performed during the execution of the main algorithm and the final solution may be checked to ensure it is in fact an irredundant cover of prime implicants.

5.2.1 Input Data Format

Data for single output problems is read in as two lists consisting of minterms followed by dontcares. These lists may be decimal, octal or binary numbers, and in the case of binary input may be in partly minimised form. No ordering of the numbers is required, repetition is allowed, and cononical terms appearing in both lists are resolved to be minterms.

Data for multiple output problems may be a series of list pairs, one pair per output and conforming to the above requirements, or else a complete partly minimised multiple output truth table may be entered in binary format.

Any problem may readily be converted to its dual, for use in cases where a product of sums realisation is desired or where falseterms and dontcares are specified. In the event that falseterms and minterms are specified, they may be entered as though they were minterms and dontcares. By listing the dontcares and minterms of the dual problem, a file is obtained which may readily be edited to become a data file for the true problem.

5.3 Program and Data Structure

In general, different tasks are confined to separate subroutines so that the coding is more readily understandable. Extensive use is made of labelled common blocks rather than using long formal parameter strings for communication, and variables within a block are generally assigned the same names wherever that block appears.

CDC CYBER and 6000 series machines require programs to specify their maximum field length requirement in advance. This applies no matter what language is used, and so any dynamic storage allocation scheme must work within this bound. The approach adopted here is to use unlabelled common ("blank common") for the working area, because it enables access to all the store remaining after the program modules are loaded. A limited dynamic storage facility is provided by allowing the user to change the size of this area during program execution, up to the system defined limit.

5.3.1 Lists

Wasted storage is kept to a minimum by using variable length lists, with a system of pointers to record either first and last locations or first location and length. Normally one word is used for each element, which may have two or more quantities packed into its sixty bits. The PI table in particular is packed as densely as practicable, having one cell per bit and the rows padded out to the nearest word boundary.

5.3.2 Boolean Terms

Initially, minterms and dontcares are represented in individual words by their numeric equivalents, but the representation used in the bulk of the program needs to accommodate the ternary choice per variable: false, true and not present (0,1 and -). Such terms are stored two bits per variable in the left most 21 bits of a word, the most significant bit being normally zero but useful as a flag bit. The right hand 39 bits are used for temporary information associated with the term, such as PI cost, or column weight. The two-bit codes were arbitrarily chosen to be:

code	meaning
00	0
01	1
10	-
11	N.A.

Other code assignments may result in some operations being performed faster, but little real advantage could be gained overall by using a different assignment, since the tree algorithm requires no manipulation of these terms.

5.3.3 Nodes

Problems arise in storing and handling nodes due to the large number involved, their variable information content, and the fact that they are being continually created and deleted. The data structure chosen is to use a variable length node index table packed at the high end of store. Each index entry consists of two words, packed with as much information as possible, including a pointer to the remaining details recorded in the main store.

The byte sizes chosen are sufficient for a working area of 131K(17 bit addresses): notice that only 15 bits are needed for predecessor and successor node pointers, since these numbers refer to positions within the index list which would never exceed 32K words.

The PI number on each link is in fact stored with the terminating node, and since each node contains pointers to its predecessor and successors, no special provision is made for link information. Figure 5.1 is a memory map of the blank common area, and details of the two types of entry are represented in figure 5.2.

5.4 Subroutine Abstracts

The following brief description of each subroutine is aimed at describing the functions performed, outlining the methods used, and pointing out any obscure points. Further details of their actual operation may be obtained from the comments embedded in the listing in the appendix.

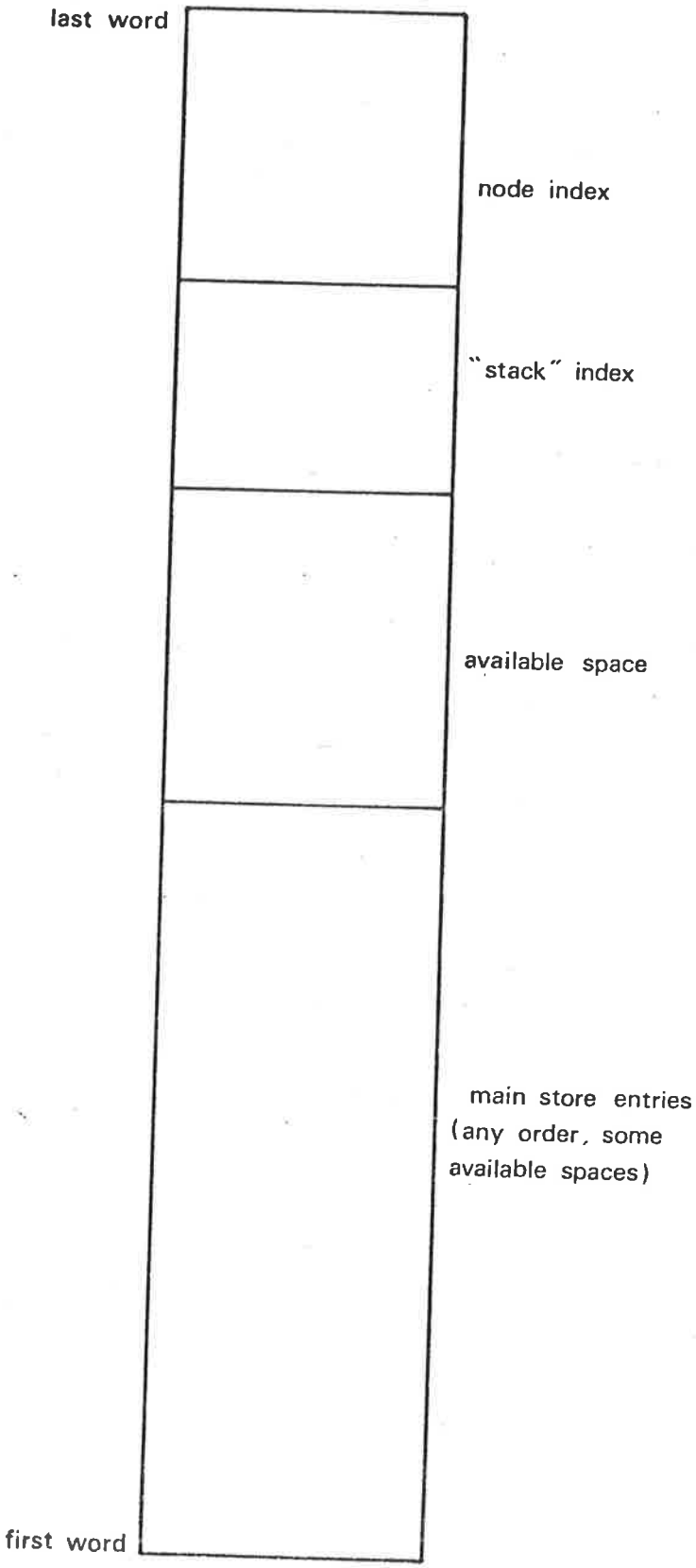
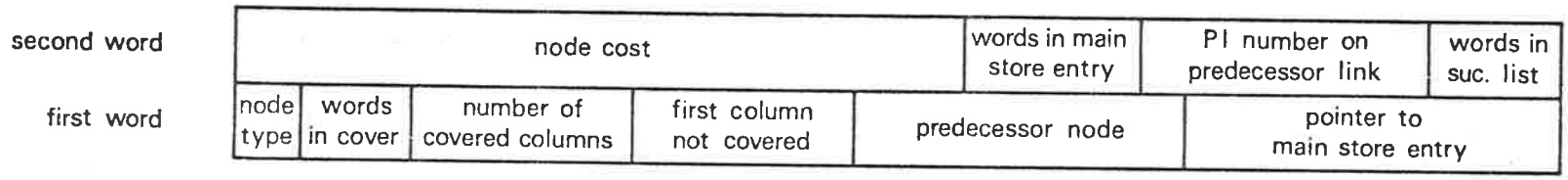


Fig 5.1 BLANK COMMON MEMORY MAP



INDEX ENTRY

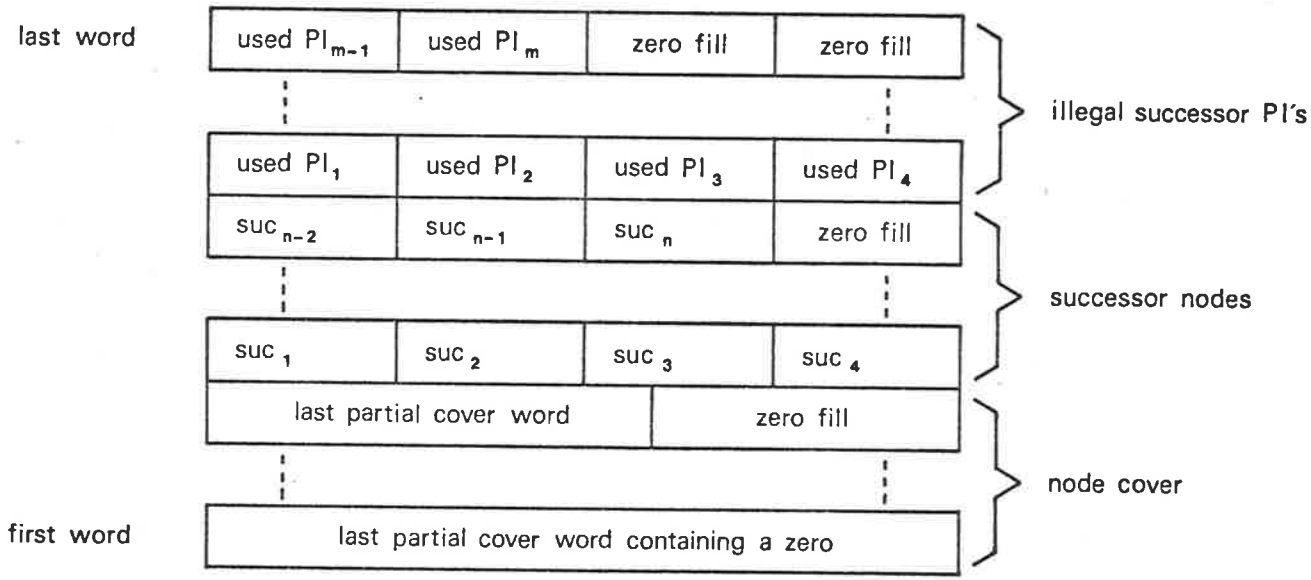


Fig 5.2 MAIN STORE ENTRY

5.4.1 ADD adds a new node to the tree. The parameter CUSLMAX indicates roughly how much extra space the new node needs to leave in order that a "used sibling list" may be added later.

5.4.2 CHECK checks a solution set of terms for any glaring errors. If it is completed successfully, all terms are prime implicants, none is redundant, all minterms are covered and no extra term is covered. If the redundancy check fails for a multiple output problem, it is not interpreted as an error because the offending PI may have been included to reduce secondary connections.

5.4.3 CONSENS computes a complete set of PIs by iterative consensus, and may be applied either to a new problem or to a set of (prime) implicants generated in some other way. Its use in this program is virtually obsolete, as HWA performs the same function more efficiently. The method is to compare each term with all subsequent terms in an attempt to form a new term by consensus. As each term is finished with, a pointer is recorded to indicate its last comparison, so that it may later be compared only with new terms.

5.4.4 CONTROL is by far the most complex routine as it controls the whole program. Basically, it accepts instructions from the user, compares them with its instruction set, and then performs the required tasks, which are usually a simple parameter change or subroutine call. CONTROL may also be called by

several routines as they are executing, through the PAUSE entry point. As its name implies, this facility causes the main program to pause while the user performs other tasks through the control routine. This is mainly used as a debugging aid, so that the tree structure may be examined during execution, but it also enables an approximate solution to be extracted or allows a self-timing feature to be used. The User's Handbook (Appendix B) contains details of the various commands and their functions.

5.4.5 DELETE deletes a node from the tree. If the node concerned has any successors, they are deleted too, and if its predecessor node is left with no successors, it is deleted also. This "backup" mode of deletion continues as long as each predecessor has no other branches. The node index list is continuously compressed so as to leave no holes, but this makes some of the housekeeping complicated enough to merit its own subroutine, MOVE. If the terminal node is deleted, there is temporarily no terminal node in the tree, and the terminal cost is incremented by one to ensure that one having the same cost as the old one may be created if a cheaper one does not eventuate.

5.4.6 FAULT is simply a link routine from the fault-finding routine (POS) back to CONTROL. It stops all processing to allow the user to investigate more details.

5.4.7 HEADING simply prints a heading on the print out, but also contains two service routines used by READATA. Entry point READIN reads and records input data and is basically a character manipulator. Entry SORTHEM sorts the minterms and dontcares into ascending order, and records them so that the problem may be regenerated without re-reading. For multiple output problems, the dontcare list is augmented with sufficient new terms to ensure that every legal combination of outputs is covered for each input term.

5.4.8 HEAVY finds a covering set of PIs (which represents an upper bound on the cost of an optimum solution) by using a single-pass "heaviest first" algorithm. PIs are selected according to two criteria: number of crosses in that row of the PI table (weight) and PI cost. The size of the formal parameter WCOEF determines which criterion predominates.

5.4.9 HWA computes a complete set of PIs for a given problem using Hwa's method. The minterm and dontcare lists must be in the format produced by READATA, RANDOM or RANDY. The algorithm used is described in section 3.2.

5.4.10 INDEP establishes whether a new node is independent of the existing nodes during construction of the tree. The new one may be deleted immediately, or it may cause existing nodes to be deleted. Once this happens, the new node is certain to be added to the tree, and so it is no longer tested for deletion.

5.4.11 MOVE performs the necessary housekeeping when a node index entry is relocated during a delete sequence.

5.4.12 NET is the main routine to implement the tree structuring algorithm. Its operation is described in section 3.4.

5.4.13 OUT contains three output routines, the first of which prints a solution set of PIs together with some statistical information about the solution. The DUMPF entry point dumps an area of store by printing it in one of several formats. The LISTER entry points heads a routine to print lists of minterms, dontcares or PIs, as required.

5.4.14 PITABLE prints a copy of the PI table.

5.4.15 POS subjects the tree under construction by NET to a thorough series of tests to ensure that no faults are present. It is very useful as a debugging aid, and also provides a summary of the current size of the tree, broken down into the total numbers of each node type.

5.4.16 PROGRAM is a figure-head program which does little more than pass control to CONTROL. This structure is used because the controlling routine utilises a pause and return facility which may not be incorporated in the main program block because the RETURN statement is illegal.

5.4.17 QMTONET sets up tables for NET. The minterms are sorted into ascending order of weight before constructing the PI table, and pointers are added to the minterm list to shorten the searching required for the covering PIs. PI costs are also calculated at this time.

5.4.18 RANDOM generates a "reasonably random" problem for use in testing the various procedures. Each of the 2^n canonical terms has a probability of .5 of being selected as a minterm, and a .125 chance of being a dontcare. Problems having this distribution of terms were found to be usually more demanding with respect to algorithm efficiency.

The entry point REPEAT is used to redirect control to the appropriate routine when a problem is to be regenerated.

5.4.19 RANDY generates a "random" problem in the sense that the numbers of minterms and dontcares are first chosen at random, before the two lists are filled by randomly selected terms.

5.4.20 READATA and its alternate entry point MREAD supervise the reading and preparation of input data for user-specified single and multiple output problems. Most of the work involved is relegated to the various routines embedded in subroutine HEADING. Entry RECREAT is used for regenerating either type of problem.

5.4.21 RSWAP converts a problem to its complement, so that the solution may be interpreted as a sum of product terms (in which every literal is to be complemented).

5.4.22 RTOQM reformats a problem to the format suitable for CONSENS or CHECK.

5.4.23 SELECT selects an unticked node in the tree to continue branching. Either of two strategies is used, depending on the amount of storage currently in use.

5.4.24 SHORTY compresses the storage being used during the construction of the tree into one contiguous block so as to provide the largest possible empty block. Its auxiliary function is to move the entry for one particular node so as to be adjacent to the unused block: this is required whenever more information is to be added to that already present. The shortening of store is only used when absolutely necessary so as to minimise the time used in this purely housekeeping exercise.

5.4.25 SOLUTIO extracts a solution set of PIs from the tree by following the path from the terminal node back to the root node. Such a set is only optimal after NET has been executed, but is usually near minimal well before all possible branches have been exhausted. Notice that because a terminal node does not exist initially or may be deleted, SOLUTIO may not always be able to return a solution.

5.4.26 CMSTORE contains all of the machine language routines used by the FORTRAN routines. It is loaded as a program so as to record the load-time field length before the information is lost by executing any code produced by the FTN compiler. As soon as this is done, PROGRAM begins executing in the usual way. By this means, the program is able to utilise all of the available store automatically.

6. RESULTS

6.1 Introduction

The "result" of a study such as this is the algorithm itself. Naturally, for this to be a meaningful result, the algorithm needs to be implemented as a computer program, which serves to provide the two functions of verifying that the method works and providing a useful design aid. In the following sections, some indication is given of the powers and limitations of the program listed in the appendix.

It is difficult, if not impossible to predict in advance how long a given problem will take. It might happen for example that a particularly complex 7 variable problem takes a hundred times as long as some user-defined 10 variable problems. However, for most practical applications, the program provides a quick and easy solution to both single and multiple output network optimisation.

6.2 Time and Storage Requirements

It has been shown that for a 10 variable problem, the number of PIs may be as large as 7680 (Meo, 1968). The number of minterms of such a problem is likely to be around 900, and hence the PI table would require 7680 X 900 cells. Now even if no bits were wasted, this table would require more than 110K (60 bit) words of store, which exceeds the capacity of the main memory for most present day computer installations.

Clearly then the program cannot be expected to handle every arbitrary problem of 10 variables.

PI tables for smaller problems may be accommodated, but the working area required for the tree manipulation must be large enough to permit a reasonable growth before the strategies are weakened by space becoming critical.

Although increasing the store available may in general be expected to reduce the time required for execution (because the more efficient strategy is in use longer), there is a well defined lower limit on the execution time for any particular problem. It was found that using a memory size greater than 32K was rarely warranted, as any problem requiring so much space was likely to require more than 30 minutes central processor time. This was a self imposed time limit arising out of consideration for the many other users competing for the computing facilities.

Figure 6.1 shows the time and space requirements for a representative test problem. As anticipated, execution time is roughly constant for large field lengths, but as field length is reduced, the time increases, until for a very small store the time required is very large.

Figure 6.2 demonstrates the effectiveness of the switch in strategy which occurs as the storage limit is approached. The one employed when space is critical is designed to drastically curtail the growth of the tree by causing nodes to be deleted. Once this is done, sufficient area is freed to enable the original strategy to be employed which leads to the final solution more quickly.

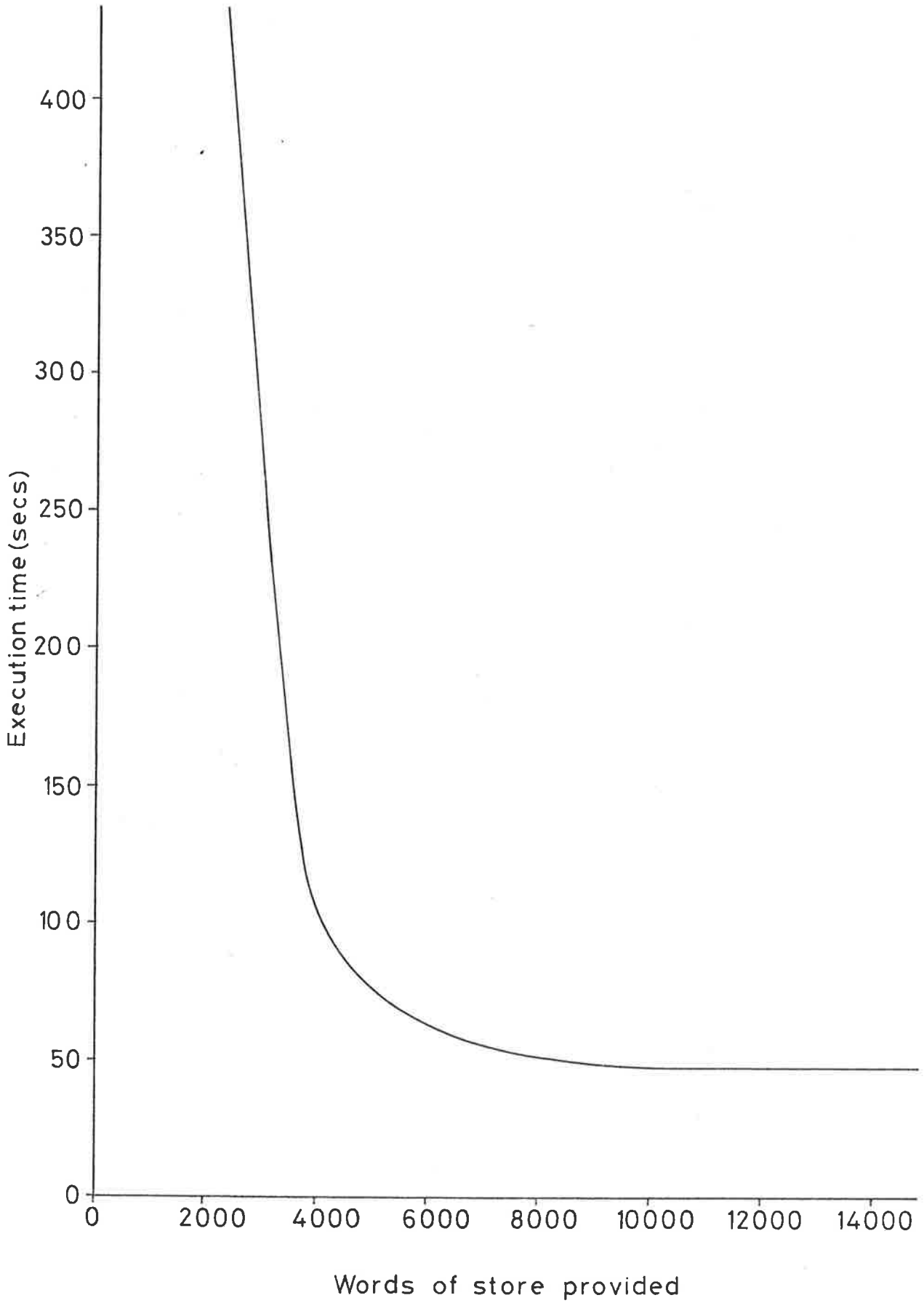


Fig 6.1 TIME / STORE TRADE-OFF

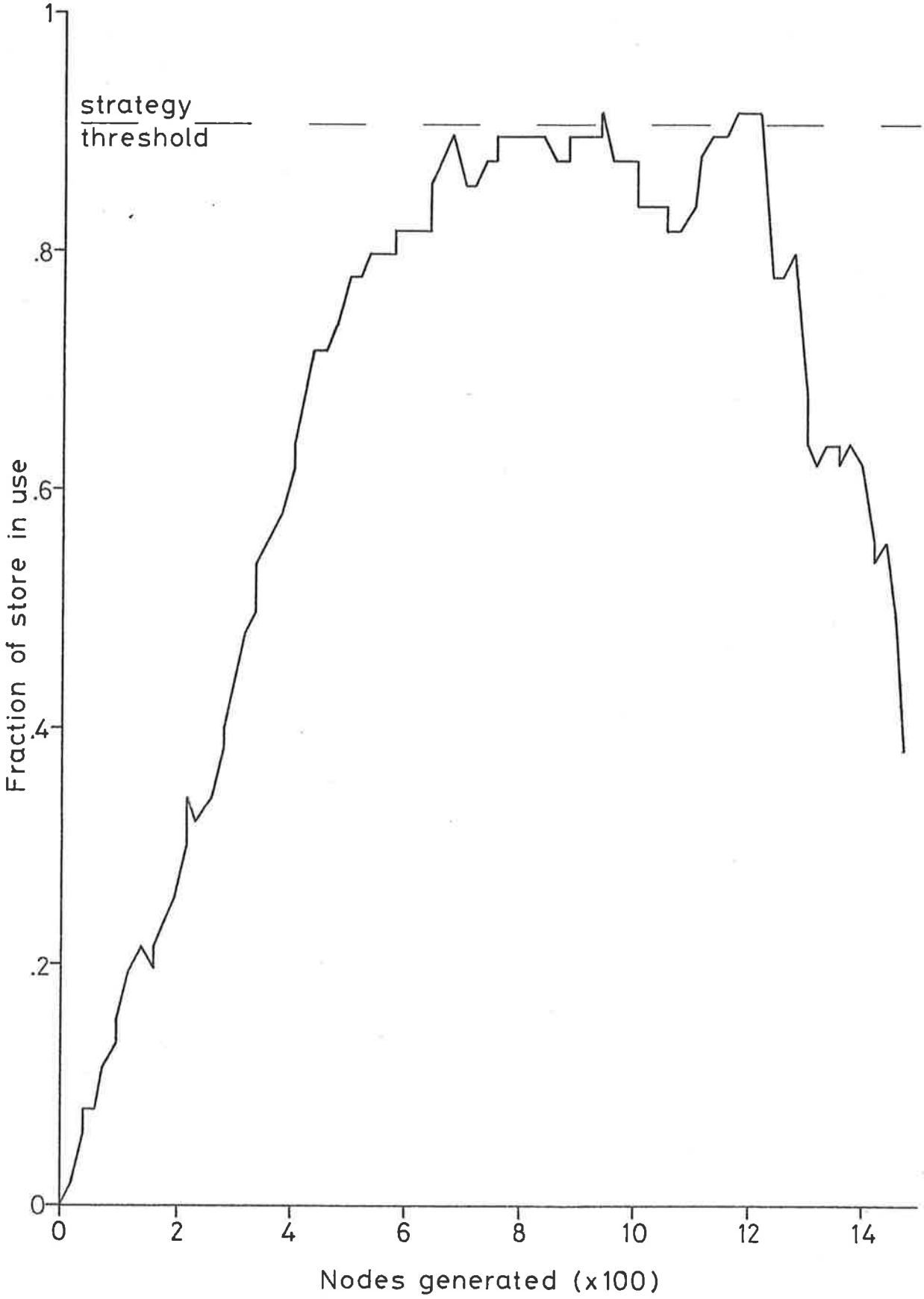


Fig 6.2 STORE UTILISATION

Finally, figure 6.3 displays the relationship between the number of nodes generated and the elapsed time for a typical problem. This curve is increasing, indicating that as the tree grows, it takes more time to perform the various tests on it between generating new nodes. In fact it is expected that the intervening time intervals should be a linear function of the size of the tree, and so this curve should approximate a quadratic, which in fact it does.

6.3 Typical Performance Figures

Randomly generated problems (using subroutine RANDOM) were found to be a severe test, because there is usually a higher degree of symmetry and less complexity in practical circuits. Table 6.1 was compiled by solving 20 such examples for each value of n, and averaging the results.

number of variables (n)	4	5	6	7	8
number of minterms	8.0	16.4	33.5	65.7	129.3
number of dontcares	2.5	3.9	8.3	15.2	31.2
number of PIs	5.0	12.1	29.6	66.9	148.1
execution time (secs)	0	0	.2	16.0	85*

Table 6.1 Average Execution Times
(* denotes typical time)

It may be seen that any problem of less than 8 variables presents little difficulty, in fact the times for 4 and 5 variables were too small to be measured. The figure quoted for 8 variable examples is a "typical" or median rather than a mean, because a small proportion of the generated examples took a very long time, exceeding

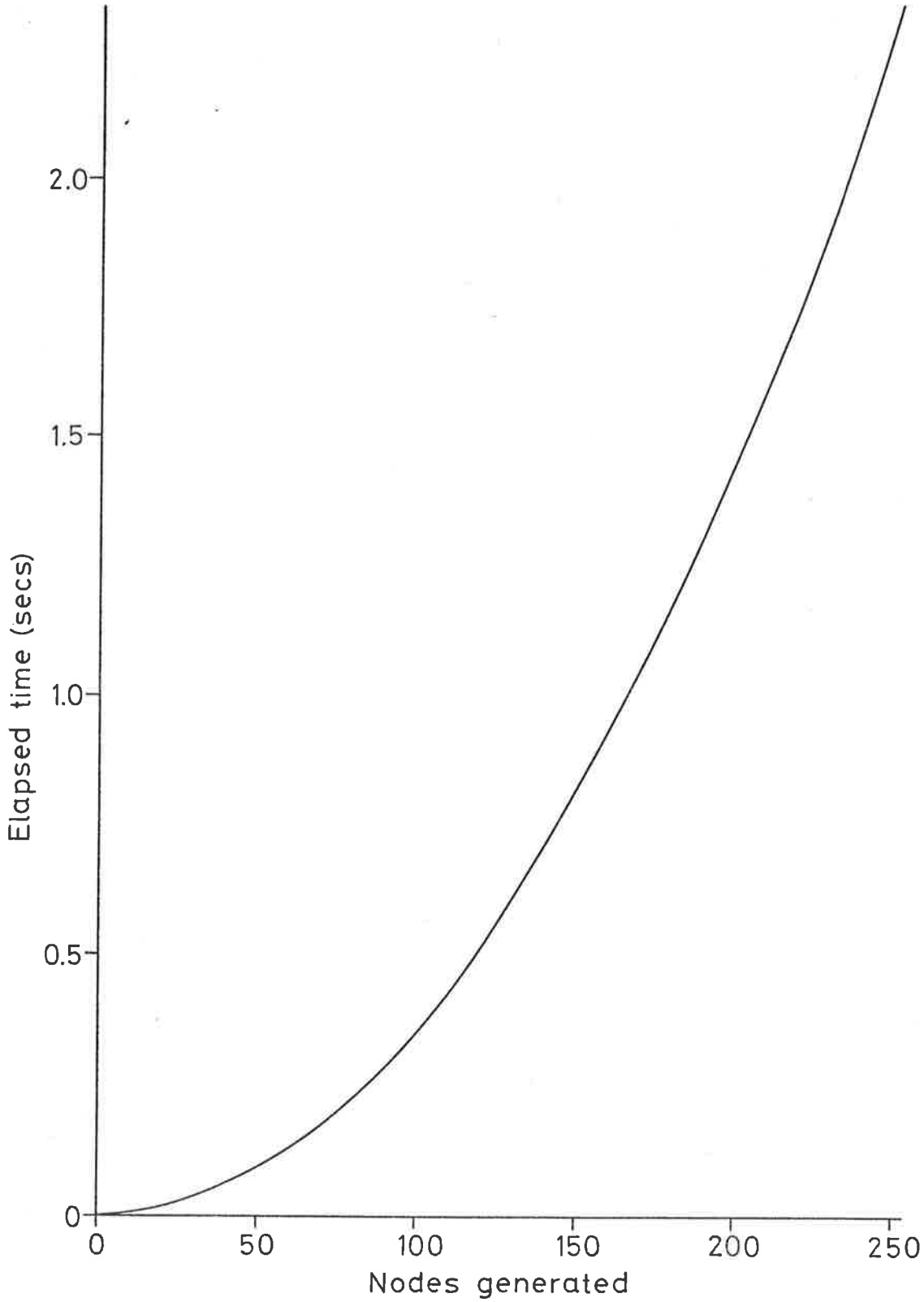


Fig 6.3 RATE OF TREE GROWTH

an arbitrarily assigned time limit of 256 seconds. Examples generated in this way in 9 (and 10) variables require a much greater memory size in order to achieve reasonable execution times, and so no results are quoted for these cases. Notice, however, that in all cases an approximate solution is still quickly available which is at least as good as a heaviest-first one and which becomes cheaper, the longer the program is allowed to run.

The results for six particular examples are summarised in table 6.2. These are problems 1 to 6 of appendix D, and were arbitrarily selected to indicate the order of magnitude of typical execution times.

Example	1	2	3	4	5	6
Input variables	9	9	10	10	5	5
Output variables	1	1	1	1	5	5
Number of minterms	90	101	108	53	65	55
Number of dontcares	87	42	90	642	235	153
Number of PIs	137	115	128	321	76	53
Time to generate PIs	.5	.3	.5	25.9	1.2	.6
Total time (secs)	46.0	2.1	1.8	34.7	39.1	1.4

Table 6.2 Further examples

6.4 Comparisons with other Programs

The obvious question to be considered at this stage is just how does the algorithm compare to presently available ones. It is virtually impossible to do quantitative analysis on algorithms per se, and so it was decided to compare their performances when implemented as computer programs.

There are several alternative ways of doing this: if this author were simply to encode other authors' methods, the results could be challenged on the grounds that the

resulting programs were not as efficient as they might have been due to personal bias (intentional or otherwise). It would be unreasonable to expect others to rewrite their programs to suit the computing installation here, and conversly the job of rewriting this program to suit the vagiaries of the many other computers used for this work is beyond the scope of this research.

It seems that the only viable alternative, therefore, is to compare this program on this machine with other programs run on other machines. This is not the only complication, however, because in making the comparisons we must bear in mind the differences due to using different languages, differences in compiler and assembler efficiencies, and differences in the programming skills of the various authors.

Now even if we accept that the comparison exercise is still worthwhile if only as a broad indicator, it must now be decided what set of "typical" examples to use for testing. No set of standardised problems has yet appeared in the literature, and nor is one ever likely, as different methods favour different types of problems. (Local extraction, for example, works best when the number of terms in the solution is small with respect to 2^n, and allows n, number of input variables, to be quite high.)

For want of any more acceptable method of problem selection, the procedure adopted was simply to write to several authors known to be active in this field and to ask them to supply some selected "typical" examples, complete with execution times. Notice that this is likely to bias the results in their favour, as they would only

select examples which their programs could solve in a reasonable time.

It would appear from Ewing et al (1961) that programs employing the extraction algorithm were available through IBM'S SHARE Library at that time. Unfortunately neither these nor any updates are generally available at present, and according to private correspondence with one of the co-authors of the above paper, his present version could not be used for comparison because "a new version of the compiler has, apparently, introduced a new bug in the program" (J.P. Roth, 1978).

An approach was made in person to Dr. A. Dunworth of the University of N.S.W. resulting in a useful exchange of ideas, but unfortunately he did not have access to any suitable programs (Dunworth, 1978).

Examples 7 to 21 in Appendix D were generated by Dr. P.S. Noe of Texas A & M University, and were used in comparing results with Prof. H.T. Nagle of Auburn University in a recent article (Rhyne et al, 1977). Their results are tabulated in table 6.3 together with the times taken by the program of Appendix A.

The computer used by them was an IBM 370/155, and the language was FORTRAN IV in both cases, which means that execution times for similar algorithms should be comparable. As the table shows, this program compares very favourably for nearly all the examples, with the worst results occurring where a long time was required to generate the PIs. It must be remembered that the routine used for this function is based directly on Hwa's, and does not form part of the algorithm as such.

Example	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Inputs	4	4	4	4	5	5	5	5	6	7	7	8	9	6	9	6
Outputs	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Minterms	2	9	14	9	12	4	10	15	16	92	4	45	4	39	100	38
Dontcares	11	0	0	0	0	12	7	0	9	0	81	156	503	0	100	0
PIs	3	7	12	6	12	4	10	6	18	37	4	12	5	37	4	35
Time to Generate PIs	0	0	0	0	0	.1	0	.1	0	.2	.4	4.3	33.0	.1	.5	0
Total time (secs)	0	0	.1	.1	.1	.1	.1	.1	0	.4	.5	4.4	33.0	2.7	.6	1.5
Time (Rhyne et al)	.31	.38	.44	.40	.48	.37	.33	.41	.37	.59	.44	.62	.92	1.31	.58	-
Time (Nagle)	.40	.38	.49	.41	.40	.45	.45	.42	.60	2.19	1.75	6.58	38.99	-	6.08	.96

Table 6.3 Comparisons with Rhyne et al and Nagle.

It seems from other correspondence however, that the times quoted for Rhyne's program may in fact have been derived for obtaining an approximate (non-minimum) solution, because in a letter to Prof. Nagle dated 3/10/74, it was stated in reference to problem 20 that "the minimum solution took 6 minutes". This is born out by the fact that the optimum solution (see appendix D) is cheaper than his by 4 terms. This being so, it is clear that the present program is to be preferred since it guarantees an optimum solution in the time stated.

Nagle's program apparently does not produce optimum solutions either, because the solution it found ostensibly for problem 20 was neither correct nor as cheap as the true optimum. Assuming that the fault was a mistake in data entry rather than in the program itself, his "solution" effectively defines a new problem (number 22 in appendix D), which may be expressed in 15 terms: 2 fewer than his.

Professor D.L. Dietmeyer of the University of Wisconsin was also approached, as he is presently working on minimisation algorithms and has recently published a revised edition of his book (Dietmeyer, 1978). Of the 40 10-variable multiple output examples which he selected, only 5 took longer to solve using this algorithm, and in each case, generating the PIs accounted for all the time taken. In fact only 3 examples registered more than .05 seconds to compute optimum solutions from the PI tables, and these were each .1 seconds, the smallest measurable increment of time.

In order to obtain further results, 4 examples selected by the author were used in comparative runs, and the results are shown in table 6.4.

Example	23	24	25	26
Inputs	7	7	8	5
Outputs	1	1	1	5
Minterms	63	68	33	65
Dontcares	14	13	188	27
PIs	56	74	159	76
Time to Generate PIs	.3	.3	2.3	1.2
Total time (secs)	24.9	33.0	17.0	39.1
Time (Dietmeyer)	18.592	38.017	97.451	54.915
Terms (optimum)	23	21	11	20
Literals (optimum)	111	102	47	75
Terms (Dietmeyer)	23	21	12	21
Literals (Dietmeyer)	113	102	52	79

Table 6.4 Comparisons with Dietmeyer

Again it would appear that the two programs are not doing the same job: Dietmeyer's solutions are not optimum, and so even though his program is faster on one of the four, this program is obviously to be preferred where the object is to find a truly minimum solution.

The processor used by Dietmeyer was "a Harris machine with a 24-bit word length and a machine cycle of 750 ns" (personal correspondence, 1978), compared with the Cyber's 60-bit words and memory cycle time of 1000 ns. This indicates that execution times should differ by no more than a ratio 3:4 due to machine differences alone.

In summary, therefore, computer programs capable of producing absolute minimum cost solutions for the examples considered do not seem to be readily available, and those programs which were available for comparative tests were in general significantly slower than the one reproduced in appendix A.

7. DISCUSSION

7.1 Unsuccessful Avenues

As intimated in previous sections, the strategies selected were the results of considerable research into a variety of approaches, some of which are outlined here in justifying those finally chosen.

(a) A comprehensive system for ordering the columns of the PI table was developed, based on such attributes as weight (number of crosses), adjacency (defined by Biswas, 1971), weights and costs of covering PIs. It was arranged to sort the columns using any number of these keys in any order, but it was found that the most significant quantity was weight, and to a lesser extent adjacency (which is highly correlated with weight anyway). Thus it was determined that ordering columns according to the number of crosses is best, taking into account both time and storage.

(b) Initially, no attempt was made to order the PIs, but this lead to a high degree of redundant branching (with the same combinations of terms being selected in different orders). "Static" ordering in which the PIs are sorted into a fixed hierachy afforded a considerable improvement: illegal successor PIs being determined by reference to such a list. Some time was spent determining which attributes could best be used for this task: PI weight, cost, and the weights and adjacencies of covered columns were all tried. Finally, the

"dynamic" ordering as previously described was adopted, the advantages being that the resulting tree is smaller and that sorting is performed automatically by the act of node selection.

(c) The node selection strategy proved to be the most critical with respect to efficiency, and also the most difficult to determine. The following quantities were considered: cost (c), number of covered columns (n), the position of the first zero in the cover (z), average cost (c/n) and the ratio (c/z). In addition various ideas were tried in attempting to relate the attributes of unticked nodes to those already in use, such as dividing by or subtracting normalising values. Some test problems fared better with other strategies, but generally, the one selected gave the best overall performance.

(d) Multi-level circuits may be considered as being composed of several 2-level circuits combined together. If the overall network is optimal, then each 2-level segment must also be optimal, and so this algorithm could be used in the following way. Choose a suitable set of subfunctions to be realised at level 2 and compute an optimal circuit. Choose further sets for the remaining levels and optimise each pair of levels until the desired outputs are obtained. The choices of subfunctions could probably be handled by a tree technique similar to that already described, but unfortunately the author was unable to develop the concept into a workable form.

7.2 Branch and Bound Techniques

This study provided a valuable insight into some of the important aspects involved in applying branch and bound techniques in a computer program. Practical difficulties arise because of the variable amount of information to be stored: nodes may be added or deleted almost anywhere within the tree, leading to problems in organising storage efficiently. The method chosen in the present case is to maintain pointers between nodes in both directions, so that when it becomes necessary to move a node so as to consolidate some available store, all other nodes affected by the change may readily be found.

Theoretical considerations can point the way towards the branching strategies required, but in the final analysis the only reliable way of determining which to use is to actually try them. Naturally the aims should be to restrict branching in the early levels of the tree and to inhibit redundant or repetitive branching, as described in chapter 3. In general, however, there seems to be no escape from the fact that the power of branch and bound techniques is heavily dependent on heuristic strategies.

7.3 Future Prospects

This algorithm has the advantage that it is applicable generally, and is not restricted to problems below a certain size. The program itself is only written to cater for ten variables, but its performance is limited by computer speed and memory: as bigger and faster computers become available, so it will extend the range of problems capable of true optimisation.

The class of ten variable problems almost certainly represents the upper limit for any approach requiring a PI table. This arises from a consideration of the memory sizes required, and the fact that computational time rises so dramatically as the size of the prime implicant table grows. Fortunately, however, problems of this size or larger are rarely encountered in practice, and may usually be subdivided into more manageable segments when they are.

Obviously, as technology improves and component costs decrease, designers are less likely to require this type of program, and are more likely to employ PLA's, ROMs and the like. However, the need for logic minimisation will still exist where for example manual methods are inadequate in reducing an expression to fit an available FPLA. Manufacturers who produce in quantity are still likely to use cheap SSI in preference to MSI or LSI for purely economic reasons, and so they have a continuing need for optimisation software. In research and development fields, too, the "bread-boarding" stage of a project is often realised in discrete gates, so that changes are readily made and do not require

replacement of expensive chips.

Finally, there is no escaping the fact that when speed of response is of paramount importance, two-level logic networks of the type considered here provide the best circuit architecture presently possible. There is no doubt, therefore, that optimisation algorithms such as this will have a demand well into the future.

7.4 Conclusions

Virtually every logic network in whatever application is partly or wholly combinational. All the non-memory elements in a sequential circuit for example constitute a combinational circuit, and yet common as the problem is, there is still no satisfactory general method for optimising such networks. No procedure for directly obtaining a solution has been developed which will operate on problems of more than eight variables, and indications are that some form of trial and error method is necessary. In order for such an approach to be efficient, the various choices must be conducted in a well-regimented manner, with all redundant operations avoided. The algorithm which has been developed here performs this function by using a reasonably sophisticated form of a branch and bound technique to construct an incompletely specified tree.

The foremost advantage this has over existing methods is its generality: there is (conceptually) no limit to the size of problem for which this algorithm will find a solution. Concepts such as essential prime implicants and dominated rows or columns receive no

special consideration, and computer storage is used efficiently because the program adapts itself to whatever is available.

Scant attention has been paid in the past to multiple output networks, probably because the number of variables puts their solution beyond the reach of existing minimisation techniques. With this program, however, problems in five input and five output variables for example may be handled with ease.

A further advantage over existing approaches is the flexibility of the cost function: a designer can minimise the number of packages required or even specifically preclude a particular size gate if none happen to be available.

The algorithm itself is an important contribution to the armoury of optimisation techniques. With the ready availability of computers, branch and bound methods are certain to gain wider acceptance, and as they do, programmers will find existing software such as this invaluable - particularly in such fields as integer programming.

Although limited to a certain extent, this program will handle most practical single or multiple output problems, and even where it fails, it can still provide a workable near-optimal solution. There is no doubt that it represents a significant improvement over existing methods and available software, providing engineers and designers with a practical and versatile means for optimising combinational logic.

```
      SUBROUTINE ADD(CUSLMAX)
      IMPLICIT INTEGER(A-Z)
      COMMON MINTERM,COM(2041)
      COMMON/STATS/NV,NO,LASTRAN,SEED, NOMIN,D,NPIMP,NEPI,
     * TERMINL,NONODES,THYME,TOTALG,TOTALL,TOTALT,
     * PFL,COSTMAX,GCOEF,LCOEF,PCOEF,TCOEF,PVALUES(10),SEL
      COMMON/POINERS/DEXAL,FINAL,IPOINT,NODE,NW,PIT,POINTER,S
     * ,SEXAL,START,THIS
      COMMON/NODE/COVL,COST,PRED,Z, NEWCOVL,NEWCOST,NEWN,NEWZ
     * ,PINO,SUCLIST(256),NEWUSL(256),NUSED,NSUCCS,NEWCOV(17)
      COMMON/DATA/H,NO101,ZERO,NO120
      DATA H,NO101,ZERO,PVALUES/10HUUUUUUUUUU,1LP,0,
     * 0,3,4,6,12,12,12,12,100,100/
      DATA NO120/37777770000000000000B/
      SNM(I,J,K)=SHIFT(COM(I),J).A..N.MASK(K)
      EXCOST(I)=SNM(I+1,33,27)

C FIRST SEE IF THERE IS SUFFICIENT SPACE
      CALL PAUSE("ADD")
      NEWLEN=0 $ IF(NEWCOVL.NE.0)NEWLEN=NEWCOVL+CUSLMAX
      CALL SHORTY(NEWLEN+2,0) $ NONODES=NONODES+1

C THERE IS ENOUGH ROOM (NOW)
C COPY DOWN TOP STACK ELEMENT TO MAKE ROOM FOR INDEX
      SEXAL=SEXAL-2 $ DEXAL=DEXAL-2
      COM(SEXAL)=COM(DEXAL) $ COM(SEXAL+1)=COM(DEXAL+1)
      NSUCCS=NSUCCS+1 $ SUCLIST(NSUCCS)=DEXAL
      IF(NEWCOVL.NE.0)GO TO 1

C NEW NODE IS TERMINAL - USES NONE OF MAIN STORE
      COM(DEXAL)=SHIFT(3,57).OR.SHIFT(NEWN,42).OR.
     * SHIFT(NEWZ,32).OR.SHIFT(PFL-THIS,17)
      COM(DEXAL+1)=SHIFT(NEWCOST,27).OR.SHIFT(PINO,6)
      FINAL=DEXAL $ CALL PAUSE("ADDFINAL") $ RETURN

C NEW NODE IS NOT TERMINAL
    1 COM(DEXAL)=
     * SHIFT(1,57).OR.SHIFT(NEWCOVL,52).OR.SHIFT(NEWN,42)
     * .OR.SHIFT(NEWZ,32).OR.SHIFT(PFL-THIS,17).OR.POINTER+1
      COM(DEXAL+1)=SHIFT(NEWCOST,27).OR.SHIFT(NEWLEN,19).OR.
     * SHIFT(PINO,6)

C COPY COVER
      DO 3 I=1,NEWCOVL
    3 COM(POINTER+I)=NEWCOV(I)
      POINTER=POINTER+NEWLEN
      RETURN $ END
```

```
      FUNCTION CHECK(DUM)
      IMPLICIT INTEGER(A-Z)
      COMMON MINTERM(1),COM(2041)
      COMMON/STATS/NV,NO,LASTRAN,SEED, NOMIN,D,NPIMP,NEFI,
     " TERMINL,NONODES,THYME,TOTALG,TOTALL,TOTALT,
     " PFL,COSTMAX,GCOEF,LCOEF,PCOEF,TCOEF,PVALUES(10),SEL
      COMMON/POINERS/DEXAL,FINAL,IPOINT,NODE,NW,PIT,POINTER,S
     " ,SEXAL,START,THIS
      COMMON/NODE/COVL,COST,PRED,Z, NEWCOVL,NEWCOST,NEWN,NEWZ
     " ,PINO,SUCLIST(256),NEWUSL(256),NUSED,NSUCCS,NEWCOV(17)
      COMMON/DATA/H,NO101,ZERO,NO120
      COMMON/OPUNIT/U

      P2=NOMIN+1+NOMIN+D $ COM(P2)=NO101 $ P3=P2+1

C EXAMINE EACH TERM IN THE SOLUTION
      DO 1 I=1,TOTALT
      TEMP=SUCLIST(I)
      B=SHIFT((TEMP-H).A.H,-1) $ A=C=.N.SHIFT(TEMP.A.H,-1)
      POINTER=P2 $ J=NOMIN

C REGENERATE ALL CANONICAL TERMS COVERED
    4 J=J+1 $ IF((B.A.C)-(MASK(21).A.COM(J)))2,3,4
    2 WRITE(6,6)DISPLAY(B.A.C) $ CHECK=-1 $ GO TO 5
    6 FORMAT(* --- ERROR IN SOLUTION: *A10
     " * EXTRA TERM IS CREATED*)

C TEMPORARILY RECORD THESE TERMS
    3 POINTER=POINTER+1 $ COM(POINTER)=B.A.C

C RECORD COLUMN WEIGHTS AND THIS TERM IF IT IS FIRST TO COVER
C THIS COLUMN
      IF((COM(J).A..N.MASK(30)).EQ.0)
     " COM(J)=COM(J).OR.SHIFT(I,15)
      COM(J)=COM(J)+1
      IF(C.EQ.-0)GO TO 22
      C=(C+1).OR.A
      GO TO 4
   22 TEMPA=SHIFT(1,40)

C STEP THROUGH (INPUT) LITERALS TO SEE IF ANY IS REDUNDANT
      DO 7 IPP=1,NV
      IF(IPP.LE.NO)GO TO 7
      IF((TEMP.A.TEMPA).NE.0)GO TO 7
      TEMPB=SHIFT(TEMPA,59) $ J=NOMIN

C ATTEMPT TO SHOW THAT THIS TERM IS NOT PRIME
C (I.E. HAS A REDUNDANT LITERAL)
      DO 8 IP=P3,POINTER
    9 J=J+1 $ IF(XOR(COM(IP),TEMPB)-(COM(J).A.MASK(21)))7,8,9
    8 CONTINUE
      WRITE(6,10)DISPLAY(TEMP) $ CHECK=-2 $ GO TO 5
   10 FORMAT(* --- ERROR IN SOLUTION: *A10* IS NOT PRIME*)
    7 TEMPA=SHIFT(TEMPA,2)
    1 CONTINUE
      J=NOMIN
```

```
C EXAMINE MINTERM WEIGHTS
      DO 15 I=1,NOMIN
   12 J=J+1
      IF(SHIFT(MINTERM(I),-39).NE.SHIFT(COM(J),-39))GO TO 12
      IF((COM(J).A..N.MASK(45))-1)13,14,15

C ENSURE NONE IS ZERO
   13 WRITE(6,16)DISPLAY(MINTERM(I)) $ CHECK=-3 $ GO TO 19
   16 FORMAT(* --- ERROR IN SOLUTION: *A10* IS NOT COVERED*)
   14 IPP=SHIFT(COM(J),45).A..N.MASK(45)

C FLAG EACH NECESSARY PRIME IMPLICANT
      SUCLIST(IPP)=SUCLIST(IPP).OR.MASK(1)
   15 CONTINUE

C SCAN EACH TERM OF THE SOLUTION TO ENSURE EACH WAS NECESSARY
      DO 11 I=1,TOTALT
      IF(SUCLIST(I).LT.0)GO TO 11
      IF(LASTRAN.EQ.4)GO TO 23

C THIS REDUNDANCY CHECK IS NOT APPLICABLE
C TO 2ND STAGE MULTIPLE O/P CIRCUITS
      WRITE(6,17)DISPLAY(SUCLIST(I)) $ CHECK=-4 $ GO TO 19
   17 FORMAT(* --- ERROR IN SOLUTION: *A10* IS REDUNDANT*)
   11 CONTINUE
      WRITE(U,18) $ CHECK=0 $ GO TO 19
   18 FORMAT(* THIS SOLUTION IS AT LEAST AN IRREDUNDANT *
     " *COVER OF PRIME IMPLICANTS*)
   23 WRITE(U,24) $ CHECK=0
   24 FORMAT(* THIS SOLUTION IS AT LEAST A COVER OF PRIME *
     " *IMPLICANTS*)

C TIDY UP FLAGS ETC.
   19 DO 20 I=1,TOTALT
   20 SUCLIST(I)=SUCLIST(I).A..N.MASK(1)
    5 DO 21 I=1,NPIMP
   21 COM(NOMIN+I)=COM(NOMIN+I).A.MASK(21)
      RETURN $ END
```

```
      SUBROUTINE CONSENS
      IMPLICIT INTEGER(A-Z)
      COMMON MINTERM(1),COM(2041)
      COMMON/STATS/NV,NO,LASTRAN,SEED, NOMIN,D,NPIMP,NEPI,
     " TERMINL,NONODES,THYME,TOTALG,TOTALL,TOTALT,
     " PFL,COSTMAX,GCOEF,LCOEF,PCOEF,TCOEF,PVALUES(10),SEL
      COMMON/DATA/H,NO101,ZERO,NO120
      COMMON/OPUNIT/U
      LOGICAL NEWNODE,PCOVERM
      CLR(I)=COM(I).A.NO120
      PCOVERM(P,M)=(SHIFT(XOR(CLR(P),CLR(M-1)),1).A..N.
     " XOR(CLR(P),CLR(M-1)).A.H).EQ.0

      P1=NOMIN+1 $ TOP=NOMIN+NPIMP
    2 TEMP=TOP $ TOP=NOMIN

C FIRST SET POINTERS IN EACH TERM
C (AND CLEAR DELETED TERMS WHEN RESTARTING)
      DO 1 I=P1,TEMP
      IF(COM(I).LT.0)GO TO 1
      TOP=TOP+1 $ COM(TOP)=COM(I).A.MASK(21).OR.TOP
    1 CONTINUE $ IF(TOP.EQ.PFL)STOP "CONSENS SHORT OF STORE"
    4 NOW=P1 $ SAVETOP=TOP

C CONSIDER EACH TERM OF THE LIST IN TURN
   10 IF(COM(NOW).LT.0)GO TO 3 $ PNOW=COM(NOW).A.MASK(21)
      IBOT=COM(NOW).A..N.MASK(45) $ ITOP=TOP

C COMPARE WITH ELEMENTS FROM POINTER TO (CURRENT) END-OF-LIST
      DO 5 I=IBOT,ITOP
      IF(COM(I).LT.0)GO TO 5 $ PI=COM(I).A.MASK(21)
      A=(H-PI).OR.(H-PNOW) $ B=SHIFT(A,1).A.A.A.H
      IF(COUNT(B).NE.1)GO TO 5

C CREATE THE CONSENSUS TERM AND COMPARE IT WITH THE WHOLE LIST
      MEMBER=H-XOR(A,SHIFT(B,-1).OR.B) $ NEWNODE=.F.
      DO 6 J=P1,TOP
      IF(COM(J).LT.0)GO TO 6 $ PJ=COM(J).A.MASK(21)

C IGNORE NEW TERM OF IT IS SUBSUMED
      IF(NEWNODE)GO TO 8
      IF(COUNT(MEMBER-H.A.H-PJ).EQ.0)GO TO 5
    8 IF(COUNT(PJ-H.A.H-MEMBER).NE.0)GO TO 6

C DELETE EXISTING ONE IF IT IS SUBSUMED
C (BY SETTING FLAG IN TOP BIT)
      COM(J)=MASK(1) $ NEWNODE=.T.
    6 CONTINUE
      IF(TOP.EQ.PFL)GO TO 2

C ADD THE NEW TERM TO THE END OF THE LIST
      TOP=TOP+1 $ COM(TOP)=MEMBER.OR.TOP
      IF(COM(NOW).LT.0)GO TO 3

C RESET POINTER FOR THIS TERM
    5 CONTINUE $ COM(NOW)=PNOW.OR.TOP
    3 NOW=NOW+1 $ IF(NOW.LT.TOP)GO TO 10

C IF ANY NEW TERMS APPEARED, GO BACK AND REPEAT THE PROCESS
      IF(SAVETOP.NE.TOP)GO TO 4 $ TEMP=NOMIN
```

```
C COMPRESS LIST BY REMOVING FLAGGED ELEMENTS
      DO 7 I=P1,TOP
      IF(COM(I).LT.0)GO TO 7
      TEMP=TEMP+1 $ COM(TEMP)=COM(I).A.MASK(21)
    7 CONTINUE
      NPIMP=0 $ COM(NOMIN)=NO101

C FINALLY, DELETE ANY PI.S WHICH COVER ONLY DONTCARES
      DO 20 I=P1,TEMP
      DO 26 J=1,NOMIN
      IF(PCOVERM(I,J))GO TO 27
   26 CONTINUE $ GO TO 20
   27 NPIMP=NPIMP+1 $ COM(NOMIN+NPIMP)=COM(I)
   20 CONTINUE
      WRITE(U,13)NPIMP
   13 FORMAT(* NUMBER OF PRIME IMPLICANTS:*I5)
      RETURN $ END
```

```
      SUBROUTINE CONTROL(STRING)
      IMPLICIT INTEGER(A-Z)
      COMMON MINTERM,COM(2041)
      COMMON/SELFLAG/SELFLAG
      COMMON/STATS/NV,NO,LASTRAN,SEED, NOMIN,D,NPIMP,NEFI,
     " TERMINL,NONODES,THYME,TOTALG,TOTALL,TOTALT,
     " PFL,COSTMAX,GCOEF,LCOEF,PCOEF,TCOEF,PVALUES(10),SEL
      COMMON/POINERS/DEXAL,FINAL,IPOINT,NODE,NW,PIT,POINTER,S
     " ,SEXAL,START,THIS
      COMMON/NODE/COVL,COST,PRED,Z, NEWCOVL,NEWCOST,NEWN,NEWZ
     " ,PINO,SUCLIST(256),NEWUSL(256),NUSED,NSUCCS,NEWCOV(17)
      COMMON/DATA/H,NO101,ZERO,NO120
      COMMON/OPUNIT/U
      COMMON/IPUNIT/UI,RSFLAG
      COMMON/PAUSE/PMODE,RETNO,RSTRING,RETPRIN,DISABLE,PCOUNT
      LOGICAL SELFLAG,DISABLE,LA,LB,RSFLAG,PMODE,RETPRIN,
     " TIMER,QMODE
      DIMENSION CHAR(3),INDEX(40),P(12)
      REAL TREMP,SEL,STATS(31),THYME,SECOND,ENDTIME
      EQUIVALENCE (STATS,NV)
      DATA NUMBER,INDEX/40,"CHECK",
     " "CONNECT","CONSENSUS","COSTS",
     " "DISABLE","DISCONNECT","DUMP","ENABLE","END","HEAVY",
     " "HWA","LASTRAN","LIST","MREAD","NET","NO","NODE","NV",
     " "OUT","PFL","PITABLE",
     " "POS","QMTONET","QUICK","RANDOM","RANDY","READATA",
     " "REPEAT","RESTART","RETURN","REWIND","RSWAP","RTOQM",
     " "SEED","SKIP","SOLUTION","SPECIAL","STATS","TEACH",
     " "TIME"/
      NM(I,J)=COM(I).A..N.MASK(J)
      SNM(I,J,K)=SHIFT(COM(I),J).A..N.MASK(K)
      EXTYPE(I)=SNM(I,3,57)
      EXCOVL(I)=SNM(I,8,55)
      EXN(I)=SNM(I,18,50)
      EXZ(I)=SNM(I,28,50)
      EXPRED(I)=PFL-SNM(I,43,45)
      EXLOC(I)=NM(I,43)
      EXCOST(I)=SNM(I+1,33,27)
      EXLEN(I)=SNM(I+1,41,52)
      EXPINO(I)=SNM(I+1,54,47)
      EXNSUC(I)=NM(I+1,54)
      EXSUCL(I)=EXLOC(I)+EXCOVL(I)
      EXUSL(I)=EXSUCL(I)+EXNSUC(I)
      EXLAST(I)=EXLOC(I)+EXLEN(I)-1
      EXBYTE(I,J)=SNM(J,I,45)
      XPCOST(I)=NM(NOMIN+I,27)
      TAGWT(I)=
     " COUNT(COM(NOMIN+I).A.SHIFT(MASK(NO+NO),NO+NO+39))

      SAVEPFL=PFL

C RESTART
   40 WRITE(6,1) $ QMODE=.F.
      DISABLE=SELFLAG=.T. $ RETPRIN=PMODE=TIMER=RSFLAG=.F.
      TERMINL=-0 $ LASTRAN=3 $ PFL=SAVEPFL $ RSTRING=1H
      REWIND 1 $ COSTMAX=.N.MASK(1) $ RSAVEU=6 $ UI=RECNO=0
      TCOEF=4608 $ LCOEF=1 $ GCOEF=PCOEF=RETNO=0 $ GO TO 34
    1 FORMAT(* CONTROL-*)
```

```
      ENTRY PAUSE $ IF(DISABLE)RETURN
      IF(STRING.EQ.8HSOLUTION)GO TO 120
      IF(TIMER)GO TO 73 $ IF(RETNO.LE.0)GO TO 76
      IF(RSTRING.NE.1H .A.RSTRING.NE.STRING)GO TO 96
      RETNO=RETNO-1 $ IF(RETNO.EQ.0)GO TO 76
 96   IF(RETPRIN)WRITE(6,61)STRING $ RETURN
 73   IF(SECOND(A).LT.ENDTIME)RETURN $ TIMER=.F.
 76   IF(PMODE)GO TO 105 $ IF(QMODE)82,122
120   RETNO=0 $ PMODE=TIMER=.F.
122   WRITE(6,61)STRING,3H ** $ GO TO 15
 61   FORMAT(A11,A3)
 34   WRITE(6,3)
  3   FORMAT(3H **)
 15   READ(5,4)CHAR $ U=SAVEU=6
  4   FORMAT(3A10)
      MSK=MASK(6)
      IF((CHAR(1).A.MSK).EQ.1L )GO TO 15
      CALL FREEFMT(CHAR,P,P(3),P(5),P(7),P(9),P(11))
      DO 87 I=1,11,2
 87   IF((P(I).A.MSK).EQ.1LS)U=SAVEU=4
      F=1 $ L=NUMBER
 84   TEMP=CHAR(1).A.MSK $ LAST=0
      DO 85 I=F,L
      IF(TEMP.NE.(INDEX(I).A.MSK))GO TO 85
      IF(LAST.EQ.0)FIRST=I $ LAST=I
 85   CONTINUE
      IF(LAST.EQ.0)GO TO 12
      IF(FIRST.EQ.LAST)GO TO
     " (45,49,89,21,50,53,17,51,60,91,90,80,58,107,6,
     " 119,13,28,29,27,11
     " ,72,31,9,32,65,106,14,40,77,68,35,36,59,67,117,38,23,86
     " ,20),FIRST
      MSK=SHIFT(MSK,-6) $ F=FIRST $ L=LAST $ GO TO 84
 12   WRITE(6,43) $ GO TO 34
 43   FORMAT(* CONTROL DOES NOT RECOGNISE THIS CODE*)

C CHECK
 45   U=4 $ LA=RSFLAG $ CALL REPEAT $ RSFLAG=LA
      IF(LA)CALL RSWAP $ TEMP=NPIMP $ CALL RTOQM $ U=SAVEU
      CALL CHECK(A) $ NPIMP=TEMP $ GO TO 34

C CONNECT
 49   DECODE(20,16,P(1))I $ CALL CONNEC(I)
 16   FORMAT(I20)
      GO TO 34

C CONSENSUS
 89   CALL CONSENS $ GO TO 34
```

```
C COSTS
   21   WRITE(U,47) $ READ(5,4)CHAR
   47   FORMAT(* ENTER T,L,G,P,COST,FLAG:*/)
        CALL FREEFMT(CHAR,P,P(3),P(5),P(7),P(9),P(11))
        DECODE(100,37,P(1))TCOEF,LCOEF,GCOEF,PCOEF,TREMP
   37   FORMAT(4I20,E20.0)
        COSTMAX=TREMP
        IF(COUNT(.N.TCOEF).EQ.0)TCOEF=4608
        IF(COUNT(.N.LCOEF).EQ.0)LCOEF=1
        IF(COSTMAX.EQ.0)COSTMAX=.N.MASK(1)
        IF(TCOEF+LCOEF.EQ.0)WRITE(6,24)
   24   FORMAT(* --- WARNING: ZERO PI COSTS MAY OCCUR*)
        IF(PCOEF.EQ.0)GO TO 34
        IF(P(11).EQ.1H .A.P(12).EQ.1H )GO TO 34
        WRITE(U,8) $ READ(5,66)PVALUES $ GO TO 34
    8   FORMAT(* ENTER PVALUES: (10I5)*/)
   66   FORMAT(10I5)

C DISABLE
   50   DISABLE=.T. $ GO TO 95

C DISCONNECT
   53   DECODE(20,16,P(1))I $ CALL DISCON(I)
        GO TO 34

C DUMP
   17   ADDTYPE=DUMTYPE=IPTYPE=0 $ LA=LB=.F.
        DO 19 I=1,9,2
        TEMPA=P(I).A.MASK(12) $ IF(TEMPA.NE.2L  )GO TO 22
        IF(LA)GO TO 71 $ LA=.T.
        DECODE(20,26,P(I))A $ GO TO 19
   71   IF(LB)GO TO 19 $ LB=.T.
        DECODE(20,26,P(I))B $ GO TO 19
   22   TEMP=TEMPA.A.MASK(6)
        IF(DUMTYPE.NE.0)GO TO 97 $ IF(TEMP.EQ.1LO)DUMTYPE=-1
        IF(TEMP.EQ.1LB)DUMTYPE=1 $ IF(TEMP.EQ.1LF)DUMTYPE=2
   97   IF(IPTYPE.NE.0)GO TO 98 $ IF(TEMP.EQ.1LA)IPTYPE=-1
        IF(TEMP.EQ.1LR)IPTYPE=1 $ IF(TEMP.EQ.1LM)IPTYPE=2
   98   IF(ADDTYPE.NE.0)GO TO 19
        IF(TEMPA.EQ.2LPA)ADDTYPE=-1
        IF(TEMPA.EQ.2LPR)ADDTYPE=1
        IF(TEMPA.EQ.2LPM)ADDTYPE=2
   19   CONTINUE $ DUMTYPE=MAXO(DUMTYPE,0)
        IF(COUNT(-A).EQ.0.OR..N.LA)A=0
        IF(COUNT(-B).EQ.0.OR..N.LB)B=A
        IF(ADDTYPE.EQ.0)ADDTYPE=IPTYPE
        ADDTYPE=MAXO(ADDTYPE,0)
        IF(IPTYPE-1)100,101,102
  102   A=PFL-A $ B=PFL-B
  101   A=LOCF(COM(A)) $ B=LOCF(COM(B))
  100   IF(A.LE.B)GO TO 103 $ I=A $ A=B $ B=I
  103   CALL DUMPC(P,A,B,DUMTYPE,ADDTYPE) $ GO TO 34

C ENABLE
   51   DISABLE=.F.
   95   RETPRIN=.F. $ RETNO=0 $ RSTRING=1H  $ GO TO 34

C END
   60   CALL DISCON(5) $ CALL DISCON(6) $ STOP "END PROGRAM"
```

```
C HEAVY
   91   IF((P(1).A.MASK(6)).NE.1LC)GO TO 92
        IF((P(3).A.MASK(6)).NE.1LW)GO TO 12
        CALL HEAVY(.001) $ GO TO 34
   92   IF((P(1).A.MASK(6)).NE.1LW)GO TO 12
        IF((P(3).A.MASK(6)).NE.1LC)GO TO 12
        CALL HEAVY(40.) $ GO TO 34

C HWA
   90   CALL HWA $ GO TO 34

C LASTRAN
   80   DECODE(20,16,P(1))LASTRAN $ GO TO 34

C LIST
   58   A=J=0 $ LA=.F. $ K=72
        DO 62 I=1,11,2
        TEMP=P(I).A.MASK(12)
        IF(TEMP.NE.2LRS)GO TO 93 $ LA=.T. $ GO TO 62
   93   TEMP=TEMP.A.MASK(6)
        IF(TEMP.EQ.1LB)A=-1
        IF(TEMP.EQ.1LA)J=7 $ IF(TEMP.EQ.1LM)J=J.OR.1
        IF(TEMP.EQ.1LD)J=J.OR.2 $ IF(TEMP.EQ.1LP)J=J.OR.4
        IF(TEMP.EQ.1LO)A=1
        IF(P(I+1).NE.1H )DECODE(20,16,P(I))K
   62   CONTINUE $ IF(J.EQ.0)GO TO 12 $ U=4
        CALL REPEAT $ IF(LA.OR.RSFLAG)CALL RSWAP $ U=SAVEU
        IF(U.EQ.4)U=7 $ CALL LISTER(J,A,K) $ GO TO 34

C MREAD
  107   LASTRAN=4 $ UI=5
        IF((P(1).A.MASK(6)).NE.1LF)GO TO 118 $ RECNO=RECNO+1
        UI=1 $ IF((P(3).A.MASK(6)).NE.1LR)GO TO 118
        REWIND 1 $ RECNO=1
  118   CALL MREAD $ GO TO 34

C NET
    6   ASSIGN 34 TO GOBACK
  111   RSAVEU=U $ PMODE=TIMER=.F. $ RETNO=0 $ CALL NET
        IF(LASTRAN.NE.4)GO TO GOBACK(34,88)
   83   TEMP=FINAL
  112   PINO=EXPINO(TEMP) $ IF(PINO.EQ.0)GO TO 108
        COM(NOMIN+PINO)=COM(NOMIN+PINO).OR.MASK(1)
        TEMP=EXPRED(TEMP) $ GO TO 112
  108   TEMP=NOMIN+1 $ TOP=NOMIN+NPIMP
        DO 109 I=1,NPIMP
        IF(COM(NOMIN+I).GE.0.A.(XPCOST(I).GE.2
       " .OR.TAGWT(I).EQ.0))GO TO 109
        COM(NOMIN+I)=COM(NOMIN+I).A..N.MASK(1)
        MSK=SHIFT(1,38)
  110   IF(TAGWT(I).LE.1)GO TO 113
  114   MSK=SHIFT(MSK,2)
        IF((MSK.A.COM(NOMIN+I)).EQ.0)GO TO 114
        COM(NOMIN+I)=COM(NOMIN+I).A..N.MSK
        IF(TEMP.EQ.NOMIN+I)GO TO 115
        COM(TEMP)=COM(NOMIN+I).A.MASK(21-NO-NO).OR.MSK
        TEMP=TEMP+1 $ GO TO 110
  115   TOP=TOP+1
        COM(TOP)=COM(NOMIN+I).A.MASK(21-NO-NO).OR.MSK
        GO TO 110
```

```
  113  COM(TEMP)=COM(NOMIN+I).A.MASK(21)
       TEMP=TEMP+1
  109  CONTINUE $ IF(TOP.EQ.NOMIN+NPIMP)GO TO 116
       NPIMP=NPIMP+NOMIN+1
       DO 121 I=NPIMP,TOP
       COM(TEMP)=COM(I)
  121  TEMP=TEMP+1
  116  NPIMP=TEMP-NOMIN-1
       SAVET=TCOEF $ TCOEF=1 $ SAVEL=LCOEF
       SAVEG=GCOEF $ SAVEP=PCOEF $ GCOEF=PCOEF=LCOEF=RETNO=0
       U=4
       CALL QMTONET $ TCOEF=SAVET $ LCOEF=SAVEL $ GCOEF=SAVEG
       PCOEF=SAVEP $ PMODE=TIMER=.F. $ CALL NET $ U=SAVEU
       GO TO GOBACK(34,88)

C NODE
   13  DECODE(20,26,P(1))TEMP $ IF(P(4).NE.1H )GO TO 48
       TEMPA=TEMP $ K=3 $ GO TO 52
   48  DECODE(20,26,P(3))TEMPA $ K=5
   52  IF((P(K).A.MASK(6)).EQ.1LR)GO TO 54 $ TEMP=PFL-TEMP
       TEMPA=PFL-TEMPA $ K=K-2
   54  IF(TEMP.LE.TEMPA)GO TO 55 $ I=TEMP $ TEMP=TEMPA
       TEMPA=I
   55  TRIPE=MASK(60)
       IF(P(K+2).EQ.1H )DECODE(20,16,P(K+2))TRIPE
       IF(COUNT(-TRIPE).EQ.0)TRIPE=-1
       DO 56 J=TEMP,TEMPA,2
       IF(TRIPE.GE.0.A.TRIPE.NE.EXTYPE(J))GO TO 56
       WRITE(6,7)PFL-J,EXTYPE(J),EXCOVL(J),EXN(J),EXZ(J)
      " ,PFL-EXPRED(J),EXCOST(J),EXPINO(J)
    7  FORMAT(* NODE *04"* TYPE"I2* COVL*I2* N *I3* Z *I3
      " * PRED *04"* COST"I10* PINO *03*B*)
       IF(EXTYPE(J).EQ.3)GO TO 56 $ A=EXLOC(J) $ B=EXSUCL(J)-1
       WRITE(U,18)10H     COVER,(COM(I),I=A,B)
   18  FORMAT(A11*; *3020/(13X3020))
       A=B+1 $ B=EXUSL(J)-1
       IF(A.LE.B)WRITE(U,18)10HSUCC. LIST,(COM(I),I=A,B)
       A=B+1 $ B=EXLAST(J) $ IF(A.GT.B)GO TO 56
       WRITE(U,18)10H  U. S. L.,(COM(I),I=A,B)
   56  CONTINUE $ GO TO 34

C NO
  119  DECODE(20,16,P(1))NO $ GO TO 34

C NV
   28  DECODE(20,16,P(1))NV $ GO TO 34

C OUT
   29  CALL OUT(0) $ GO TO 34

C PFL
   27  DECODE(20,26,P(1))PFL $ IF(PFL.LT.2)GO TO 39
       IF(PFL.GE.SAVEPFL)GO TO 39 $ WRITE(U,42)PFL $ GO TO 34
   39  PFL=SAVEPFL $ WRITE(U,44)PFL $ GO TO 34
   42  FORMAT(* PFL CHANGED TO *06*B*)
   44  FORMAT(* PFL RESET (TO *06*B)*)

C PITABLE
   11  DECODE(20,16,P(1))TEMP
       CALL PITABLE(TEMP) $ GO TO 34
```

```
C POS
   72  DECODE(20,16,P(1))PCOUNT
       IF(COUNT(-PCOUNT).NE.0)GO TO 99
       IF(STRING.NE.6HGOTONE)WRITE(6,2)
    2  FORMAT(* ---- WARNING: POS DIAGNOSTIC UNRELIABLE*)
       CALL POS $ GO TO 34
   99  DECODE(20,16,P(3))PRETTY $ PRETTY=MAXO(1,PRETTY)
       PMODE=.T. $ RETPRIN=.F. $ RSTRING=6HGOTONE
  104  RETNO=PRETTY $ RETURN
  105  CALL POS $ PCOUNT=PCOUNT-1 $ IF(PCOUNT.GT.0)GO TO 104
       PMODE=.F. $ RSTRING=1H $ GO TO 34

C QMTONET
   31  CALL QMTONET $ GO TO 34

C QUICK
    9  IF(FINAL.NE.0)GO TO 82 $ RSTRING=8HADDFINAL
       RETNO=1 $ QMODE=.T. $ RETPRIN=.F. $ RETURN
   82  DISABLE=.T. $ QMODE=.F. $ GO TO 83

C RANDOM
   32  CALL RANDOM $ LASTRAN=2 $ GO TO 34

C RANDY
   65  CALL RANDY $ LASTRAN=3 $ GO TO 34

C READATA
  106  LASTRAN=1 $ UI=5
       IF((P(1).A.MASK(6)).NE.1LF)GO TO 64 $ RECNO=RECNO+1
       UI=1 $ IF((P(3).A.MASK(6)).NE.1LR)GO TO 64
       REWIND 1 $ RECNO=1
   64  CALL READATA $ GO TO 34

C REPEAT
   14  CALL REPEAT $ GO TO 34

C RETURN
   77  DECODE(20,16,P(1))RETNO $ IF(P(3).EQ.4HTIME)GO TO 74
       RSTRING=P(3) $ RETPRIN=(P(5).A.MASK(6)).EQ.1LP
       IF(STRING.NE.5HFAULT)U=RSAVEU $ RETURN
   74  ENDTIME=SECOND(A)+RETNO $ RETNO=0 $ TIMER=.T.
       U=RSAVEU $ RETURN

C REWIND
   68  DECODE(20,16,P(1))I $ I=MAXO(1,I) $ IF(I.EQ.1)RECNO=0
       REWIND I $ GO TO 34

C RSWAP
   35  CALL RSWAP $ RSFLAG=.T. $ GO TO 34

C RTOQM
   36  CALL RTOQM $ GO TO 34

C SEED
   59  DECODE(20,26,P(1))SEED $ GO TO 34
   26  FORMAT(O20)
```

```
C SKIP
   67   DECODE(20,16,P(1))J $ IF(COUNT(.N.J).EQ.0)J=1
        IF(J)75,34,79
   75   J=RECNO+J $ RECNO=0 $ REWIND 1
   79   IF(J.LE.0)GO TO 34 $ RECNO=RECNO+J
        DO 78 I=1,J
   81   READ(1,4) $ IF(EOF(1))78,81
   78   CONTINUE $ GO TO 34

C SOLUTION
  117   CALL SOLUTIO $ GO TO 34

C SPECIAL
   38   CALL REPEAT $ RSFLAG=(P(1).A.MASK(12)).EQ.2LRS
        IF(RSFLAG)CALL RSWAP
        CALL HWA
        CALL QMTONET $ ASSIGN 88 TO GOBACK $ GO TO 111
   88   CALL OUT(0) $ SAVEU=6 $ GO TO 45

C STATS
   23   STATS(9)=TERMINL $ STATS(16)=COSTMAX
        WRITE(6,25)STATS $ TERMINL=STATS(9)
        COSTMAX=STATS(16) $ GO TO 34
   25   FORMAT(* NV*I2* NO*I2* LA*I1* SEED*O16* NOMIN*I4* D*I3
       " * NPIMP*I4* NEPI*
       " I3/* TERM*1PG8.2* NN*OPI5* T*F6.1* TG*I3* TL*I4* TT*I3
       " * PFL*O6*B OLD*1PG8.2/* GC*OPI3* LC*I3* PC*I3* TC*I4
       " * PVALS*10I3*SEL*F5.3)

C TEACH
   86   WRITE(6,94)INDEX $ GO TO 34
   94   FORMAT(24X*---- CONTROL COMMANDS ----*/
       " 21X*(FOR DETAILS REFER TO HANDBOOK)*/(6A12))

C TIME
   20   CALL SECOND(TREMP)
        IF((P(1).A.MASK(6)).NE.1LA)TREMP=TREMP-THYME
        WRITE(6,57)TREMP $ GO TO 34
   57   FORMAT(F7.1* SECONDS*)
        END
```

```
      SUBROUTINE DELETE(OLDN)
      IMPLICIT INTEGER (A-Z)
      COMMON MINTERM,COM(2041)
      COMMON/STATS/NV,NO,LASTRAN,SEED, NOMIN,D,NPIMP,NEPI,
     * TERMINL,NONODES,THYME,TOTALG,TOTALL,TOTALT,
     * PFL,COSTMAX,GCOEF,LCOEF,PCOEF,TCOEF,PVALUES(10),SEL
      COMMON/POINERS/DEXAL,FINAL,IPOINT,NODE,NW,PIT,POINTER,S
     * ,SEXAL,START,THIS
      COMMON/NODE/COVL,COST,PRED,Z, NEWCOVL,NEWCOST,NEWN,NEWZ
     * ,PINO,SUCLIST(256),NEWUSL(256),NUSED,NSUCCS,NEWCOV(17)
      COMMON/DATA/H,NO101,ZERO,NO120
      LOGICAL DOMINO
      DATA NO32515/777777774000037777 7B/
      NM(I,J)=COM(I).A..N.MASK(J)
      SNM(I,J,K)=SHIFT(COM(I),J).A..N.MASK(K)
      EXTYPE(I)=SNM(I,3,57)
      EXCOVL(I)=SNM(I,8,55)
      EXPRED(I)=PFL-SNM(I,43,45)
      EXLOC(I)=NM(I,43)
      EXCOST(I)=SNM(I+1,33,27)
      EXLEN(I)=SNM(I+1,41,52)
      EXPINO(I)=SNM(I+1,54,47)
      EXNSUC(I)=NM(I+1,54)
      EXSUCL(I)=EXLOC(I)+EXCOVL(I)
      EXUSL(I)=EXSUCL(I)+EXNSUC(I)
      EXLAST(I)=EXLOC(I)+EXLEN(I)-1
      EXBYTE(I,J)=SNM(J,I,45)

      OLD=OLDN
      CALL PAUSE("DELETE")
      OLDTYPE=EXTYPE(OLD)
      IF(OLDTYPE.NE.0)GO TO 1

C STACK DELETE: DO IT AND GO
      S=S-EXLEN(OLD)-2
      COM(OLD)=COM(SEXAL) $ COM(OLD+1)=COM(SEXAL+1)
      SEXAL=SEXAL+2 $ RETURN
    1 IPOINT=EXPRED(OLD)
      DOMINO=OLD.NE.THIS $ IF(DOMINO)GO TO 25
      THIS=0 $ GO TO 9
   25 IF(IPOINT.NE.THIS)GO TO 9

C DELETION OF NEWLY CREATED NODE
      I=0
   11 I=I+1
      IF(SUCLIST(I).NE.OLD)GO TO 11
      SUCLIST(I)=SUCLIST(NSUCCS) $ NSUCCS=NSUCCS-1
      S=S-EXLEN(OLD)
      NUSED=NUSED+1 $ NEWUSL(NUSED)=EXPINO(OLD) $ GO TO 7

C DELETION OF ESTABLISHED NODE
    9 FIRST=EXSUCL(IPOINT) $ LAST=EXUSL(IPOINT)-1
      MOLD=PFL-OLD

C SEARCH IPOINT'S SUCCESSOR LIST FOR OLD
   40 DO 2 I=15,60,15
      TEMP=EXBYTE(I,FIRST)
      IF(TEMP.EQ.MOLD)GO TO 12
    2 CONTINUE
      FIRST=FIRST+1 $ GO TO 40
```

```
C OVERWRITE IT WITH 1ST BYTE OF LAST WORD OF LIST
   12   TEMP=EXBYTE(15,LAST)
        COM(FIRST)=
       * SHIFT(SHIFT(COM(FIRST),I).A.MASK(45).OR.TEMP,60-I)
        COM(LAST)=SHIFT(COM(LAST),15).A.MASK(45)
        IF(COM(LAST).NE.0)GO TO 3

C SUCCESSOR LIST IS ONE WORD SHORTER,
C COPY DOWN USL AND RESET INDEX
        S=S-1 $ COM(IPOINT+1)=COM(IPOINT+1)-(SHIFT(1,19).OR.1)
        L=EXLAST(IPOINT) $ IF(LAST-L)27,4,3
   27   COM(LAST)=COM(L)
    4   COM(L)=COM(L+1)
    3   IF(OLDTYPE-2)5,6,14
    6   S=S-2 $ CALL SHORTY(2,0) $ GO TO 28
   13   S=S-EXLEN(OLD) $ GO TO 7

C TYPE 3 DELETE: DO IT NOW
   14   FINAL=0
    7   CALL MOVE(DEXAL,OLD)
        COM(DEXAL)=COM(SEXAL) $ COM(DEXAL+1)=COM(SEXAL+1)
        SEXAL=SEXAL+2 $ S=S-2 $ DEXAL=DEXAL+2

C DELETION COMPLETE: EXIT IF IPOINT HAS OTHER SUCCESSORS
    8   IF(EXNSUC(IPOINT).NE.0)GO TO 19
        IF(IPOINT.EQ.THIS)RETURN
        CALL PAUSE("DELBACKUP")
        OLD=IPOINT $ OLDTYPE=EXTYPE(OLD) $ IPOINT=EXPRED(OLD)
        IF(IPOINT.EQ.PFL)STOP "CANT FIND CHEAPER SOLUTION"
        DOMINO=EXCOST(OLD).GE.TERMINL $ GO TO 9

C RESET ORIGINAL PARAMETER IN CASE IT WOULD POINT
C TO A NOW NON-EXISTENT NODE
   19   IF(OLDN.NE.0)OLDN=MAX0(OLDN,SEXAL-2) $ RETURN

C IS THIS A DOMINATED TYPE 1 -
    5   FIRST=EXSUCL(IPOINT)
        LENM1=EXUSL(IPOINT)-FIRST-1 $ IF(LENM1.LT.0)GO TO 20

C YES, ADD ITS PI.NO. TO THE U.S.L. OF EACH TYPE 1 SIBLING
        DO 90 J=ZERO,LENM1
        DO 90 I=15,60,15
        TEMP=EXBYTE(I,FIRST+J) $ IF(TEMP.EQ.0)GO TO 20
        TEMP=PFL-TEMP
        IF(EXTYPE(TEMP).NE.1)GO TO 90
        F=EXUSL(TEMP) $ L=EXLAST(TEMP) $ IF(F.GT.L)GO TO 91
        DO 92 K=30,60,15
        IF(EXBYTE(K,L).NE.0)GO TO 92
        COM(L)=COM(L).OR.SHIFT(EXPINO(OLD),60-K) $ GO TO 90
   92   CONTINUE
   91   CALL SHORTY(1,TEMP) $ POINTER=POINTER+1
        COM(POINTER)=SHIFT(EXPINO(OLD),45)
        COM(TEMP+1)=COM(TEMP+1)+SHIFT(1,19)
        FIRST=EXSUCL(IPOINT)
   90   CONTINUE
```

```
C TYPE 1 DELETE: CREATE STACK MEMBER IF REQUIRED
   20   IF(DOMINO)GO TO 13
        S=S-EXLEN(OLD)+EXCOVL(OLD)
        TEMPA=COM(OLD) $ TEMPB=COM(OLD+1)
        CALL MOVE(DEXAL,OLD)
        COM(DEXAL)=TEMPA.A.NO32515
        COM(DEXAL+1)=
      " TEMPB.A.MASK(33).OR.SHIFT(EXCOVL(DEXAL),19)
        DEXAL=DEXAL+2 $ GO TO 8

C TYPE 2 DELETE: REMOVE WHOLE SUBTREE
   23   TEMP=POINTER-SAVPOIN $ S=S-TEMP $ CALL SHORTY(TEMP,0)
   28   SAVPOIN=POINTER $ POINTER=POINTER+1 $ COM(POINTER)=OLD
        L=POINTER
   10   CONTINUE
        K=COM(L)
        FIRST=EXSUCL(K) $ LAST=EXUSL(K)-1
        IF(FIRST.GT.LAST)GO TO 16

C ADD ALL SUCCESSORS TO A STACK
        DO 17 J=FIRST,LAST
        DO 17 I=15,60,15
        TEMP=EXBYTE(I,J) $ IF(TEMP.EQ.0)GO TO 16
        POINTER=POINTER+1 $ IF(POINTER.EQ.SEXAL)GO TO 23
        COM(POINTER)=PFL-TEMP
   17   CONTINUE
   16   CONTINUE
        L=L+1 $ IF(L.LE.POINTER)GO TO 10
        II=SAVPOIN+2 $ IF(POINTER.LT.II)GO TO 15

C SORT THE STACK
        DO 24 I=II,POINTER $ TEMP=COM(I-1)
        DO 24 J=I,POINTER
        IF(TEMP.LT.COM(J))GO TO 24
        COM(I-1)=COM(J) $ COM(J)=TEMP $ TEMP=COM(I-1)
   24   CONTINUE
   15   II=POINTER-SAVPOIN $ L=SAVPOIN+1
        DO 21 J=1,II
        IF(DEXAL.NE.COM(L))GO TO 22

C DELETE DEXAL
        L=L+1 $ K=DEXAL $ GO TO 26

C COPY DEXAL TO TOPMOST NODE TO BE DELETED
   22   K=COM(POINTER) $ POINTER=POINTER-1
   26   S=S-EXLEN(K)-2
        IF(K.NE.FINAL)GO TO 18 $ FINAL=0 $ TERMINL=TERMINL+1
   18   CALL MOVE(DEXAL,K)
        COM(DEXAL)=COM(SEXAL) $ COM(DEXAL+1)=COM(SEXAL+1)
        SEXAL=SEXAL+2
   21   DEXAL=DEXAL+2
        POINTER=SAVPOIN $ GO TO 8
        END
```

```
      SUBROUTINE FAULT(NUMBER)
      COMMON/PAUSE/PMODE,RETNO,RSTRING,RETPRIN,DISABLE,PCOUNT
      LOGICAL DISABLE,PMODE
      INTEGER RETNO

      DISABLE=PMODE=.F. $ RETNO=PCOUNT=0 $ WRITE(6,1)NUMBER
    1 FORMAT(* --- FAULT NUMBER*I3* DETECTED IN TREE*)
      CALL PAUSE("FAULT") $ RETURN $ END
```

```
$HEADING
      SUBROUTINE HEADING(RADIX,FIXEDR)
      IMPLICIT INTEGER (A-Z)
      COMMON MINTERM(1),COM(2040),DONTCAR(1)
      COMMON/STATS/NV,NO,LASTRAN,SEED, NOMIN,D,NPIMP,NEPI,
     " TERMINL,NONODES,THYME,TOTALG,TOTALL,TOTALT,
     " PFL,COSTMAX,GCOEF,LCOEF,PCOEF,TCOEF,FVALUES(10),SEL
      COMMON/OPUNIT/U
      COMMON/POINERS/DEXAL,FINAL,IPOINT,NODE,NW,PIT,POINTER,S
     " ,SEXAL,START,THIS
      COMMON/IPUNIT/UI,RSFLAG
      COMMON/INPDATA/MINTS(18),DONS(18),NVAR,NOP
      COMMON/NODE/COVL,COST,PRED,Z, NEWCOVL,NEWCOST,NEWN,NEWZ
     " ,PINO,SUCLIST(256),NEWUSL(256),NUSED,NSUCCS,NEWCOV(17)
      COMMON/DATA/H,NO101,ZERO,NO120
      DIMENSION LINE(8),MONTH(12)
      LOGICAL MINSTIL,FIXEDR,BETWEEN,RSFLAG
      DATA DISPO/1RO/
      DATA MONTH/
     " 9H JAN.  19,9H FEB.  19,9H MARCH 19,9H APRIL 19,
     " 9H MAY   19,9H JUNE  19,9H JULY  19,9H AUG.  19,
     " 9H SEPT. 19,9H OCT.  19,9H NOV.  19,9H DEC.  19/

C CHECK THAT NV IS IN RANGE, AND PRINT A HEADING
      CALL DATE(TEMPA) $ CALL TIME(TEMPB)
      IF(NV.GE.1.A.NV.LE.10)GO TO 1
      WRITE(6,90) $ RETURN
   90 FORMAT(* ---- ERROR: NV OUT OF BOUNDS*)
    1 DECODE(10,93,TEMPA)A,B,C
   93 FORMAT(1XA2,1XI2,1XI2)
      DECODE(10,97,TEMPB)CCL,COL
   97 FORMAT(1XI2,1XA2)
      TEMP=2L A $ IF(CCL.GE.12)TEMP=2L P
      IF(CCL.GE.13)CCL=CCL-12
      WRITE(U,7)CCL,COL,TEMP,C,MONTH(B),A
    7 FORMAT(1HOI12*:*A2,A2*.M.*I3,A9,A2)
      WRITE(U,95)NV
   95 FORMAT(* NUMBER OF VARIABLES:*I3)
      WRITE(U,98)NOMIN,D
   98 FORMAT(* NUMBER OF MINTERMS:*I5
     " *,  NUMBER OF DONTCARES:*I5)
      RSFLAG=.F.
      RETURN

      ENTRY READIN

C SERVICE ROUTINE TO HANDLE INPUT DATA (1 CHARACTER AT A TIME)
      TOPJ=PFL-2040
      LOW=HIGH=0 $ MINSTIL=BETWEEN=.T.
   12 READ(UI,99)LINE $ IF(EOF(UI).NE.0)RETURN
   99 FORMAT(8A10)

C EXAMINE THE 80-COLUMN LINE
      DO 3 WORD=1,8
      DO 3 ICH=1,10
      CHAR=SHIFT(LINE(WORD),6*ICH).A..N.MASK(54)
      IF(RADIX)14,13,15
```

```
C 'BINARY' IS STORED BIT-BY-BIT IN 2 TERMS, HIGHEST AND LOWEST
   13  IF(CHAR.NE.1R0)GO TO 16 $ LOW=SHIFT(LOW,1)
       HIGH=SHIFT(HIGH,1) $ GO TO 24
   16  IF(CHAR.NE.1R1)GO TO 17 $ LOW=SHIFT(LOW,1)+1
       HIGH=SHIFT(HIGH,1)+1 $ GO TO 24
   17  IF(CHAR.NE.1R-)GO TO 6 $ LOW=SHIFT(LOW,1)
       HIGH=SHIFT(HIGH,1)+1 $ GO TO 24
   14  IF(CHAR.LT.1R0.OR.CHAR.GT.1R7)GO TO 6
       LOW=SHIFT(LOW,3).OR.(CHAR-DISP0) $ GO TO 24
   15  IF(CHAR.LT.1R0.OR.CHAR.GT.1R9)GO TO 6
       LOW=LOW*10+CHAR-DISP0
   24  IF(ICH.EQ.10.A.WORD.EQ.8)GO TO 21 $ BETWEEN=.F.
       GO TO 3
    6  IF(BETWEEN)GO TO 10

C END OF DIGIT STRING: STORE TERMS IN MINTERM/DONTCARE LISTS
   21  IF(RADIX.NE.0)GO TO 18 $ IF(FIXEDR)GO TO 2
       A=C=XOR(HIGH,-LOW)

C SINGLE O/P BINARY FORMAT: REGENERATE ALL POSSIBLE TERMS
   20  IF(MINSTIL)GO TO 22 $ IF(D.EQ.TOPJ)CALL SHORTD(0)
       D=D+1 $ DONTCAR(D)=HIGH.A.A $ GO TO 23
   22  NOMIN=NOMIN+1 $ MINTERM(NOMIN)=HIGH.A.A
       IF(NOMIN.EQ.2040)CALL SHORTM(0)
   23  IF(A.EQ.0)GO TO 4 $ A=(A.A..N.MASK(1))+1.OR.C
       GO TO 20

C MULTIPLE OUTPUT BINARY:
C TERMS FOR EACH 1 IN O/P PART MUST BE REGENERATED
    2  TEMPL=LOW.A..N.MASK(60-NO)
       TEMPH=HIGH.A..N.MASK(60-NO)
       LOW=LOW.A.MASK(60-NO) $ HIGH=HIGH.A.MASK(60-NO)

C SCAN O/P LITERALS TO DETERMINE WHETHER MINTERM OR DONTCARE
       DO 92 I=1,NO
       TEMP=SHIFT(MASK(1),I) $ IF((TEMP.A.TEMPH).EQ.0)GO TO 92
       MINSTIL=(TEMP.A.TEMPL).NE.0
       A=C=XOR(HIGH,-LOW)

C GENERATE ALL POSSIBLE TERMS AS ABOVE
   19  IF(MINSTIL)GO TO 94 $ IF(D.EQ.TOPJ)CALL SHORTD(0)
       D=D+1 $ DONTCAR(D)=(HIGH.OR.TEMP).A.A $ GO TO 96
   94  NOMIN=NOMIN+1 $ MINTERM(NOMIN)=(HIGH.OR.TEMP).A.A
       IF(NOMIN.EQ.2040)CALL SHORTM(0)
   96  IF(A.EQ.0)GO TO 92 $ A=(A.A..N.MASK(1))+1.OR.C
       GO TO 19
   92  CONTINUE $ GO TO 4

C OCTAL OR DECIMAL FORMAT: JUST STORE IN APPROPRIATE LIST
   18  IF(MINSTIL)GO TO 91 $ IF(D.EQ.TOPJ)CALL SHORTD(0)
       D=D+1 $ DONTCAR(D)=LOW $ GO TO 4
   91  NOMIN=NOMIN+1 $ MINTERM(NOMIN)=LOW
       IF(NOMIN.EQ.2040)CALL SHORTM(0)
    4  LOW=HIGH=0 $ BETWEEN=.T.

C PERIOD DENOTES END OF DATA STRING
   10  IF(CHAR.EQ.1R.)RETURN
       IF(FIXEDR)GO TO 3
```

```
C INTERPRET SPECIAL CHARACTERS AS REQUIRED
      IF(CHAR.NE.1RB)GO TO 5 $ RADIX=0 $ GO TO 3
    5 IF(CHAR.NE.1RD)GO TO 8 $ RADIX=1 $ GO TO 3
    8 IF(CHAR.NE.1RO)GO TO 9 $ RADIX=-1 $ GO TO 3
    9 IF(CHAR.EQ.1R/)MINSTIL=.F.
    3 CONTINUE $ GO TO 12

      ENTRY SORTHEM
      CALL SHORTM(0)
      CALL SHORTD(0)
      IF(LASTRAN.NE.4)GO TO 50

C FOR MULTIPLE O/P, AUGMENT THE DONTCARE LIST TO ENSURE
C NO 1 IN O/P PART OF PI.S
      MPOIN=OMP=ODP=1 $ INC=SHIFT(1,NO)
      TOPRI=XOR(MASK(60-NV),MASK(60-NO))
      DO 89 PRI=ZERO,TOPRI,INC

C COMBINE TERMS WITH COMMON I/P PARTS
      COMBINE=0 $ DPOIN=ODP
   27 IF((MINTERM(MPOIN).A.TOPRI).NE.PRI.OR.MPOIN.GT.NOMIN)
     " GO TO 26 $ COMBINE=COMBINE.OR.MINTERM(MPOIN)
      MPOIN=MPOIN+1 $ GO TO 27
   26 IF((DONTCAR(DPOIN).A.TOPRI).NE.PRI.OR.DPOIN.GT.D)
     " GO TO 28
      COMBINE=COMBINE.OR.DONTCAR(DPOIN) $ DPOIN=DPOIN+1
      GO TO 26
   28 IF(COMBINE.EQ.0)GO TO 89
      TEMP=MPOIN+DPOIN-OMP-ODP $ TEMP=SHIFT(1,TEMP)-TEMP

C CREATE SPACE FOR THE EXTRA TERMS
      DO 29 J=DPOIN,D
   29 DONTCAR(DPOIN+D-J+TEMP)=DONTCAR(DPOIN+D-J)
      D=D+TEMP
      A=C=XOR(COMBINE,-PRI)

C GENERATE ALL LEGAL COMBINATIONS OF THE O/P PART
   84 TEMP=COMBINE.A.A $ IF(TEMP.NE.MINTERM(OMP))GO TO 83
      OMP=OMP+1 $ GO TO 32
   83 DONTCAR(ODP)=TEMP $ ODP=ODP+1
   32 IF(A.EQ.0)GO TO 89 $ A=(A.A..N.MASK(1))+1.OR.C
      GO TO 84
   89 CONTINUE

C RECORD THE LISTS IN COMPRESSED FORMAT TO OBVIATE RE-READING
   50 NVAR=NV $ NOP=NO
      DO 47 I=1,18
   47 MINTS(I)=DONS(I)=0
      DO 48 I=1,NOMIN
      TEMP=MINTERM(I) $ J=TEMP/60+1
   48 MINTS(J)=MINTS(J).OR.SHIFT(MASK(1),60*J-TEMP)
      IF(D.EQ.0)RETURN
      DO 49 I=1,D
      TEMP=DONTCAR(I) $ J=TEMP/60+1
   49 DONS(J)=DONS(J).OR.SHIFT(MASK(1),60*J-TEMP) $ RETURN
      END
```

```
      SUBROUTINE HEAVY(WCOEF)
      IMPLICIT INTEGER(A-Z)
      COMMON MINTERM(1),COM(2041)
      COMMON/TEMPARY/TEMPRY(40)
      COMMON/STATS/NV,NO,LASTRAN,SEED, NOMIN,D,NPIMP,NEPI,
     " TERMINL,NONODES,THYME,TOTALG,TOTALL,TOTALT,
     " PFL,COSTMAX,GCOEF,LCOEF,PCOEF,TCOEF,PVALUES(10),SEL
      COMMON/POINERS/DEXAL,FINAL,IPOINT,NODE,NW,PIT,POINTER,S
     " ,SEXAL,START,THIS
      COMMON/NODE/COVL,COST,PRED,Z, NEWCOVL,NEWCOST,NEWN,NEWZ
     " ,PINO,SUCLIST(256),NEWUSL(256),NUSED,NSUCCS,NEWCOV(17)
      COMMON/DATA/H,NO101,ZERO,NO120
      REAL WCOEF,MAXIM,SELCOST
      NM(I,J)=COM(I).A..N.MASK(J)
      XPCOST(I)=NM(NOMIN+I,27)

      TERMINL=TOTALT=TOTALL=TOTALG=0

C CLEAR TEMPRY FOR STORING THE COVER AS IT IS GENERATED
      DO 1 I=1,NW
    1 TEMPRY(I)=0
    4 MAXIM=0. $ COVPOIN=PIT

C STEP THROUGH THE PI.S TO FIND HEAVIEST
      DO 2 I=1,NPIMP $ WEIGHT=0
      DO 3 J=1,NW
    3 WEIGHT=WEIGHT+COUNT(COM(COVPOIN+J).A..N.TEMPRY(J))
      COVPOIN=COVPOIN+NW
      IF(WEIGHT.EQ.0)GO TO 2

C COMPUTE EACH COST FUNCTION
C DEFAULT COST COEFFICIENTS FOR QMTONET ASSUMED
      SELCOST=WCOEF*WEIGHT+(40-XPCOST(I))

C CHOOSE HIGHEST
      IF(SELCOST.LE.MAXIM)GO TO 2
      MAXIM=SELCOST $ GOTCHA=I
    2 CONTINUE
      IF(MAXIM.EQ.0.)GO TO 6

C SELECT THIS ROW
      DO 5 I=1,NW
    5 TEMPRY(I)=TEMPRY(I).OR.COM(PIT+(GOTCHA-1)*NW+I)
      TEMP=COM(NOMIN+GOTCHA).A.MASK(21)
      NOLITS=COUNT(H.A..N.TEMP)+NV-30
      TOTALT=TOTALT+1 $ TOTALL=TOTALL+NOLITS
      IF(NOLITS.GT.1)TOTALG=TOTALG+1
      SUCLIST(TOTALT)=TEMP
      TERMINL=TERMINL+XPCOST(GOTCHA)
      GO TO 4
    6 SUCLIST(TOTALT+1)=TERMINL
      RETURN $ END
```

```
      SUBROUTINE HWA
      IMPLICIT INTEGER(A-Z)
      COMMON MINTERM(1),COM(2040),DONTCAR(1)
      COMMON/STATS/NV,NO,LASTRAN,SEED, NOMIN,D,NPIMP,NEPI,
     " TERMINL,NONODES,THYME,TOTALG,TOTALL,TOTALT,
     " PFL,COSTMAX,GCOEF,LCOEF,PCOEF,TCOEF,PVALUES(10),SEL
      COMMON/POINERS/DEXAL,FINAL,IPOINT,NODE,NW,PIT,POINTER,S
     " ,SEXAL,START,THIS
      COMMON/NODE/COVL,COST,PRED,Z, NEWCOVL,NEWCOST,NEWN,NEWZ
     " ,PINO,SUCLIST(256),NEWUSL(256),NUSED,NSUCCS,NEWCOV(17)
      COMMON/DATA/H,NO101,ZERO,NO120
      COMMON/OPUNIT/U
      LOGICAL PCOVERM,FLAG,NOTPOSS,SUBSUMD,SUBSUMS
      CLR(I)=COM(I).A.NO120
      JOINT(I,K)=((CLR(I).A.CLR(K).OR.SHIFT(CLR(K),1)
     " .A.H-CLR(I)).A.H.OR.SHIFT(H.A..N.H-CLR(I).A..N.
     " (CLR(I).A.CLR(K)),-1)).A.NO120
      NOTPOSS(I,K)=COUNT(H-CLR(I).A.H.A.CLR(K).OR.
     " SHIFT(H-CLR(I).A.H.A.CLR(K)-H,-1)).NE.1
      SUBSUMD(I)=(MEMBER-H.A.H-CLR(I)).EQ.0
      SUBSUMS(I)=(CLR(I)-H.A.H-MEMBER).EQ.0
      PCOVERM(P,M)=(SHIFT(XOR(CLR(P),CLR(M-1)),1).A..N.
     " XOR(CLR(P),CLR(M-1)).A.H).EQ.0

C EXPAND MINTERMS AND RECORD WEIGHT
      DO 1 I=1,NOMIN
      TEMP=QMXPAND(MINTERM(I))
    1 MINTERM(I)=TEMP.OR.COUNT(TEMP)
      IF(D.EQ.0)GO TO 5

C DITTO DONTCARES
      DO 2 I=1,D
      TEMP=QMXPAND(DONTCAR(I))
    2 DONTCAR(I)=TEMP.OR.COUNT(TEMP)
    5 LAST=J=NOMIN

C COMBINE THE TWO LISTS INTO ASCENDING WEIGHT, NOMIN THRU LAST
      DO 3 INDEX=ZERO,NV
      DO 4 I=1,NOMIN
      IF((MINTERM(I).A.15).NE.INDEX)GO TO 4

C CLEAR WEIGHT FROM MINTERM ENTRY
      COM(LAST)=MINTERM(I) $ MINTERM(I)=MINTERM(I).A.MASK(21)
      LAST=LAST+1
    4 CONTINUE
      IF(D.EQ.0)GO TO 3
      DO 13 I=1,D
      IF((DONTCAR(I).A.15).NE.INDEX)GO TO 13
      COM(LAST)=DONTCAR(I) $ LAST=LAST+1
   13 CONTINUE
    3 CONTINUE

C TEMP=LAST ENTRY, =DUMMY WITH WEIGHT NV+1
      COM(LAST)=NV+1 $ TEMP=LAST

C J POINTS TO NEXT ELEMENT OF SOURCE LIST
    7 INDEX=COM(J).A.15 $ FTOP=LAST+1
```

```
C COPY TOP LIST TO WORKING AREA (FTOP THRU LAST)
    6   LAST=LAST+1 $ COM(LAST)=COM(J).A.MASK(21) $ J=J+1
        IF(LAST.EQ.PFL)STOP "HWA NEEDS MORE STORE (1)"
        IF((COM(J).A.15).EQ.INDEX)GO TO 6
   16   IF(J.EQ.TEMP)GO TO 8
        INDEX=INDEX+1
        IF((COM(J).A.15).NE.INDEX)GO TO 7
        FBOT=LAST+1 $ LTOP=LAST

C COPY BOTTOM LIST TO WORKING AREA (FBOT THRU LAST)
    9   LAST=LAST+1 $ COM(LAST)=COM(J).A.MASK(21) $ J=J+1
        IF(LAST.EQ.PFL)STOP "HWA NEEDS MORE STORE (2)"
        IF((COM(J).A.15).EQ.INDEX)GO TO 9
        I=LTOP $ LBOT=LAST

C SCAN TOP LIST (LAST TO FIRST)
   10   M=LBOT

C SCAN BOTTOM LIST (LAST TO FIRST)
   11   K=M

C SCAN ELEMENTS OF GROUP
   12   IF(NOTPOSS(I,K))GO TO 24

C CREATE NEW MEMBER
        MEMBER=JOINT(I,K) $ FLAG=.F. $ L=M

C SCAN ELEMENTS OF GROUP
   14   IF(FLAG)GO TO 15 $ IF(SUBSUMD(L))GO TO 24
   15   IF(SUBSUMS(L))29,18

C SUBSUMES AN EXISTING -
   29   IF(FLAG)GO TO 17 $ FLAG=.T.

C OVERWRITE EXISTING, OR
        COM(L)=COM(L).A..N.MASK(45).OR.MEMBER $ GO TO 18

C DELETE EXISTING
   17   COM(SAVEL)=
      " COM(SAVEL).A.MASK(45).OR.COM(L).A..N.MASK(45)
        COM(L)=MASK(1) $ IF(K.EQ.L)K=SAVEL $ L=SAVEL
   18   SAVEL=L $ L=COM(L).A..N.MASK(45) $ IF(L.NE.0)GO TO 14

C GROUP TESTING FINISHED
        IF(FLAG)GO TO 21

C RECORD NEW MEMBER
        IF(LAST.EQ.PFL)STOP "HWA NEEDS MORE STORE (3)"
        LAST=LAST+1 $ COM(LAST)=MEMBER
        COM(SAVEL)=COM(SAVEL).OR.LAST
   21   IF(COM(I).LT.0)GO TO 30
        IF(SUBSUMS(I))COM(I)=COM(I).OR.MASK(1)
   30   L=FTOP

C SCAN ELEMENTS OF TOP LIST
   22   IF(L.GT.LTOP)GO TO 24
        IF(SUBSUMS(L))GO TO 23 $ L=L+1 $ GO TO 22

C SUBSUMED - EITHER SET DELETE FLAG OR
   23   IF(L.GT.I)GO TO 31 $ COM(L)=COM(L).OR.MASK(1)
        L=L+1 $ GO TO 22
```

```
C DELETE IT NOW
   31  COM(L)=COM(LTOP) $ LTOP=LTOP-1 $ GO TO 22

C CONTINUE WITH GROUP
   24  K=COM(K).A..N.MASK(45) $ IF(K.NE.0)GO TO 12

C GROUP FINISHED
       M=M-1 $ IF(M.GE.FBOT)GO TO 11

C BOTTOM LIST FINISHED - DELETE ELEMENT I IF NECESSARY
       IF(COM(I).GE.0)GO TO 25
       COM(I)=COM(LTOP) $ LTOP=LTOP-1
   25  I=I-1 $ IF(I.GE.FTOP)GO TO 10

C TOP LIST FINISHED
       I=LTOP

C COMPRESS BOTTOM LIST
       DO 28 K=FBOT,LAST
       IF(COM(K).LT.0)GO TO 28 $ I=I+1
       COM(I)=COM(K).A.MASK(21)
   28  CONTINUE

C RESET POINTERS - BOTTOM LIST BECOMES TOP LIST
       LAST=I $ FTOP=LTOP+1
       GO TO 16

C FINISHED
C COMPRESS PI LIST, ELIMINATING THOSE WITH ROW-WEIGHT ZERO
    8  NPIMP=0 $ TEMP=TEMP+1 $ COM(NOMIN)=NO101
       DO 20 I=TEMP,LAST
       DO 26 J=1,NOMIN
       IF(PCOVERM(I,J))GO TO 27
   26  CONTINUE $ GO TO 20
   27  CONTINUE
       NPIMP=NPIMP+1 $ COM(NOMIN+NPIMP)=COM(I)
   20  CONTINUE
       WRITE(U,19)NPIMP
   19  FORMAT(* NUMBER OF PRIME IMPLICANTS:*I5)
       RETURN $ END
```

```
      LOGICAL FUNCTION INDEP(XIST,ADDIT)
      IMPLICIT INTEGER(A-Z)
      COMMON MINTERM,COM(2041)
      COMMON/NODE/COVL,COST,PRED,Z, NEWCOVL,NEWCOST,NEWN,NEWZ
     " ,PINO,SUCLIST(256),NEWUSL(256),NUSED,NSUCCS,NEWCOV(17)
      LOGICAL ADDIT,FLAG
      NM(I,J)=COM(I).A..N.MASK(J)
      SNM(I,J,K)=SHIFT(COM(I),J).A..N.MASK(K)
      EXCOVL(I)=SNM(I,8,55)
      EXN(I)=SNM(I,18,50)
      EXZ(I)=SNM(I,28,50)
      EXLOC(I)=NM(I,43)
      EXCOST(I)=SNM(I+1,33,27)
      EXSUCL(I)=EXLOC(I)+EXCOVL(I)

      INDEP=.T. $ TEMP=EXSUCL(XIST) $ TEMPB=NEWCOVL+1

C SKIP THIS FIRST TEST IF PROPOSED NODE IS SURE TO BE ADDED
      IF(ADDIT)GO TO 1 $ FLAG=.F.

C TEST THE PROPOSED NODE FOR DELETION
      IF(NEWCOST-EXCOST(XIST))2,3,4
    3 IF(NEWZ-EXZ(XIST))5,6,7
    6 IF(NEWN-EXN(XIST))8,9,10

C IDENTICAL NODES SO FAR, SET FLAG
C TO ENSURE TWO-WAY DOMINATION CHECKS
    9 FLAG=.T. $ GO TO 8
    4 IF(NEWZ.GT.EXZ(XIST))RETURN
    5 IF(NEWN.GT.EXN(XIST))RETURN
    8 TEMPA=EXCOVL(XIST)
      DO 11 I=1,TEMPA
      IF(COUNT(NEWCOV(TEMPB-I).A..N.COM(TEMP-I)).EQ.0)
     " GO TO 11
      IF(FLAG)GO TO 10 $ RETURN
   11 CONTINUE

C DELETE PROPOSED NODE
      NUSED=NUSED+1 $ NEWUSL(NUSED)=PINO $ INDEP=.F. $ RETURN

C TEST EXISTING NODE FOR DELETION
    1 IF(NEWCOST.GT.EXCOST(XIST))RETURN
      IF(NEWCOVL.EQ.0)GO TO 13
    2 IF(NEWZ.LT.EXZ(XIST))RETURN
    7 IF(NEWN.LT.EXN(XIST))RETURN
   10 DO 14 I=1,NEWCOVL
      IF(COUNT(COM(TEMP-I).A..N.NEWCOV(TEMPB-I)).NE.0)RETURN
   14 CONTINUE

C DELETE IT, SET FLAG INDICATING PROPOSED NODE WILL BE ADDED
   13 CALL DELETE(XIST) $ ADDIT=.T. $ RETURN $ END
```

```
      SUBROUTINE MOVE(FROM,TO)
      IMPLICIT INTEGER(A-Z)
      COMMON MINTERM,COM(2041)
      COMMON/STATS/NV,NO,LASTRAN,SEED, NOMIN,D,NPIMP,NEPI,
     " TERMINL,NONODES,THYME,TOTALG,TOTALL,TOTALT,
     " PFL,COSTMAX,GCOEF,LCOEF,PCOEF,TCOEF,PVALUES(10),SEL
      COMMON/POINERS/DEXAL,FINAL,IPOINT,NODE,NW,PIT,POINTER,S
     " ,SEXAL,START,THIS
      COMMON/NODE/COVL,COST,PRED,Z, NEWCOVL,NEWCOST,NEWN,NEWZ
     " ,FINO,SUCLIST(256),NEWUSL(256),NUSED,NSUCCS,NEWCOV(17)
      NM(I,J)=COM(I).A..N.MASK(J)
      SNM(I,J,K)=SHIFT(COM(I),J).A..N.MASK(K)
      EXTYPE(I)=SNM(I,3,57)
      EXCOVL(I)=SNM(I,8,55)
      EXPRED(I)=PFL-SNM(I,43,45)
      EXLOC(I)=NM(I,43)
      EXNSUC(I)=NM(I+1,54)
      EXSUCL(I)=EXLOC(I)+EXCOVL(I)
      EXUSL(I)=EXSUCL(I)+EXNSUC(I)
      EXBYTE(I,J)=SNM(J,I,45)

      IF(FROM.EQ.TO)RETURN
      PRE=EXPRED(FROM) $ I=0 $ IF(PRE.EQ.THIS)GO TO 103

C FROM IS ESTABLISHED:
C CORRECT THE SUCCESSOR LIST OF ITS PREDECESSOR
      FIRST=EXSUCL(PRE)
      MFROM=PFL-FROM $ MTO=PFL-TO
  104 DO 105 I=15,60,15
      TEMP=EXBYTE(I,FIRST) $ IF(TEMP.EQ.MFROM)GO TO 106
  105 CONTINUE $ FIRST=FIRST+1 $ GO TO 104
  106 COM(FIRST)=
     " SHIFT(SHIFT(COM(FIRST),I).A.MASK(45).OR.MTO,60-I)
      IF(FROM.NE.THIS)GO TO 107

C FROM IS THIS: CORRECT THIS AND PRED REFERENCES OF NEW NODES
      THIS=TO $ IF(NSUCCS.EQ.0)GO TO 110
      DO 108 I=1,NSUCCS
      TEMP=SUCLIST(I)
  108 COM(TEMP)=
     " SHIFT(SHIFT(COM(TEMP),43).A.MASK(45).OR.MTO,17)
      GO TO 110
  103 I=I+1 $ IF(SUCLIST(I).NE.FROM)GO TO 103

C FROM IS A NEW NODE: CORRECT APPROPRIATE SUCLIST MEMBER
      SUCLIST(I)=TO
      IF(FROM.NE.FINAL)GO TO 110

C FROM IS A TERMINAL NODE (NEW OR ESTABLISHED): CORRECT FINAL
  112 FINAL=TO $ GO TO 110
  107 IF(EXTYPE(FROM)-2)110,113,112

C FROM IS A TYPE 2 NODE: CORRECT PRED IF NECESSARY
  113 IF(PRED.EQ.FROM)PRED=TO
```

```
C ALSO CORRECT PRED REFERENCES OF ALL SUCCESSORS (IF ANY)
      FIRST=EXSUCL(FROM) $ LAST=EXUSL(FROM)-1
      IF(FIRST.GT.LAST)GO TO 110
      DO 114 J=FIRST,LAST
      DO 114 I=15,60,15
      TEMP=EXBYTE(I,J) $ IF(TEMP.EQ.0)GO TO 110
      TEMP=PFL-TEMP
  114 COM(TEMP)=
     * SHIFT(SHIFT(COM(TEMP),43).A.MASK(45).OR.MTO,17)

C FINALLY, CORRECT IPOINT IF NECESSARY, AND DO THE MOVING
  110 IF(IPOINT.EQ.FROM)IPOINT=TO
      COM(TO)=COM(FROM) $ COM(TO+1)=COM(FROM+1)
      RETURN $ END
```

```
      SUBROUTINE NET
C
C NODE STRUCTURE:        (BITS NUMBERED L TO R, 1 THRU 60)
C   INDEX ENTRY:  BITS QUANTITY              COMMENTS
C        WORD 1:   1- 3 TYPE OF NODE         0=STACK, 1=UNTICKED,
C                                            2=TICKED, 3=TERMINAL
C                  4- 8 WORDS IN COVER       0 FOR TYPE 3
C                 9-18 ONES IN COVER
C                19-28 POSN. OF 1ST ZERO
C                29-43 PREDECESSOR           RELATIVE TO PFL
C                44-60 LOCATION              RELATIVE TO COM
C        WORD 2:   1-33 COST                 INTEGER
C                  34-41 PACKET LENGTH
C                  42-54 PI. NUMBER
C                  55-60 WORDS IN SUC. LIST
C   MAIN ENTRY:          COVER               ZERO FILL,LEADING -0
C                                            WORDS SUPPRESSED
C          15-BIT BYTES SUCCESSOR LIST       ZERO FILL
C                                            RELATIVE TO PFL
C          15-BIT BYTES USED SIBLING LIST    ZERO FILL
C
      IMPLICIT INTEGER(A-Z)
      COMMON MINTERM(1),COM(2041)
      COMMON/SELFLAG/SELFLAG
      COMMON/ZBASE/ZBASE
      COMMON/TEMPARY/RTEMPRY(40)
      COMMON/OPUNIT/U
      COMMON/STATS/NV,NO,LASTRAN,SEED, NOMIN,D,NPIMP,NEPI,
     " TERMINL,NONODES,THYME,TOTALG,TOTALL,TOTALT,
     " PFL,COSTMAX,GCOEF,LCOEF,PCOEF,TCOEF,PVALUES(10),SEL
      COMMON/POINERS/DEXAL,FINAL,IPOINT,NODE,NW,PIT,POINTER,S
     " ,SEXAL,START,THIS
      COMMON/NODE/COVL,COST,PRED,Z, NEWCOVL,NEWCOST,NEWN,NEWZ
     " ,PINO,SUCLIST(256),NEWUSL(256),NUSED,NSUCCS,NEWCOV(17)
      COMMON/DATA/H,NO101,ZERO,NO120
      LOGICAL SELFLAG,SELECT,INDEF,ADDIT,ONEZERO
      DIMENSION ISLIST(500)
      REAL SIZE,SEL,THYME,SECOND,RTEMPRY
      NM(I,J)=COM(I).A..N.MASK(J)
      SNM(I,J,K)=SHIFT(COM(I),J).A..N.MASK(K)
      EXTYPE(I)=SNM(I,3,57)
      EXCOVL(I)=SNM(I,8,55)
      EXN(I)=SNM(I,18,50)
      EXZ(I)=SNM(I,28,50)
      EXPRED(I)=PFL-SNM(I,43,45)
      EXLOC(I)=NM(I,43)
      EXCOST(I)=SNM(I+1,33,27)
      EXLEN(I)=SNM(I+1,41,52)
      EXPINO(I)=SNM(I+1,54,47)
      EXNSUC(I)=NM(I+1,54)
      EXSUCL(I)=EXLOC(I)+EXCOVL(I)
      EXUSL(I)=EXSUCL(I)+EXNSUC(I)
      EXLAST(I)=EXLOC(I)+EXLEN(I)-1
      EXBYTE(I,J)=SNM(J,I,45)
      XPCOST(I)=NM(NOMIN+I,27)
      ZSCORE(I)=COM(ZBASE+EXZ(I))
```

```
C INITIALISE AND SET UP 1ST NODE
      CALL SECOND(THYME) $ NONODES=CUSLMAX=1 $ S=NW+2
      SIZE=PFL-START
      IF(SELFLAG)SEL=0.
      TERMINL=COSTMAX $ FINAL=0 $ DEXAL=SEXAL=THIS=PFL-1
      COM(DEXAL)=
     * SHIFT(1,57).OR.SHIFT(NW,52).OR.SHIFT(1,32).OR.START+1
      COM(PFL)=SHIFT(NW,19) $ POINTER=START+NW
      DO 82 I=1,NW
   82 COM(START+I)=0

C SET PARAMETER VALUES FOR THIS NODE AND
C CREATE ILLEGAL SUCCESSOR LIST
   80 Z=EXZ(THIS) $ COST=EXCOST(THIS) $ COVL=EXCOVL(THIS)
      PRED=EXPRED(THIS) $ THPINO=EXPINO(THIS)
      NODE=EXLOC(THIS) $ NILSUC=NSUCCS=NUSED=0 $ IPOINT=THIS
  100 J=EXUSL(IPOINT) $ L=EXLAST(IPOINT)
      IF(J.GT.L)GO TO 101
      DO 102 K=J,L
      DO 102 I=15,60,15
      TEMP=EXBYTE(I,K) $ IF(TEMP.EQ.0)GO TO 101
      IF(NILSUC.EQ.500)STOP "ISLIST TOO SHORT"
      NILSUC=NILSUC+1 $ ISLIST(NILSUC)=TEMP
  102 CONTINUE
  101 IPOINT=EXPRED(IPOINT) $ IF(IPOINT.NE.PFL)GO TO 100
      A=SNM(Z-1,36,45) $ B=SNM(Z-1,51,45)
      WORD=(Z+59)/60 $ BIT=Z-WORD*60 $ WORD=WORD+PIT+(A-2)*NW
      CALL PAUSE("NETREADY")
      DO 1 PINO=A,B
      WORD=WORD+NW
      IF(SNM(WORD,BIT,59).EQ.0)GO TO 1

C MAKE SURE COST IS LESS THAN TERMINL
      NEWCOST=COST+XPCOST(PINO)
      IF(NEWCOST.GE.TERMINL)GO TO 1
      IF((NEWCOST.A.MASK(27)).NE.0)STOP "NODE COST EXCESSIVE"
      IF(NILSUC.EQ.0)GO TO 10

C MAKE SURE ITS NOT ILLEGAL
      DO 13 J=1,NILSUC
      IF(PINO.EQ.ISLIST(J))1,13
   13 CONTINUE

C COMPUTE NEWN,NEWZ,NEWCOVL,NEWCOV
   10 ONEZERO=.F. $ NEWN=NOMIN $ L=COVL-1 $ NEWCOVL=0
      DO 9 I=ZERO,L
      TEMPA=COM(WORD+I).OR.COM(NODE+I) $ TEMP=COUNT(TEMPA)
      IF(ONEZERO)GO TO 8 $ IF(TEMP.EQ.60)GO TO 9
      ONEZERO=.T. $ NEWZ=-1
    7 NEWZ=NEWZ+1 $ IF(SHIFT(TEMPA,NEWZ).LT.0)GO TO 7
      NEWN=(NW+I-COVL)*60 $ NEWZ=NEWZ+NEWN+1
    8 NEWN=NEWN+TEMP $ NEWCOVL=NEWCOVL+1
      NEWCOV(NEWCOVL)=TEMPA
    9 CONTINUE
      CALL PAUSE("TRYTHIS")
      ADDIT=NEWN.EQ.NOMIN
      IF(ADDIT)6,12
    6 NEWCOVL=0 $ NEWZ=NOMIN+1 $ TERMINL=NEWCOST
      IF(SEL.EQ.0.)SEL=1.-S/SIZE
      IF(FINAL.NE.0)CALL DELETE(FINAL)
   12 I=NSUCCS
```

```
   14  IF(I.EQ.0)GO TO 2
       IF(INDEP(SUCLIST(I),ADDIT))3,1
    3  I=I-1 $ GO TO 14
    2  DO 5 I=SEXAL,PFL,2
       IF(EXPRED(I).EQ.THIS)GO TO 5
       IF(INDEP(I,ADDIT))5,1
    5  CONTINUE
       CALL ADD(CUSLMAX)
    1  CONTINUE

C ALL SUCCESSORS FOR THIS NODE HAVE NOW BEEN FOUND
       CALL PAUSE("THATSALL")
       IF(NSUCCS.NE.0)GO TO 35 $ IF(SEL.EQ.0.)SEL=1.-S/SIZE
       CALL DELETE(THIS) $ GO TO 75
   35  IF(PRED.EQ.PFL)GO TO 99
       FIRST=EXSUCL(PRED) $ LENM1=EXUSL(PRED)-FIRST-1
       MTHIS=PFL-THIS

C PUT THIS PI.NO INTO USL OF EACH TYPE 1 SIBLING OF THIS NODE
       DO 90 J=ZERO,LENM1
       DO 90 I=15,60,15
       TEMP=EXBYTE(I,FIRST+J) $ IF(TEMP.EQ.0)GO TO 99
       IF(TEMP.EQ.MTHIS)GO TO 90 $ TEMP=PFL-TEMP
       IF(EXTYPE(TEMP).NE.1)GO TO 90
       L=EXLAST(TEMP) $ IF(EXUSL(TEMP).GT.L)GO TO 91
       DO 92 K=30,60,15
       IF(EXBYTE(K,L).NE.0)GO TO 92
       COM(L)=COM(L).OR.SHIFT(THPINO,60-K) $ GO TO 90
   92  CONTINUE
   91  CALL SHORTY(1,TEMP) $ POINTER=POINTER+1
       COM(POINTER)=SHIFT(THPINO,45)
       COM(TEMP+1)=COM(TEMP+1)+SHIFT(1,19)
       FIRST=EXSUCL(PRED)
   90  CONTINUE

C ADD THE SUCCESSOR LIST FOR THIS NODE AND INCREMENT ITS TYPE
   99  EXTRA=(NSUCCS+3)/4 $ CALL SHORTY(EXTRA,THIS)
       COM(THIS)=COM(THIS)+SHIFT(1,57)
       FIRST=EXUSL(THIS) $ LAST=EXLAST(THIS)
       IF(FIRST.GT.LAST)GO TO 31
       DO 32 I=FIRST,LAST
   32  COM(FIRST+LAST-I+EXTRA)=COM(FIRST+LAST-I)
   31  COM(THIS+1)=COM(THIS+1)+SHIFT(EXTRA,19).OR.EXTRA
       POINTER=POINTER+EXTRA $ FIRST=FIRST-1 $ J=0
       DO 33 I=1,NSUCCS
       TEMP=SUCLIST(I)
       IF(TEMP.EQ.FINAL)GO TO 4
       S=S-CUSLMAX $ COM(TEMP+1)=COM(TEMP+1)-SHIFT(CUSLMAX,19)
       A=EXCOST(TEMP) $ K=EXZ(TEMP)+ZBASE
   11  IF(A.GE.COM(K))GO TO 4 $ COM(K)=A $ K=K-1 $ GO TO 11
    4  IF(J.NE.0)GO TO 34
       FIRST=FIRST+1 $ COM(FIRST)=SHIFT(PFL-TEMP,45)
       J=45 $ GO TO 33
   34  J=J-15
       COM(FIRST)=COM(FIRST).OR.SHIFT(PFL-TEMP,J)
   33  CONTINUE
       IF(NUSED.EQ.0)GO TO 75
       EXTRA=J=0
```

```
C COMPRESS NEWUSL AND CREATE A USL FOR EACH NEW NODE
      DO 37 I=1,NUSED
      IF(J.NE.0)GO TO 38 $ J=45 $ EXTRA=EXTRA+1
      NEWUSL(EXTRA)=SHIFT(NEWUSL(I),45) $ GO TO 37
   38 J=J-15
      NEWUSL(EXTRA)=NEWUSL(EXTRA).OR.SHIFT(NEWUSL(I),J)
   37 CONTINUE
      IF(EXTRA.LE.CUSLMAX)GO TO 20

C SPACE LEFT BY ADD INADEQUATE
      TEMP=SUCLIST(NSUCCS) $ SUCLIST(NSUCCS)=SUCLIST(1)
      SUCLIST(1)=TEMP
      DO 36 I=1,NSUCCS
      TEMP=SUCLIST(I) $ IF(TEMP.EQ.FINAL)GO TO 36
      CALL SHORTY(EXTRA,TEMP)
      DO 39 J=1,EXTRA
   39 COM(POINTER+J)=NEWUSL(J)
      POINTER=POINTER+EXTRA
      COM(TEMP+1)=COM(TEMP+1)+SHIFT(EXTRA,19)
   36 CONTINUE $ CUSLMAX=EXTRA $ GO TO 75

C SPACE LEFT BY ADD IS ADEQUATE
   20 DO 26 I=1,NSUCCS
      TEMP=SUCLIST(I) $ IF(TEMP.EQ.FINAL)GO TO 26
      TEMPA=EXLAST(TEMP)
      S=S+EXTRA
      DO 29 J=1,EXTRA
   29 COM(TEMPA+J)=NEWUSL(J)
      COM(TEMP+1)=COM(TEMP+1)+SHIFT(EXTRA,19)
   26 CONTINUE

C SELECT A NEW NODE TO CONTINUE BRANCHING
   75 IF(SELECT(SIZE))GO TO 80

C FINISH OFF
      CALL SOLUTIO
      TERMINL=SUCLIST(TOTALT+1)
      CALL PAUSE("SOLUTION")
      THYME=SECOND(TEMP)-THYME
      RETURN $ END
```

```
      SUBROUTINE OUT(ARRAY,AA,BB,DUMTYPE,ADDTYPE)
      IMPLICIT INTEGER(A-Z)
      COMMON MINTERM(1),COM(2040),DONTCAR(1)
      COMMON/STATS/NV,NO,LASTRAN,SEED, NOMIN,D,NPIMP,NEPI,
     " TERMINL,NONODES,THYME,TOTALG,TOTALL,TOTALT,
     " PFL,COSTMAX,GCOEF,LCOEF,PCOEF,TCOEF,PVALUES(10),SEL
      COMMON/POINERS/DEXAL,FINAL,IPOINT,NODE,NW,PIT,POINTER,S
     " ,SEXAL,START,THIS
      COMMON/NODE/COVL,COST,PRED,Z, NEWCOVL,NEWCOST,NEWN,NEWZ
     " ,PINO,SUCLIST(256),NEWUSL(256),NUSED,NSUCCS,NEWCOV(17)
      COMMON/DATA/H,NO101,ZERO,NO120
      COMMON/OPUNIT/U
      COMMON/PROCESS/AT,FLLESS1,LOCFMIN
      COMPLEX PROCESS,CMP
      DIMENSION ARRAY(1),DISPF(10),FD(2),FLOTEF(2),FORM(10),
     " INCLIST(10),OCTF(2),LSTR(2)
      DATA FORM(1)/"(24(2XR1))(18(2XR2))(14(2XR3))(12(2XR4))"
     " /,FORM(5)/"(10(2XR5)) (9(2XR6)) (8(2XR7)) (7(2XR8))"/,
     " FORM(9)/" (6(2XR9))(6(2XR10))"/,DISPF0,INCLIST,DISPF(1)
     " /"(1X,05,A1,",",22,16,13,11,9,8,7,6,6," 22(2XR1))"/,
     " DISPF(2)/" 16(2XR2)) 13(2XR3)) 11(2XR4)) 9(2X,R5))"/,
     " DISPF(6)/" 8(2X,R6)) 7(2X,R7)) 6(3X,R8)) 6(2X,R9))"/,
     " DISPF(10),FLOTEF(1)/" 6(1XR10))","(1X05,A1,1P6G11.3)"/,
     " OCTF(1)/"(1X05,A2,O20,2022)"/

C PRINT A SOLUTION
      K=FORM(NV)
      WRITE(6,8)TOTALT,TOTALL,TOTALG,FLOAT(SUCLIST(TOTALT+1))
      WRITE(U,K)(DISPLAY(SUCLIST(I)),I=1,TOTALT)
    8 FORMAT(* SOLUTION:*I4* TERMS,*I6* LITERALS,*I4
     " * GATES, TOTAL COST:*1PG12.5)
      RETURN

      ENTRY DUMPF

C PARAMETERS:
C  ARRAY    ARRAY WITH ADDRESS 1 (HENCE SUBSCRIPT=ADDRESS)
C  AA       FIRST ADDRESS
C  BB       LAST ADDRESS
C  DUMTYPE  FORMAT OF LOCATION CONTENTS PRINT:
C           0=OCTAL, 1=DISPLAY, 2=FLOATING POINT
C  ADDTYPE  FORMAT OF PRINTED ADDRESSES:
C           0=ABSOLUTE( ) 1=RELATIVE(.) 2=MINUS(*)

C PREPARE TO DUMP STORAGE LOCATIONS
      LOCFMIN=LOCF(MINTERM(1)) $ FLLESS1=LOCFMIN+PFL
      AT=ADDTYPE
      IF(AA.LE.FLLESS1)GO TO 13 $ WRITE(6,4) $ RETURN
    4 FORMAT(* --- ERROR: ADDRESS OUT OF RANGE*)
    6 WRITE(6,7) $ RETURN
    7 FORMAT(* --- ERROR: NV OUT OF RANGE*)
   13 BB=MIN0(FLLESS1,MAX0(AA,BB)) $ IF(DUMTYPE-1)17,3,2
```

```
C SET UP THE APPROPRIATE VARIABLE FORMAT
   17  DO 5 I=1,2
    5  FD(I)=OCTF(I) $ INC=3 $ GO TO 9
    2  DO 1 I=1,2
    1  FD(I)=FLOTEF(I) $ INC=6 $ GO TO 9
    3  IF(NV.LE.0.OR.NV.GE.11)GO TO 6
       INC=INCLIST(NV) $ FD(1)=DISPF0
       FD(2)=DISPF(NV)
    9  C=BB-INC $ IF(C.LT.AA)GO TO 10
       IF(DUMTYPE.EQ.1)GO TO 11
       WRITE(6,FD)
      * (PROCESS(I),(ARRAY(I+K-1),K=1,INC),I=AA,C,INC)
       GO TO 10
   11  DO 14 I=AA,C,INC
   14  WRITE(6,FD)PROCESS(I),(DISPLAY(ARRAY(I+K-1)),K=1,INC)
   10  C=AA+(BB-AA)/INC*INC $ IF(DUMTYPE.EQ.1)GO TO 12

C CMP USED HERE TO OBVIATE A COMPILER FAULT
       CMP=PROCESS(C)
       WRITE(6,FD)CMP,(ARRAY(I),I=C,BB) $ RETURN
   12  WRITE(6,FD)PROCESS(C),(DISPLAY(ARRAY(I)),I=C,BB)
       RETURN

       ENTRY LISTER

C PARAMETERS:
C  ARRAY(1) QUANTITIES DESIRED:
C           BIT60=MINTERMS, BIT59=DONTCARES, BIT58=PI.S
C  AA       MINTERM/DONTCARE FORMAT:
C           -1=DISPLAY, 0=DECIMAL, 1=OCTAL
C  BB       PAGEWIDTH

       IF((ARRAY(1).A..N.MASK(58)).EQ.0)GO TO 18
       C=.N.MASK(60-NV) $ W=4 $ IF(AA)16,26,30

C SET DISPLAY FORMAT
   16  INC=(BB-NV)/(NV+2)
       ENCODE(20,15,LSTR)NV,INC,NV $ INC=INC+1
       GO TO 27

C SET INTEGER FORMAT
   26  TP=1HI $ IF(C.GE.1000)GO TO 19
       W=W-1 $ IF(C.GE.100)GO TO 19
       W=W-1 $ IF(C.GE.10)GO TO 19
       W=W-1 $ GO TO 19

C SET OCTAL FORMAT
   30  TP=1HO $ IF(C.GE.1000B)GO TO 19
       W=W-1 $ IF(C.GE.100B)GO TO 19
       W=W-1 $ IF(C.GE.10B)GO TO 19
       W=W-1
   19  INC=(BB-W)/(W+2)
       ENCODE(20,20,LSTR)TP,W,INC,TP,W $ INC=INC+1
   20  FORMAT(*(X*A1,I1*,*I3*(2X*A1,I1*))*)
   27  IF((ARRAY(1).A..N.MASK(59)).EQ.0)GO TO 22
```

```
C PRINT MINTERMS
      WRITE(U,21)9HMINTERMS:,NOMIN
   21 FORMAT(A11,I6)
      DO 25 K=1,NOMIN,INC
      C=MINO(INC-1,NOMIN-K)
      IF(AA.GE.0)GO TO 28
      WRITE(U,LSTR)(DISPLAY(QMXPAND(MINTERM(K+I))),I=ZERO,C)
      GO TO 25
   28 WRITE(U,LSTR)(MINTERM(K+I),I=ZERO,C)
   25 CONTINUE
      IF(SHIFT(ARRAY(1),58).GE.0)GO TO 18

C PRINT DONTCARES
   22 WRITE(U,21)10HDONTCARES:,D
      IF(D.EQ.0)GO TO 18
      DO 24 K=1,D,INC
      C=MINO(INC-1,D-K)
      IF(AA.GE.0)GO TO 29
      WRITE(U,LSTR)(DISPLAY(QMXPAND(DONTCAR(K+I))),I=ZERO,C)
      GO TO 24
   29 WRITE(U,LSTR)(DONTCAR(K+I),I=ZERO,C)
   24 CONTINUE
   18 IF(SHIFT(ARRAY(1),57).GE.0)RETURN

C REGERERATE PRIME IMPLICANTS AND SET DISPLAY FORMAT
      SAVEU=U $ U=4 $ CALL HWA $ U=SAVEU
      INC=(BB-NV)/(NV+2)
      ENCODE(20,15,LSTR)NV,INC,NV $ INC=INC+1
   15 FORMAT(*(XR*I2*,*I2*(2XR*I2*))*)

C PRINT PI.S
      WRITE(U,21)5HPI.S:,NPIMP
      DO 23 K=1,NPIMP,INC
      C=MINO(INC-1,NPIMP-K)
   23 WRITE(U,LSTR)(DISPLAY(COM(NOMIN+K+I)),I=ZERO,C)
      RETURN $ END
```

```
$PITABLE
      FUNCTION PITABLE(TYPE)

C QUANTITIES TO BE PRINTED AS A FUNCTION OF TYPE:
C  TYPE    PI    PI.NO.    O/P DEVICE
C .LE.1    YES   YES        TERMINAL
C      2   NO    YES        TERMINAL
C      3   NO    NO         TERMINAL
C      4   YES   YES        PRINTER
C      5   NO    YES        PRINTER
C      6   NO    NO         PRINTER
C .GE.7    NO    YES        PRINTER WITH MULTIPLE LINES PER ROW

      IMPLICIT INTEGER(A-Z)
      COMMON MINTERM(1),COM(2041)
      COMMON/STATS/NV,NO,LASTRAN,SEED, NOMIN,D,NPIMP,NEPI,
     " TERMINL,NONODES,THYME,TOTALG,TOTALL,TOTALT,
     " PFL,COSTMAX,GCOEF,LCOEF,PCOEF,TCOEF,PVALUES(10),SEL
      COMMON/POINERS/DEXAL,FINAL,IPOINT,NODE,NW,PIT,POINTER,S
     " ,SEXAL,START,THIS
      COMMON/NODE/COVL,COST,PRED,Z, NEWCOVL,NEWCOST,NEWN,NEWZ
     " ,PINO,SUCLIST(256),NEWUSL(256),NUSED,NSUCCS,NEWCOV(17)
      COMMON/DATA/H,NO101,ZERO,NO120
      DIMENSION FMT(3),CHAR(30)
      DATA TTY,LP/72,136/

      IF(NOMIN.LE.0)GO TO 8
      JJ=(NOMIN+9)/10 $ NW=(NOMIN+59)/60 $ K=NOMIN+NPIMP+1

C OVERRIDE TYPE REQUESTED IF TABLE WILL NOT FIT
      IF(NOMIN.LE.TTY-4-NV)GO TO 1
      IF(NOMIN.LE.TTY-3)GO TO 2
      IF(NOMIN.LT.TTY)GO TO 3
      IF(NOMIN.LE.LP-4-NV)GO TO 4
      IF(NOMIN.LE.LP-3)GO TO 5
      IF(NOMIN.LT.LP)GO TO 6
      IF(TYPE.GE.7)GO TO 7
    8 WRITE(6,9)TYPE,NOMIN
    9 FORMAT(* ---- WARNING: NO TYPE*I2
     " * TABLE CREATED FOR NOMIN =*I4)

C FUNCTION VALUE IS THE TYPE OF THE TABLE ACTUALLY PRINTED
      PITABLE=0 $ RETURN

    1 IF(TYPE.GT.1)GO TO 2
      ENCODE(30,10,FMT)NV,JJ-1,MOD(NOMIN-1,10)+1
   10 FORMAT(*(I3,XR*I2,3H*.*I2*A10,A*I2*)*)
      DO 12 I=1,NPIMP
      CALL DISCOV(COM(K),CHAR,NOMIN) $ K=K+NW
   12 WRITE(6,FMT)I,DISPLAY(COM(NOMIN+I)),(CHAR(J),J=1,JJ)
      PITABLE=1 $ RETURN

    2 IF(TYPE.GT.2)GO TO 3
      DO 13 I=1,NPIMP
      CALL DISCOV(COM(K),CHAR,NOMIN) $ K=K+NW
   13 WRITE(6,11)I,(CHAR(J),J=1,JJ)
   11 FORMAT(I3*.*6A10,A9)
      PITABLE=2 $ RETURN
```

```
 3  IF(TYPE.GT.3)GO TO 4
    DO 14 I=1,NPIMP
    CALL DISCOV(COM(K),CHAR,NOMIN) $ K=K+NW
14  WRITE(6,15)(CHAR(J),J=1,JJ)
15  FORMAT(* ,*7A10,A1)
    PITABLE=3 $ RETURN

 4  IF(TYPE.GT.4)GO TO 5
    ENCODE(30,10,FMT)NV,JJ-1,MOD(NOMIN-1,10)+1
    DO 17 I=1,NPIMP
    CALL DISCOV(COM(K),CHAR,NOMIN) $ K=K+NW
17  WRITE(7,FMT)I,DISPLAY(COM(NOMIN+I)),(CHAR(J),J=1,JJ)
    PITABLE=4

16  WRITE(6,18) $ RETURN
18  FORMAT(* PI TABLE COPIED TO PRINT FILE*)

 5  IF(TYPE.GT.5)GO TO 6
    DO 19 I=1,NPIMP
    CALL DISCOV(COM(K),CHAR,NOMIN) $ K=K+NW
19  WRITE(7,20)I,(CHAR(J),J=1,JJ)
20  FORMAT(I3*,*13A10,A3)
    PITABLE=5 $ GO TO 16

 6  IF(TYPE.GT.6)GO TO 7
    DO 21 I=1,NPIMP
    CALL DISCOV(COM(K),CHAR,NOMIN) $ K=K+NW
21  WRITE(7,22)(CHAR(J),J=1,JJ)
22  FORMAT(* ,*13A10,A5)
    PITABLE=6 $ GO TO 16

 7  IF(NOMIN.GT.300)GO TO 8
    DO 23 I=1,NPIMP
    CALL DISCOV(COM(K),CHAR,NOMIN) $ K=K+NW
23  WRITE(7,24)I,(CHAR(J),J=1,JJ)
24  FORMAT(I6*,*13A10/(6X*,*13A10))
    PITABLE=7 $ GO TO 16 $ END
```

```
      SUBROUTINE POS
      IMPLICIT INTEGER(A-Z)
      COMMON MINTERM,COM(2041)
      DIMENSION NUMZERO(5),NUM(4)
      LOGICAL FLAG
      COMMON/STATS/NV,NO,LASTRAN,SEED, NOMIN,D,NPIMP,NEPI,
     " TERMINL,NONODES,THYME,TOTALG,TOTALL,TOTALT,
     " PFL,COSTMAX,GCOEF,LCOEF,PCOEF,TCOEF,PVALUES(10),SEL
      COMMON/POINERS/DEXAL,FINAL,IPOINT,NODE,NW,PIT,POINTER,S
     " ,SEXAL,START,THIS
      COMMON/OPUNIT/U
      COMMON/DATA/H,NO101,ZERO,NO120
      EQUIVALENCE (NUMZERO(2),NUM(1))
      NM(I,J)=COM(I).A..N.MASK(J)
      SNM(I,J,K)=SHIFT(COM(I),J).A..N.MASK(K)
      EXTYPE(I)=SNM(I,3,57)
      EXCOVL(I)=SNM(I,8,55)
      EXN(I)=SNM(I,18,50)
      EXZ(I)=SNM(I,28,50)
      EXPRED(I)=PFL-SNM(I,43,45)
      EXLOC(I)=NM(I,43)
      EXCOST(I)=SNM(I+1,33,27)
      EXLEN(I)=SNM(I+1,41,52)
      EXPINO(I)=SNM(I+1,54,47)
      EXNSUC(I)=NM(I+1,54)
      EXSUCL(I)=EXLOC(I)+EXCOVL(I)
      EXUSL(I)=EXSUCL(I)+EXNSUC(I)
      EXLAST(I)=EXLOC(I)+EXLEN(I)-1
      EXBYTE(I,J)=SNM(J,I,45)

      IF(PIT.GE.POINTER)                                  CALL FAULT( 1)
      IF(POINTER.GE.SEXAL)                                CALL FAULT( 2)
      IF(SEXAL.GT.DEXAL)                                  CALL FAULT( 3)
      IF(DEXAL.GE.PFL)                                    CALL FAULT( 4)
      IF(MOD(DEXAL-SEXAL,2).NE.0)                         CALL FAULT( 5)
      IF(MOD(PFL-DEXAL,2).NE.1)                           CALL FAULT( 6)
      DO 1 I=ZERO,4
    1 NUM(I)=0 $ SCOUNT=0
      DO 2 I=SEXAL,PFL,2

C TT=0,1,2,3,4
      NN=EXN(I) $ ZZ=EXZ(I) $ CC=EXCOST(I) $ PP=EXPRED(I)
      PNO=EXPINO(I)
      TT=EXTYPE(I) $ IF(TT.LT.0)                          CALL FAULT( 7)
      IF(TT.GT.3)                                         CALL FAULT( 8)
      IF(ZZ.EQ.1)TT=4 $ NUM(TT)=NUM(TT)+1
      LL=EXLEN(I) $ SCOUNT=SCOUNT+LL+2
      IF(TT-3)3,4,16

C TT=0,1,2
    3 IF(EXCOVL(I).NE.NW-(ZZ-1)/60)                       CALL FAULT(10)
      IF(NN.GE.NOMIN)                                     CALL FAULT(11)
      IF(ZZ.GT.NN+1)                                      CALL FAULT(12)
      IF(CC.GE.TERMINL)                                   CALL FAULT(13)
      IF(EXLOC(I).LE.PIT)                                 CALL FAULT( 9)
      IF(EXLAST(I).GT.POINTER)                            CALL FAULT(14)
      IF(COUNT(-COM(EXLOC(I))).EQ.0)                      CALL FAULT(15)
      IF(TT.NE.0)GO TO 5
```

```
C TT=0
      IF(NN.LE.0)                                  CALL FAULT(16)
      IF(ZZ.LE.1)                                  CALL FAULT(17)
      IF(CC.LE.0)                                  CALL FAULT(18)
      IF(I.GE.DEXAL)                               CALL FAULT(19)

C TT=0,4
    6 IF(PF.NE.PFL)                                CALL FAULT(20)
      IF(PNO.NE.0)                                 CALL FAULT(21)

C TT=0,1,2,3,4
    7 IF(EXTYPE(I).EQ.2)GO TO 21

C TYPE=0,1,3
      IF(EXNSUC(I).NE.0)                           CALL FAULT(41)
      GO TO 2

C TT=3
    4 IF(EXCOVL(I).NE.0)                           CALL FAULT(22)
      IF(NN.NE.NOMIN)                              CALL FAULT(23)
      IF(ZZ.NE.NOMIN+1)                            CALL FAULT(24)
      IF(CC.NE.TERMINL)                            CALL FAULT(25)
      IF(LL.NE.0)                                  CALL FAULT(26)
      IF(I.NE.FINAL)                               CALL FAULT(27)
      IF(EXLOC(I).NE.0)                            CALL FAULT(56)
      IF(EXNSUC(I).NE.0)                           CALL FAULT(57)
      GO TO 8

C TT=4
   16 IF(EXCOVL(I).NE.NW)                          CALL FAULT(47)
      IF(NN.NE.0)                                  CALL FAULT(49)
      IF(CC.NE.0)                                  CALL FAULT(51)
      IF(COM(EXLOC(I)).NE.0)                       CALL FAULT(55)
```

```
C TT=1,2,4
    5  FIRST=EXUSL(I) $ LAST=EXLAST(I)
       IF(FIRST.GT.LAST)GO TO 9

C CHECK USED SIBLING LIST
       F=EXSUCL(PP) $ L=EXUSL(PP)-1
       IF(F.GT.L)                                          CALL FAULT(28)
       DO 10 JJ=FIRST,LAST
       DO 10 II=15,60,15
       TEMP=EXBYTE(II,JJ) $ IF(TEMP.NE.0)GO TO 11
       IF(JJ.NE.LAST)                                      CALL FAULT(29)
       IF(II.EQ.15)                                        CALL FAULT(48)
       IF(NM(JJ,II).NE.0)                                  CALL FAULT(50)
       GO TO 12
   11  IF(TEMP.EQ.PNO)                                     CALL FAULT(30)
       FLAG=.F.
       DO 13 J=F,L
       DO 13 IJ=15,60,15
       TEMPA=EXBYTE(IJ,J) $ IF(TEMPA.EQ.0)GO TO 10
       TEMPA=PFL-TEMPA
       IF(EXPINO(TEMPA).NE.TEMP)GO TO 13
       IF(FLAG)                                            CALL FAULT(31)
       IF(COM(TEMPA).LT.0)                                 CALL FAULT(52)
       FLAG=.T. $ COM(TEMPA)=COM(TEMPA).OR.MASK(1)
   13  CONTINUE
   10  CONTINUE
   12  DO 15 J=F,L
       DO 15 IJ=15,60,15
       TEMPA=EXBYTE(IJ,J) $ IF(TEMPA.EQ.0)GO TO 9
       TEMPA=PFL-TEMPA
       COM(TEMPA)=NM(TEMPA,1)
   15  CONTINUE
    9  IF(TT.EQ.4)GO TO 6

C TT=1,2
       IF(NN.LE.EXN(PP))                                   CALL FAULT(33)
       IF(ZZ.LE.EXZ(PP))                                   CALL FAULT(34)
       IF(CC.LT.EXCOST(PP))                                CALL FAULT(35)

C TT=1,2,3
    8  IF(PP.LT.DEXAL)                                     CALL FAULT(58)
       F=EXSUCL(PP) $ L=EXUSL(PP)-1
       IF(F.GT.L)                                          CALL FAULT(36)

C CHECK I IS IN SUCCESSOR LIST OF PRED
       FLAG=.F. $ MI=PFL-I
       DO 17 JJ=F,L
       DO 17 II=15,60,15
       TEMP=EXBYTE(II,JJ) $ IF(TEMP.EQ.0)GO TO 18
       IF(TEMP.NE.MI)GO TO 17
       IF(FLAG)                                            CALL FAULT(37)
       FLAG=.T.
   17  CONTINUE
   18  IF(.N.FLAG)                                         CALL FAULT(38)
       J=I $ IF(PNO.EQ.0)                                  CALL FAULT(39)
```

```
C CHECK PNO NOT ILLEGAL
   19  J=EXPRED(J) $ IF(J.EQ.PFL)GO TO 7
       F=EXUSL(J) $ L=EXLAST(J)
       IF(F.GT.L)GO TO 19
       DO 20 JJ=F,L
       DO 20 II=15,60,15
       TEMP=EXBYTE(II,JJ) $ IF(TEMP.EQ.0)GO TO 19
       IF(TEMP.EQ.PNO)                                  CALL FAULT(40)
   20  CONTINUE $ GO TO 19

C TYPE=2
   21  F=EXSUCL(I) $ L=EXUSL(I)-1
       IF(F.GT.L)                                       CALL FAULT(42)

C CHECK SUCCESSOR LIST
       DO 22 JJ=F,L
       DO 22 II=15,60,15
       TEMP=EXBYTE(II,JJ) $ IF(TEMP.NE.0)GO TO 25
       IF(JJ.NE.L)                                      CALL FAULT(54)
       IF(II.EQ.15)                                     CALL FAULT(32)
       IF(NM(JJ,II).NE.0)                               CALL FAULT(53)
       GO TO 23
   25  TEMP=PFL-TEMP
       IF(TEMP.LT.DEXAL)                                CALL FAULT(59)
       IF(COM(TEMP).LT.0)                               CALL FAULT(43)
       IF(EXPRED(TEMP).NE.I)                            CALL FAULT(44)
       COM(TEMP)=-COM(TEMP)
   22  CONTINUE
   23  DO 24 JJ=F,L
       DO 24 II=15,60,15
       TEMP=EXBYTE(II,JJ) $ IF(TEMP.EQ.0)GO TO 2
       TEMP=PFL-TEMP
       COM(TEMP)=-COM(TEMP)
   24  CONTINUE

    2  CONTINUE
       IF(NUM(4).NE.1)                                  CALL FAULT(45)
       IF(SCOUNT.NE.S)                                  CALL FAULT(46)
       IF(NUM(3).EQ.0.A.FINAL.NE.0)                     CALL FAULT(60)
       DO 27 I=ZERO,3
   27  NUM(4)=NUM(4)+NUM(I)

C PRINT NUMBER OF NODES OF TYPE 0,1,2,3 AND TOTAL RESPECTIVELY
       WRITE(U,26)(NUM(I),I=ZERO,4)
   26  FORMAT(4I5,I7)
       RETURN $ END
```

```
      PROGRAM PROGRAM(DATA=512,DUMMY=200,INPUT=100,
     * OUTPUT=200,PRNT=200,TAPE1=DATA,
     * TAPE4=DUMMY,TAPE5=INPUT,TAPE6=OUTPUT,TAPE7=PRNT)

C DATA   CONTAINS FILES OF MINTERMS AND DONTCARES
C DUMMY  IS A DUMMY OUTPUT FILE, OFTEN REWOUND, NEVER PRINTED
C INPUT  CONTAINS THE INPUT RECORD FOR CONTROL
C OUTPUT HAS THE IMPORTANT OUTPUT INFORMATION
C PRNT   CONTAINS OFF-LINE TYPE OUTPUT

      IMPLICIT INTEGER (A-Z)
      COMMON MINTERM,COM(2041)
      COMMON/STATS/NV,NO,LASTRAN,SEED, NOMIN,D,NPIMP,NEPI,
     * TERMINL,NONODES,THYME,TOTALG,TOTALL,TOTALT,
     * PFL,COSTMAX,GCOEF,LCOEF,PCOEF,TCOEF,PVALUES(10),SEL
      COMPLEX FIELD

      CALL CONNEC(5) $ CALL CONNEC(6) $ REWIND 4
      WRITE(6,1)FIELD(PFL,COM)
    1 FORMAT(* ADDRESS OF COM(0): *05
     * *B, CURRENT FIELD LENGTH: *06*B*)
      PFL=PFL-1
      CALL CONTROL("START")
      END
```

```
      SUBROUTINE QMTONET
      IMPLICIT INTEGER(A-Z)
      COMMON/OPUNIT/U
      LOGICAL PCOVERM
      COMMON MINTERM(1),COM(2041)
      COMMON/ZBASE/ZBASE
      COMMON/STATS/NV,NO,LASTRAN,SEED, NOMIN,D,NPIMP,NEPI,
     " TERMINL,NONODES,THYME,TOTALG,TOTALL,TOTALT,
     " PFL,COSTMAX,GCOEF,LCOEF,PCOEF,TCOEF,PVALUES(10),SEL
      COMMON/POINERS/DEXAL,FINAL,IPOINT,NODE,NW,PIT,POINTER,S
     " ,SEXAL,START,THIS
      COMMON/DATA/H,NO101,ZERO,NO120
      MWEIGHT(I)=MINTERM(I).A..N.MASK(51)
      CLR(I)=COM(I).A.NO120
      PCOVERM(P,M)=(SHIFT(XOR(CLR(NOMIN+P),CLR(M-1)),1).A..N.
     " XOR(CLR(NOMIN+P),CLR(M-1)).A.H).EQ.0

      WRITE(U,11)TCOEF,LCOEF,GCOEF,PCOEF,FLOAT(COSTMAX)

C PRINT CURRENT COST COEFFICIENTS
   11 FORMAT(* COST COEFFICIENTS:*4I6,1PG12.3)
      IF(PCOEF.NE.0)WRITE(U,1)PVALUES
    1 FORMAT(* PACKAGE COEFFICIENTS:*10I4)

C INITIALISE POINTERS AND SET FLAG
      NW=(NOMIN+59)/60 $ NEPI=CSTFLAG=0
      PIT=POINTER=NOMIN+NPIMP

C SCAN MINTERMS TO COMPUTE WEIGHT AND SET EPI FLAGS
C RECORD ALSO FIRST AND LAST COVERING PI.S
      DO 3 M=1,NOMIN
      MINTERM(M)=MINTERM(M).A.MASK(21)
      DO 2 P=1,NPIMP
      IF(PCOVERM(P,M))4,2
    4 IF(MWEIGHT(M).EQ.0)FIRSTPI=P
      MINTERM(M)=MINTERM(M)+1 $ LASTPI=P
    2 CONTINUE
      IF(FIRSTPI.EQ.LASTPI)
     " COM(NOMIN+LASTPI)=COM(NOMIN+LASTPI).OR.MASK(1)
    3 MINTERM(M)=
     " MINTERM(M).OR.SHIFT(FIRSTPI,24).OR.SHIFT(LASTPI,9)

C SORT MINTERMS INTO ASCENDING WEIGHT
      IF(NOMIN.EQ.1)GO TO 15
      DO 5 II=2,NOMIN $ I=II-1
      DO 5 J=II,NOMIN
      IF(MWEIGHT(I).LE.MWEIGHT(J))GO TO 5
      TEMP=MINTERM(I) $ MINTERM(I)=MINTERM(J)
      MINTERM(J)=TEMP
    5 CONTINUE

C EXAMINE EACH P.I. IN TURN
   15 DO 6 P=1,NPIMP
      BITCNT=PWT=0
```

```
C SCAN MINTERMS TO CONSTRUCT NEXT ROW OF P.I.TABLE
      DO 7 M=1,NOMIN
      IF(BITCNT.NE.0)GO TO 8 $ BITCNT=59
      POINTER=POINTER+1 $ COM(POINTER)=0 $ GO TO 9
    8 BITCNT=BITCNT-1
    9 IF(PCOVERM(P,M))10,7
   10 PWT=PWT+1
      COM(POINTER)=COM(POINTER).OR.SHIFT(1,BITCNT)
    7 CONTINUE

C INCREMENT NEPI AS NECESSARY
      TEMP=COM(NOMIN+P).A.MASK(21-NO-NO)
      IF(TEMP.LT.0)NEPI=NEPI+1

C COMPUTE AND ENTER COST
      NOLITS=COUNT(H.A..N.TEMP)+NV-NO-30
      TEMPA=TCOEF+LCOEF*NOLITS+PCOEF*PVALUES(NOLITS)
      IF(NOLITS.GT.1)TEMPA=TEMPA+GCOEF
      TEMPB=TEMPA.A..N.MASK(27)
      IF(TEMPA.NE.TEMPB)CSTFLAG=CSTFLAG.OR.MASK(1)
      IF(TEMPB.EQ.0)CSTFLAG=CSTFLAG.OR.1
      COM(NOMIN+P)=
     * COM(NOMIN+P).A.MASK(21).A..N.MASK(1).OR.TEMPB
    6 CONTINUE $ ZBASE=POINTER $ START=POINTER+NOMIN
      COM(ZBASE+1)=0
      DO 13 I=2,NOMIN
   13 COM(ZBASE+I)=NO101

C PRINT MESSAGE IF COST EXCEEDS 33 BITS
      IF(CSTFLAG.LT.0)WRITE(6,12)
   12 FORMAT(* --- WARNING: PI COSTS NEGATIVE OR TRUNCATED*)
      IF(SHIFT(CSTFLAG,59).LT.0)WRITE(6,14)
   14 FORMAT(* --- WARNING: ZERO PI COST MAY CAUSE ERRORS*)
      RETURN $ END
```

```
      SUBROUTINE RANDOM
      IMPLICIT INTEGER(A-Z)
      COMMON MINTERM(2041),DONTCAR(1)
      COMMON/STATS/NV,NO,LASTRAN,SEED, NOMIN,D,NPIMP,NEPI,
     * TERMINL,NONODES,THYME,TOTALG,TOTALL,TOTALT,
     * PFL,COSTMAX,GCOEF,LCOEF,PCOEF,TCOEF,PVALUES(10),SEL
      COMMON/POINERS/DEXAL,FINAL,IPOINT,NODE,NW,PIT,POINTER,S
     * ,SEXAL,START,THIS
      COMMON/NODE/COVL,COST,FRED,Z, NEWCOVL,NEWCOST,NEWN,NEWZ
     * ,PINO,SUCLIST(256),NEWUSL(256),NUSED,NSUCCS,NEWCOV(17)
      COMMON/DATA/H,NO101,ZERO,NO120
      COMMON/OPUNIT/U
      REAL RANF,TRCOST

      CALL TIME(TEMPA) $ CALL SECOND(TEMPB)
      SEED=-(TEMPA+TEMPB) $ GO TO 9

      ENTRY REPEAT

C REDIRECT THE 'REPEAT' CALL ACCORDING TO THE LASTRAN FLAG
      GO TO (1,9,7,1) LASTRAN
    1 CALL RECREAT $ RETURN
    7 CALL REPEATY $ RETURN

    9 CALL RANSET(SEED)
      CALL RANGET(SEED)
    2 FORMAT(* RANDOM NUMBER SEED: *O20)

C ONE FOR GOOD MEASURE
      TRCOST=RANF(TRCOST)
      A=SHIFT(1,NV)-1
      TEMP=J=NOMIN=0 $ K=29

C GET NEXT RANDOM NUMBER
    3 TEMPA=RANF(TRCOST).A..N.MASK(30)
      TEMPB=
     * .N.TEMPA.A.RANF(TRCOST).A.RANF(TRCOST).A..N.MASK(30)

C INTERPRET LOWER 30 BITS AS NEXT 30 TERMS
      DO 4 I=TEMP,K
      IF((TEMPA.A.1).EQ.0)GO TO 5
      NOMIN=NOMIN+1 $ MINTERM(NOMIN)=I $ GO TO 6

C DONTCARES ARE FORMED FROM BITS WHICH WERE ZERO
C AND ARE ONE IN TWO MORE RANDOM NUMBERS
    5 IF((TEMPB.A.1).EQ.0)GO TO 6
      J=J+1 $ DONTCAR(J)=I
    6 TEMPA=SHIFT(TEMPA,-1) $ TEMPB=SHIFT(TEMPB,-1)
      IF(I.EQ.A)GO TO 8
    4 CONTINUE
      TEMP=TEMP+30 $ K=K+30 $ GO TO 3
    8 D=J $ CALL HEADING(0) $ WRITE(U,2)SEED
      NO=0
      RETURN $ END
```

```
      SUBROUTINE RANDY
      IMPLICIT INTEGER(A-Z)
      COMMON MINTERM(2041),DONTCAR(1)
      COMMON/STATS/NV,NO,LASTRAN,SEED, NOMIN,D,NPIMP,NEPI,
     * TERMINL,NONODES,THYME,TOTALG,TOTALL,TOTALT,
     * PFL,COSTMAX,GCOEF,LCOEF,PCOEF,TCOEF,PVALUES(10),SEL
      COMMON/POINERS/DEXAL,FINAL,IPOINT,NODE,NW,PIT,POINTER,S
     * ,SEXAL,START,THIS
      COMMON/NODE/COVL,COST,PRED,Z, NEWCOVL,NEWCOST,NEWN,NEWZ
     * ,PINO,SUCLIST(256),NEWUSL(256),NUSED,NSUCCS,NEWCOV(17)
      COMMON/DATA/H,NO101,ZERO,NO120
      COMMON/OPUNIT/U
      REAL RANF,RCOST,THRESH1,THRESH2,TRCOST

      CALL TIME(TEMPA) $ CALL SECOND(TEMPB)
      SEED=-(TEMPA+TEMPB)

      ENTRY REPEATY
      CALL RANSET(SEED)
      CALL RANGET(SEED)
    2 FORMAT(* RANDY NUMBER SEED: *020)
      TRCOST=RANF(RANF(TRCOST))
      TEMP=SHIFT(1,NV) $ A=TEMP-1

C NUMBERS OF MINTERMS AND DONTCARES SELECTED AT RANDOM
      NOMIN=RANF(TRCOST)*A+1
      J=RANF(TRCOST)*(TEMP-NOMIN)
      IP=K=D=0
      DO 3 I=IP,A
      RCOST=TEMP-I

C SET THRESHOLDS ACCORDING TO NUMBERS SELECTED SO FAR
      THRESH1=(NOMIN-K)/RCOST $ THRESH2=(J-D)/RCOST+THRESH1
      TRCOST=RANF(TRCOST)

C COMPARE NEW RANDOM NUMBER WITH THRESHOLDS
C TO DETERMINE STATUS OF NEXT TERM
      IF(TRCOST.GE.THRESH1)GO TO 4
      K=K+1 $ MINTERM(K)=I $ GO TO 3
    4 IF(TRCOST.GE.THRESH2)GO TO 3
      D=D+1 $ DONTCAR(D)=I
    3 CONTINUE
      CALL HEADING(0) $ WRITE(U,2)SEED
      NO=0
      RETURN $ END
```

```
      SUBROUTINE READATA
      IMPLICIT INTEGER(A-Z)
      COMMON MINTERM(2041),DONTCAR(1)
      COMMON/STATS/NV,NO,LASTRAN,SEED, NOMIN,D,NPIMP,NEPI,
     * TERMINL,NONODES,THYME,TOTALG,TOTALL,TOTALT,
     * PFL,COSTMAX,GCOEF,LCOEF,PCOEF,TCOEF,PVALUES(10),SEL
      COMMON/POINERS/DEXAL,FINAL,IPOINT,NODE,NW,PIT,POINTER,S
     * ,SEXAL,START,THIS
      COMMON/IPUNIT/UI,RSFLAG
      COMMON/INPDATA/MINTS(18),DONS(18),NVAR,NOP
      COMMON/NODE/COVL,COST,FRED,Z, NEWCOVL,NEWCOST,NEWN,NEWZ
     * ,PINO,SUCLIST(256),NEWUSL(256),NUSED,NSUCCS,NEWCOV(17)
      COMMON/DATA/H,NO101,ZERO,NO120
      COMMON/OPUNIT/U
      LOGICAL FIXEDR

      SEED=0

C PRINT INSTRUCTIONS
      IF(UI.EQ.5)WRITE(U,13)
   13 FORMAT(* ENTER DATA IN BINARY(0,1,-) OR OCTAL OR *
     * *DECIMAL(DEFAULT),*/* WITH ANY DELIMITERS INCLUDING *
     * *E-O-L (80 CHARS). NUMBER TYPES RESET*/* WITH *
     * *EMBEDDED B,O,D RESPECTIVELY*/* MINTERMS TERMINATED *
     * *BY SLASH, DATA TERMINATED BY PERIOD OR E-O-R*/)
      NO=NOMIN=D=0 $ RADIX=1 $ FIXEDR=.F.

C READ IN A PROBLEM
    1 CALL READIN(RADIX,FIXEDR)
    2 CALL SORTHEM(0)
   50 CALL HEADING(0) $ IF(LASTRAN.EQ.4)WRITE(U,12)NO
   12 FORMAT(* NUMBER OF OUTPUTS:*I5)
      RETURN
```

```
      ENTRY MREAD
      IF(NO.GE.1.A.NO.LT.NV)GO TO 3 $ WRITE(6,4) $ RETURN
    4 FORMAT(* --- ERROR: NO IS OUT OF RANGE*)

C REQUEST DATA FORMAT
    3 IF(UI.EQ.5)WRITE(U,5) $ READ(UI,6)TEMP
      IF(EOF(UI).NE.0)GO TO 25
    5 FORMAT(* INPUT DATA FORMAT: INDIVIDUAL OR COMBINED? *)
    6 FORMAT(A1)
      IF(TEMP.EQ.1HI)GO TO 7 $ IF(TEMP.NE.1HC)GO TO 3

C COMBINED: ISSUE INSTRUCTIONS AND READ THE DATA
      IF(UI.EQ.5)WRITE(U,8)
    8 FORMAT(* ENTER DATA IN BINARY STRINGS OF (0,1,-) :-*/)
      NOMIN=D=0 $ RADIX=0 $ FIXEDR=.T. $ GO TO 1

C INDIVIDUAL: ISSUE INSTRUCTIONS
    7 IF(UI.EQ.5)WRITE(U,9)
    9 FORMAT(* ENTER MINTERMS AND DONTCARES FOR EACH OUTPUT *
     " *SEPARATELY:-*/)
      NOMIN=D=0 $ RADIX=1 $ FIXEDR=.F.
      DO 10 I=1,NO
      ON1=NOMIN+1 $ OD1=D+1

C READ DATA FOR EACH OUTPUT VARIABLE INDIVIDUALLY
      CALL READIN(RADIX,FIXEDR)

C REFORMAT TERMS EACH TIME
      DO 11 J=ON1,NOMIN
   11 MINTERM(J)=SHIFT(MINTERM(J),NO).OR.SHIFT(1,NO-I)
      DO 10 J=OD1,D
   10 DONTCAR(J)=SHIFT(DONTCAR(J),NO).OR.SHIFT(1,NO-I)
      GO TO 2
   25 WRITE(6,44) $ RETURN
   44 FORMAT(* --- WARNING: INPUT UNUSABLE*)

      ENTRY RECREAT

C RECREATE A PROBLEM (SINGLE OR MULTIPLE OUTPUT)
C FROM INFORMATION IN /INPDATA/
      NOMIN=D=0 $ NV=NVAR $ NO=NOP
      DO 51 I=ZERO,2000B
      J=I/60+1 $ TEMP=SHIFT(MASK(1),60*J-I)
      IF((MINTS(J).A.TEMP).EQ.0)GO TO 52
      NOMIN=NOMIN+1 $ MINTERM(NOMIN)=I
   52 IF((DONS(J).A.TEMP).EQ.0)GO TO 51
      D=D+1 $ DONTCAR(D)=I
   51 CONTINUE
      GO TO 50
      END
```

```
      SUBROUTINE RSWAP

C TO READ SOLUTION, NEGATE EVERY LITERAL AND
C INTERPRET AS PRODUCT OF SUMS
      IMPLICIT INTEGER(A-Z)
      COMMON MINTERM(1),COM(2040),DONTCAR(1)
      COMMON/STATS/NV,NO,LASTRAN,SEED, NOMIN,D,NPIMP,NEPI,
     " TERMINL,NONODES,THYME,TOTALG,TOTALL,TOTALT,
     " PFL,COSTMAX,GCOEF,LCOEF,PCOEF,TCOEF,PVALUES(10),SEL
      COMMON/POINERS/DEXAL,FINAL,IPOINT,NODE,NW,PIT,POINTER,S
     " ,SEXAL,START,THIS
      COMMON/NODE/COVL,COST,PRED,Z, NEWCOVL,NEWCOST,NEWN,NEWZ
     " ,PINO,SUCLIST(256),NEWUSL(256),NUSED,NSUCCS,NEWCOV(17)
      COMMON/DATA/H,NO101,ZERO,NO120
      COMMON/OPUNIT/U

      B=SHIFT(1,NV)-1
      MINTERM(NOMIN+1)=DONTCAR(D+1)=2000
      POINTER=NOMIN $ K=N=1 $ IP=0

C NEW MINTERMS ARE NOT DONTCARES OR OLD MINTERMS
      DO 1 I=IP,B
      IF(I.NE.MINTERM(K))GO TO 2 $ K=K+1 $ GO TO 1
    2 IF(I.NE.DONTCAR(N))GO TO 5 $ N=N+1 $ GO TO 1
    5 POINTER=POINTER+1 $ COM(POINTER)=I
    1 CONTINUE
      TEMP=NOMIN
      NOMIN=POINTER-NOMIN
      POINTER=TEMP

C OVERWRITE OLD MINTERM LIST WITH NEW (DONTCARES UNTOUCHED)
      DO 7 I=1,NOMIN
      POINTER=POINTER+1
    7 MINTERM(I)=COM(POINTER)
      WRITE(U,8)NOMIN
    8 FORMAT(* NUMBER OF MINTERMS AFTER SWAPPING:*I5)
      RETURN $ END
```

```
      SUBROUTINE RTOQM
      IMPLICIT INTEGER(A-Z)
      COMMON MINTERM(1),COM(2040),DONTCAR(1)
      COMMON/STATS/NV,NO,LASTRAN,SEED, NOMIN,D,NPIMP,NEPI,
     * TERMINL,NONODES,THYME,TOTALG,TOTALL,TOTALT,
     * PFL,COSTMAX,GCOEF,LCOEF,PCOEF,TCOEF,PVALUES(10),SEL
      COMMON/POINERS/DEXAL,FINAL,IPOINT,NODE,NW,PIT,POINTER,S
     * ,SEXAL,START,THIS
      COMMON/NODE/COVL,COST,PRED,Z, NEWCOVL,NEWCOST,NEWN,NEWZ
     * ,PINO,SUCLIST(256),NEWUSL(256),NUSED,NSUCCS,NEWCOV(17)
      COMMON/DATA/H,NO101,ZERO,NO120

      TEMP=NOMIN

C DELIMIT LISTS WITH SPECIAL MARKER
      MINTERM(NOMIN+1)=DONTCAR(D+1)=200 01777B
      J=K=1 $ TEMPA=QMXPAND(MINTERM) $ TEMPB=QMXPAND(DONTCAR)
   15 IF(TEMPA-TEMPB)12,13,14

C REFORMAT MINTERMS AND ALSO ADD TO LIST OF TERMS
   12 TEMP=TEMP+1 $ COM(TEMP)=MINTERM(J)=TEMPA $ J=J+1
      TEMPA=QMXPAND(MINTERM(J)) $ GO TO 15

C ADD DONTCARES (REFORMATTED) TO LIST OF TERMS
   14 TEMP=TEMP+1
      COM(TEMP)=TEMPB $ K=K+1 $ TEMPB=QMXPAND(DONTCAR(K))
      GO TO 15

C DEMARK MINTERM LIST AND SET NPIMP TO NUMBER OF TERMS
C (IN CASE CONSENS IS USED)
   13 COM(NOMIN)=NO101
      NPIMP=NOMIN+D
      RETURN $ END
```

```
      LOGICAL FUNCTION SELECT(SIZE)
      IMPLICIT INTEGER(A-Z)
      COMMON MINTERM,COM(2041)
      COMMON/ZBASE/ZBASE
      REAL CRITVAL,NORMS,TREMP,SEL,SIZE
      COMMON/STATS/NV,NO,LASTRAN,SEED, NOMIN,D,NPIMP,NEPI,
     " TERMINL,NONODES,THYME,TOTALG,TOTALL,TOTALT,
     " PFL,COSTMAX,GCOEF,LCOEF,PCOEF,TCOEF,PVALUES(10),SEL
      COMMON/POINERS/DEXAL,FINAL,IPOINT,NODE,NW,PIT,POINTER,S
     " ,SEXAL,START,THIS
      EQUIVALENCE (CRITVAL,CRIT)
      SNM(I,J,K)=SHIFT(COM(I),J).A..N.MASK(K)
      EXTYPE(I)=SNM(I,3,57)
      EXN(I)=SNM(I,18,50)
      EXZ(I)=SNM(I,28,50)
      EXCOST(I)=SNM(I+1,33,27)
      ZSCORE(I)=COM(ZBASE+EXZ(I))

      CALL PAUSE("SELECT")

C ENSURE INDEX HAS AT LEAST ONE TYPE 0 NODE AT THE BOTTOM
      IF(SEXAL.NE.DEXAL)GO TO 6
      IF(EXTYPE(DEXAL-2).EQ.0)GO TO 6
      CALL SHORTY(2,0) $ S=S-2 $ COM(DEXAL-2)=0
    6 NORMS=S/SIZE $ I=PFL-1
      CRIT=0 $ IF(NORMS.GT.SEL)GO TO 2
      NVAL=.N.MASK(1)
    3 I=I-2
      IF(EXTYPE(I)-1)4,1,3

C EXAMINE TYPE 1 NODES ONLY
    1 C=EXCOST(I) $ TEMP=C-ZSCORE(I)
      IF(NVAL.LT.TEMP)GO TO 3
      TREMP=EXN(I)/FLOAT(C)
      IF(NVAL.GT.TEMP)GO TO 5
      IF(TREMP.LE.CRITVAL)GO TO 3

C CHOOSE ONE WITH LOWEST COST-ZSCORE (MAXIMUM N/C IF CHOICE)
    5 CRITVAL=TREMP $ THIS=I $ NVAL=TEMP
      GO TO 3

C FOR HIGH S, MAXIMISE Z (MAXIMUM N IF CHOICE)
    2 I=I-2
      IF(EXTYPE(I)-1)4,10,2
   10 TEMP=SHIFT(EXZ(I),10).OR.EXN(I)
      IF(TEMP.LE.CRIT)GO TO 2
      CRIT=TEMP $ THIS=I
      GO TO 2

    4 SELECT=CRIT.NE.0 $ IF(SELECT)CALL PAUSE("GOTONE")
      RETURN $ END
```

```
      SUBROUTINE SHORTY(SPACE,ANODE)
      IMPLICIT INTEGER(A-Z)
      COMMON MINTERM(1),COM(2040),DONTCAR(1)
      LOGICAL ROOMOK,UNSAVE
      COMMON/TEMPARY/TEMPRY(40)
      COMMON/STATS/NV,NO,LASTRAN,SEED, NOMIN,D,NPIMP,NEPI,
     * TERMINL,NONODES,THYME,TOTALG,TOTALL,TOTALT,
     " PFL,COSTMAX,GCOEF,LCOEF,PCOEF,TCOEF,PVALUES(10),SEL
      COMMON/POINERS/DEXAL,FINAL,IPOINT,NODE,NW,PIT,POINTER,S
     " ,SEXAL,START,THIS
      DATA M3308/-776000000B/
      NM(I,J)=COM(I).A..N.MASK(J)
      SNM(I,J,K)=SHIFT(COM(I),J).A..N.MASK(K)
      EXLOC(I)=NM(I,43)
      EXLEN(I)=SNM(I+1,41,52)
      EXLAST(I)=EXLOC(I)+EXLEN(I)-1
      ROOMOK(I)=POINTER+NEED.LT.I

C UPDATE THE WORDS-IN-USE COUNT
      S=S+SPACE
      NEED=SPACE $ IF(ANODE.NE.0)GO TO 7

C EXIT IF NO SHORTENING REQUIRED
      IF(ROOMOK(SEXAL))RETURN
   10 UNSAVE=.T. $ GO TO 8

C ANODE MUST FINISH AT THE TOP
    7 IF(EXLAST(ANODE).NE.POINTER)GO TO 9
      IF(ROOMOK(SEXAL))RETURN $ GO TO 10
    9 LEN=EXLEN(ANODE) $ NEED=LEN+NEED $ BASE=EXLOC(ANODE)-1
      IF(ROOMOK(SEXAL))GO TO 12

C ANODE IS TO BE RELOCATED; COPY IT TO TEMPORARY STORE
      IF(LEN.GT.40)STOP "TEMPARY STORAGE INADEQUATE"
      DO 14 I=1,LEN
   14 TEMPRY(I)=COM(BASE+I) $ UNSAVE=.F.
      COM(ANODE)=COM(ANODE).A.MASK(43)
      COM(ANODE+1)=COM(ANODE+1).A.M3308

C PROCEDE WITH SHORTENING
    8 OLDMIN=1 $ POINTER=START
      CALL PAUSE("SHORTY")
    5 MIN=PFL

C EXTRACT NEXT LOWEST STORED NODE
      DO 1 I=SEXAL,PFL,2
      TEMPA=EXLOC(I)
      IF(TEMPA.LE.OLDMIN)GO TO 1
      IF(MIN.LT.TEMPA)GO TO 1
      MIN=TEMPA $ TEMP=I
    1 CONTINUE
      IF(MIN.EQ.PFL)GO TO 11
      OLDMIN=MIN $ LENNY=EXLEN(TEMP)
      MIN=MIN-1 $ IF(MIN.EQ.POINTER)GO TO 13

C COPY IT TO BOTTOM OF AVAILABLE STORE
      DO 4 I=1,LENNY
    4 COM(POINTER+I)=COM(MIN+I)
      COM(TEMP)=COM(TEMP).A.MASK(43).OR.POINTER+1
   13 POINTER=POINTER+LENNY $ GO TO 5
```

```
C ENSURE THAT SPACE IS NOW ADEQUATE
   11   IF(ROOMOK(SEXAL))GO TO 16 $ TOP=DEXAL-2
        SAVING=0
        CALL PAUSE("BAILOUT")

C STORAGE CRITICAL: TRY FOR SPACE BY DELETING STACK ELEMENTS
   22   IF(SEXAL.EQ.DEXAL)STOP "STORAGE INADEQUATE"
        MAX=0

C SELECT THE ONE USING MOST LOCATIONS (PROBABLY OLD ANYWAY)
        DO 6 I=SEXAL,TOP,2
        TEMPA=EXLEN(I)
        IF(MAX.GE.TEMPA)GO TO 6
        MAX=TEMPA $ TEMP=I
    6   CONTINUE

C DELETE ITS INDEX ENTRY
        OLDMIN=MINO(OLDMIN,EXLOC(TEMP))
        SAVING=SAVING+EXLEN(TEMP)
        COM(TEMP)=COM(SEXAL) $ S=S-2
        COM(TEMP+1)=COM(SEXAL+1) $ SEXAL=SEXAL+2

C SEE IF THAT WILL DO
        IF(ROOMOK(SEXAL+SAVING))17,22
   17   POINTER=OLDMIN-1 $ S=S-SAVING $ GO TO 5

C SPACE IS (NOW) ADEQUATE: RESET POINTERS AS REQUIRED
   16   IF(UNSAVE)GO TO 18
        DO 20 I=1,LEN
   20   COM(POINTER+I)=TEMPRY(I)
        COM(ANODE+1)=COM(ANODE+1).OR.SHIFT(LEN,19) $ GO TO 21
   12   DO 15 I=1,LEN
   15   COM(POINTER+I)=COM(BASE+I)
   21   COM(ANODE)=COM(ANODE).A.MASK(43).OR.POINTER+1
        POINTER=POINTER+LEN
   18   NODE=EXLOC(THIS) $ RETURN

        ENTRY SHORTM
        BIGEST=-MASK(60-NV) $ IF(NOMIN-1)25,87,88
   25   WRITE(6,44) $ RETURN
   44   FORMAT(* ---- WARNING: INPUT UNUSABLE*)

C SORT MINTERMS INTO ASCENDING ORDER
   88   DO 45 KK=2,NOMIN
        K=KK-1 $ TEMP=MINTERM(K)
        DO 86 I=KK,NOMIN
        IF(MINTERM(I)-TEMP)85,35,86

C ELIMINATE ANY DUPLICATIONS
   35   MINTERM(I)=MINTERM(NOMIN) $ CALL REPLACE(I,I-1)
        CALL REPLACE(NOMIN,NOMIN-1) $ GO TO 86
   85   TEMP=MINTERM(I) $ MINTERM(I)=MINTERM(K)
        MINTERM(K)=TEMP
   86   CONTINUE
        IF(TEMP.LE.BIGEST)GO TO 45 $ CALL REPLACE(NOMIN,K-1)
        GO TO 46
   45   CONTINUE
   87   IF(MINTERM(NOMIN).GT.BIGEST)NOMIN=NOMIN-1
   46   IF(NOMIN.LE.0)GO TO 25 $ MINTERM(NOMIN+1)=BIGEST+1
        RETURN
```

```
      ENTRY SHORTD
      J=D $ D=0 $ MPOIN=1 $ IF(J-1)30,31,81

C SORT DONTCARES INTO ASCENDING ORDER
   81 DO 33 KK=2,J
      K=KK-1 $ TEMP=DONTCAR(K)
      DO 34 I=KK,J
      IF(DONTCAR(I)-TEMP)36,37,34

C ELIMINATE DUPLICATIONS
   37 DONTCAR(I)=DONTCAR(J) $ CALL REPLACE(J,J-1)
      CALL REPLACE(I,I-1) $ GO TO 34
   36 TEMP=DONTCAR(I) $ DONTCAR(I)=DONTCAR(K)
      DONTCAR(K)=TEMP
   34 CONTINUE

C TERMS IN MINTERM LIST CANNOT ALSO BE DONTCARES
   38 IF(MINTERM(MPOIN)-TEMP)40,33,39
   40 IF(MPOIN.GT.NOMIN)GO TO 30 $ MPOIN=MPOIN+1 $ GO TO 38
   39 D=D+1 $ DONTCAR(D)=TEMP
   33 CONTINUE
   31 IF(MINTERM(MPOIN)-DONTCAR(J))41,30,42
   41 IF(MPOIN.GT.NOMIN)GO TO 30 $ MPOIN=MPOIN+1 $ GO TO 31
   42 D=D+1 $ DONTCAR(D)=DONTCAR(J)
   30 RETURN $ END
```

```
$SOLUTION
      SUBROUTINE SOLUTIO
      IMPLICIT INTEGER(A-Z)
      COMMON MINTERM(1),COM(2041)
      COMMON/STATS/NV,NO,LASTRAN,SEED, NOMIN,D,NPIMP,NEPI,
     " TERMINL,NONODES,THYME,TOTALG,TOTALL,TOTALT,
     " PFL,COSTMAX,GCOEF,LCOEF,PCOEF,TCOEF,PVALUES(10),SEL
      COMMON/POINERS/DEXAL,FINAL,IPOINT,NODE,NW,PIT,POINTER,S
     " ,SEXAL,START,THIS
      COMMON/NODE/COVL,COST,PRED,Z, NEWCOVL,NEWCOST,NEWN,NEWZ
     " ,PINO,SUCLIST(256),NEWUSL(256),NUSED,NSUCCS,NEWCOV(17)
      COMMON/DATA/H,NO101,ZERO,NO120
      SNM(I,J,K)=SHIFT(COM(I),J).A..N.MASK(K)
      EXPRED(I)=PFL-SNM(I,43,45)
      EXPINO(I)=SNM(I+1,54,47)

      IF(FINAL.NE.0)GO TO 3 $ WRITE(6,4) $ RETURN
    4 FORMAT(* --- ERROR: NO TERMINAL NODE (CURRENTLY)*)
    3 I=FINAL
      TOTALT=TOTALL=TOTALG=TERM=0

C STEP BACK THROUGH TREE, EXTRACTING PI.S ON SOLUTION PATH
   79 PINO=EXPINO(I)

C ROOT NODE HAS PI.NO. ZERO
      IF(PINO.EQ.0)GO TO 83
      TEMP=COM(NOMIN+PINO).A.MASK(21-NO-NO)
      IF(LASTRAN.NE.4)GO TO 1 $ IF(TOTALT.EQ.0)GO TO 1

C IN THE MULTILPE OUTPUT CASE,
C IGNORE SUBSEQUENT OCCURENCES OF THE ONE PRIMARY GATE
      DO 2 J=1,TOTALT
      IF((SUCLIST(J).A.MASK(21-NO-NO)).NE.TEMP)GO TO 2
      SUCLIST(J)=SUCLIST(J).OR.COM(NOMIN+PINO).A.MASK(21)
      I=EXPRED(I) $ GO TO 79
    2 CONTINUE

C COMPUTE COSTS
    1 NOLITS=COUNT(H.A..N.TEMP)+NV-NO-30
      TOTALT=TOTALT+1 $ TOTALL=TOTALL+NOLITS
      IF(NOLITS.GT.1)TOTALG=TOTALG+1
      TERM=TERM+TCOEF+LCOEF*NOLITS+PCOEF*PVALUES(NOLITS)
      IF(NOLITS.GT.1)TERM=TERM+GCOEF

C SOLUTION STORED IN SUCLIST, 1 THRU TOTALT
      SUCLIST(TOTALT)=COM(NOMIN+PINO).A.MASK(21)
      I=EXPRED(I) $ GO TO 79
   83 IF(TOTALT.EQ.0)STOP "SOLUTION SET EMPTY"

C SOLUTION COST STORED IN SUCLIST TOO
C (NOT ALWAYS EQUAL TO TERMINL)
      SUCLIST(TOTALT+1)=TERM
      RETURN $ END
```

```
          IDENT   CMSTORE
          COMMENT COMPASS SUPPORT ROUTINES
          ENTRY   CMSTORE,DISCOV,DISPLAY,DUMPC,FIELD
          ENTRY   FREEFMT,PROCESS,QMXPAND,REPLACE

**********************************************************************

 CMSTORE  SX6     A0              X6=FIELD LENGTH
* LINK PROGRAM TO SAVE FIELD LENGTH BEFORE FTN INTERFERES
          SA6     FL              STORE IN FL
          EQ      =XPROGRAM       JUMP TO PROGRAM
 FL       BSS     1               SPACE TO SAVE FIELD LENGTH

**********************************************************************

 DISCOV   BSS     1               DISPLAY COVER
* PARAMETERS: INPUT ARRAY, RESULT ARRAY, NO. OF BITS
* EXPANDS ONE ROW OF PI TABLE INTO CHARS SUITABLE FOR PRINTING
          SA2     A1+1            X2=ADDR(RESULT(1))
          SA3     A2+1            X3=ADDR(NOMIN)
          SA3     X3
          SB3     X3              B3=NOMIN
          SA1     X1              X1=INPUT(1)
          SX4     1RX
          SX5     1R
          SX0     1R.
          SB1     6               B1=6 TIMES COUNTER
 RESET    SB2     9               B2=9 TIMES COUNTER
          SX6     0               CLEAR RESULT
 DCBIT    PL      X1,DZERO        TEST NEXT BIT
          BX6     X6+X4           =1, OR IN X
          EQ      TEST9
 DZERO    BX6     X6+X5           =0, OR IN BLANK
 TEST9    LX6     6               SHIFT RESULT ONE CHARACTER LEFT
          LX1     1               GET NEXT BIT
          SB2     B2-1
          NZ      B2,DCBIT        JUMP IF NOT YET 9 TIMES
          PL      X1,CZERO        TEST 10TH BIT
          BX6     X6+X4           =1,OR IN X
          EQ      DSTORE
 CZERO    BX6     X6+X0           =0, OR IN PERIOD
 DSTORE   LX1     1               GET NEXT BIT
          SA6     X2              STORE RESULT
          SB3     B3-10
          LE      B3,DISCOV       EXIT IF ALL DONE
          SX2     X2+1            INCREMENT RESULT ADDRESS
          SB1     B1-1
          NZ      B1,RESET        JUMP IF NOT YET 6 TIMES 10 BITS
          SA1     A1+1            X1=NEXT INPUT WORD
          SB1     6               B1=6 TIMES COUNTER
          EQ      RESET

**********************************************************************
```

```
 DISPLAY  BSS    1
* TRANSLATES TERMS INTO 10 CHARACTERS FOR PRINTING
          SB7    10D             B7=10 TIMES COUNTER
          SA3    X1              X3=PRIME IMPLICANT
          MX6    0               X6=ZEROS
          SX4    1R0
          SX1    1RI
          SX2    1R-
          SX5    1R*
 LOOP     LX3    1               MOVE IN BIT 1
          NG     X3,DASH         =1 IFF ITS A DASH
          LX3    1               MOVE IN BIT 2
          NG     X3,WUN          =1 IFF ITS A ONE
          BX6    X6+X4           MUST BE A ZERO
          EQ     CONT
 WUN      BX6    X6+X1
          EQ     CONT
 DASH     LX3    1               MOVE IN BIT 2
          NG     X3,ASTK         =1 IFF ITS AN ERROR
          BX6    X6+X2
          EQ     CONT
 ASTK     BX6    X6+X5           CHARACTER HAS BEEN OR-ED IN
 CONT     SB7    B7-1            DECREMENT COUNTER
          EQ     B7,DISPLAY      TEN TIMES YET-
          LX6    6               NO, SHIFT RESULT ONE CHAR LEFT
          EQ     LOOP

**********************************************************************

 DUMPC    BSS    1               LINK TO CORE DUMP ROUTINE
          SX6    1               SET ADDR(ARRAY)=1, SO THAT
          BX1    X6              ADDR(ARRAY(N)) = N, ABSOLUTE.
          SA6    A1              STORE IN ADDRESS TABLE ALSO
          RJ     =XDUMPF
          EQ     DUMPC

**********************************************************************

 FIELD    BSS    1
* RETURNS FIELD LENGTH AND ADDRESS OF BLANK COMMON
          SA2    A1+1            X2=ADDR(COM)
          SA3    FL              X3=FIELD LENGTH
          IX6    X3-X2           X6=FFL
          SA6    X1              STORE IT
          SX6    X2-1            X6=ADDR(COM(0))
          BX7    X3              X7=FIELD LENGTH
          EQ     FIELD

**********************************************************************
```

```
 FREEFMT  BSS    1              A1=ADDR(PARAMS),X1=ADDR(CHAR(1))
* FIRST PARAMETER IS A 30 CHARACTER STRING. IF CHARACTER ONE
* IS ALPHABETIC, FIRST ALPHANUMERIC STRING IS IGNORED. SECOND
* AND SUBSEQUENT PARAMETERS RECEIVE 20 CHARACTER BLOCKS OF:
* (A) NUMBERS (O,I,F OR E FORMAT), RIGHT JUSTIFIED,
* (B) ALPHANUMERIC STRINGS STARTING WITH A LETTER OR (C)
* CHARACTER STRINGS, LEFT JUSTIFIED, ALL BLANK FILLED.STRINGS
* DELIMITED BY ", EMBEDDED " REPRESENTED BY "". ANY SEPARATOR
* IS PERMISSIBLE INCLUDING NONE. UNSPECIFIED PARAMETERS AND
* NULL STRINGS("") ARE CLEARED TO ALL BLANK.
          SB6    1              B6=1=CONSTANT
          SA2    X1-1           A2=ADDR OF CHAR(0),X2=CHAR(0)
          SA1    A1+B6          X1=ADDR OF NEXT PARAM
          ZR     X1,FREEFMT     EXIT IF NO RESULT PARAMS NAMED
          MX0    54             X0=9 CHAR MASK
          SA5    =1H            X5=BLANK WORD
          RJ     KILL           CLEAR RESULT
          SB4    10             B4=10=CONSTANT
          SB3    11             B3=CURRENT STATUS=11
          SB5    3              B5=3=COUNTDOWN OF WORDS IN CHAR
          SB7    20             B7=20=CONSTANT
NXTWORD   SA2    A2+1           X2=NEXT WORD OF CHAR
          SB1    10             COUNTDOWN OF CHARS USED FROM X2
NXTCHAR   LX2    6              PUT NEXT CHAR AT RH END
          BX3    -X0*X2         X3=NEXT CHAR
          JP     B3+*
          EQ     FIRST          1  IN INITIAL A-N STRING
          EQ     SEPART         2  IN SEPARATOR STRING
          EQ     Q.WASNT        3  QUOTE WAS LAST AND WAS NOT IN
                                    DELIMITED STRING
          EQ     DELIM          4  IN A QUOTE DELIMITED STRING
          EQ     Q.WAS          5  QUOTE WAS LAST AND WAS IN
                                    DELIMITED STRING
          EQ     MANTIS         6  IN A NUMBER STRING
          EQ     HADE           7  AN E WAS LAST AND WAS IN
                                    A NUMBER STRING
          EQ     EXP            8  IN EXPONENT STRING
          EQ     ALPHNUM        9  IN AN ALPHANUMERIC STRING
          EQ     MANTIS         10 IN FRACTIONAL PART OF
                                    A NUMBER STRING
          ZR     X3,TWO         11 INITIAL STATE (ONCE ONLY)
          SX4    X3-1R0
          NG     X4,ONE         JUMP IF ALPHABETIC
          EQ     SEPART         ELSE START RECORDING
FIRST     ZR     X3,TWO         JUMP IF COLON
          SX4    X3-1R+
          NG     X4,TINUE       CONTINUE IF STILL ALPHANUMERIC
SEPART    ZR     X3,TWO         JUMP IF COLON
          SX4    X3-1R0
          NG     X4,NINE        JUMP IF ALPHABETIC
          SX4    X3-1R*
          NG     X4,SIX         JUMP IF DIGIT OR SIGN
          SX4    X3-1R.
          ZR     X4,TEN         JUMP IF A PERIOD
          SX4    X3-1R"
          ZR     X4,THREE       JUMP IF A QUOTE
          EQ     TWO            ELSE GO TO TWO
Q.WASNT   SX4    X3-1R"
          ZR     X4,FIVE        JUMP IF A QUOTE
          EQ     FOUR           ELSE GO TO FOUR
```

```
DELIM     SX4     X3-1R"
          ZR      X4,FIVE         JUMP IF A QUOTE
          EQ      RECORD          ELSE CONTINUE
Q.WAS     SX4     X3-1R"
          ZR      X4,FOUR         JUMP IF A QUOTE
          RJ      PROCES          PROCESS QUOTE DELIMITED STRING
          EQ      SEPART
MANTIS    SX4     X3-1RE
          ZR      X4,SEVEN        JUMP IF E
          SX4     X3-1R.
          ZR      X4,TEN          JUMP IF A PERIOD
          SX4     X3-1R0
          NG      X4,*+2          SKIP IF ALPHABETIC OR COLON
+         SX4     X3-1R+
          NG      X4,RECORD       CONTINUE IF DIGIT
          RJ      PROCES          PROCESS NUMBER
          EQ      SEPART
HADE      SX4     X3-1R0
          NG      X4,*+2          SKIP IF ALPHABETIC OR COLON
+         SX4     X3-1R*
          NG      X4,EIGHT        CONTINUE IF DIGIT
          RJ      KILL            ERROR IN EXPONENT FORMAT - KILL
          EQ      SEPART
EXP       SX4     X3-1R0
          NG      X4,*+2          SKIP IF ALPHABETIC OR COLON
+         SX4     X3-1R+
          NG      X4,RECORD       CONTINUE IF DIGIT
          RJ      PROCES          PROCESS REAL NUMBER
          EQ      SEPART
ALPHNUM   ZR      X3,*+2          SKIP IF COLON
+         SX4     X3-1R+
          NG      X4,RECORD       JUMP IF ALPHANUMERIC
          RJ      PROCES          PROCESS ALPHANUMERIC STRING
          EQ      SEPART
ONE       SB3     1
          EQ      TINUE
TWO       SB3     2
          EQ      TINUE
THREE     SB3     3
          EQ      TINUE
FOUR      SB3     4
          EQ      RECORD
FIVE      SB3     5
          EQ      TINUE
SIX       SB3     6
          SA4     0               A4=0=NUMBER
          EQ      RECORD
SEVEN     SB3     7
          EQ      RECORD
EIGHT     SB3     8
          EQ      RECORD
NINE      SB3     9
          EQ      RECORD
TEN       NE      B3,B4,*+1
-         RJ      KILL            TWO DECIMAL POINTS - KILL STRING
+         SA4     0               A4=0=NUMBER
          SB3     10
```

```
RECORD   EQ      B2,B7,TINUE   JUMP IF STRING ALREADY 20 CHARS
         LX6     6             SHIFT CHAR INTO RESULT
         LX7     6
         BX6     X0*X6
         BX4     -X0*X7
         BX6     X6+X4
         BX7     X0*X7
         BX7     X7+X3
         SB2     B2+B6         INCREMENT CHAR COUNT FOR RESULT
TINUE    SB1     B1-B6         DECREMENT CHAR COUNT FOR X2
         NZ      B1,NXTCHAR    JUMP IF NOT YET ALL USED
         SB5     B5-B6         DECREMENT WORD COUNT FOR CHAR
         NZ      B5,NXTWORD    JUMP IF NOT YET ALL USED
         SX4     B3-7
         ZR      X4,KILL       EXPONENT FORM UNFINISHED - KILL
         SX4     B3-4
         NG      X4,BLNKFLL    JUMP IF NO UNFINISHED PARAMETER
         RJ      PROCES        PROCESS AND STORE LAST PARAM
BLNKFLL  SA6     X1            SET REMAINING PARAMS TO BLANK
         SA7     A6+B6
         SA1     A1+1          X1=ADDR OF NEXT PARAM
         NZ      X1,BLNKFLL    JUMP IF STILL SOME PARAMS
         EQ      FREEFMT       EXIT
PROCES   BSS     1
         SX4     A4
         ZR      X4,NUMERAL    JUMP IF NUMERIC STRING
NORMISE  EQ      B2,B7,NUMERAL JUMP IF NORMALISED
         LX6     6             SHIFT LEFT 1 CHARACTER
         LX7     6
         BX4     -X0*X7
         BX6     X0*X6
         BX7     X0*X7
         BX6     X6+X4
         SX4     1R
         BX7     X7+X4
         SB2     B2+B6         INCREMENT NORMALISING COUNTER
         EQ      NORMISE
NUMERAL  SA6     X1            STORE RESULT
         SA7     A6+B6
         SA1     A1+1          X1=ADDR OF NEXT PARAM
         ZR      X1,FREEFMT    EXIT IF NO MORE PARAMS
         RJ      KILL          CLEAR RESULT REGISTERS
         EQ      PROCES
KILL     BSS     1
         SB2     B0            B2=0=NUMBER OF CHARS IN RESULT
         BX6     X5            SET RESULT WORDS BLANK
         BX7     X5
         SA4     B6            A4=1=ALPHANUMERIC STRING
         EQ      KILL
```

****

```
 PROCESS  BSS    1
* RETURNS A RELOCATED ADDRESS AND A SPECIAL CHARACTER FOR USE
* BY DUMP ROUTINE
          USE    /PROCESS/
 COMVAR   BSS    3              VARIABLES: AT,FLLESS1,LOCFMIN
          USE    *
          SA1    X1             X1=ADDRESS
          SA2    COMVAR         X2=AT
          MX0    59             X0=-1
          IX3    X2+X0          X3=AT-1
          ZR     X3,AT1         JUMP IF AT IS 1
          PL     X3,AT2         JUMP IF AT IS 2
          SA4    =1H
          BX6    X1             AT .LE.0, ADDRESS IS UNMODIFIED
          BX7    X4
          EQ     PROCESS
 AT1      SA5    COMVAR+2       X5=LOCFMIN
          SA4    =1H.
          IX6    X1-X5          SUBTRACT LOCFMIN FROM ADDRESS
          BX7    X4
          EQ     PROCESS
 AT2      SA5    COMVAR+1       X5=FLLESS1
          SA4    =1H*
          IX6    X5-X1          SUBTRACT ADDRESS FROM FLLESS1
          BX7    X4
          EQ     PROCESS

**********************************************************************

 QMXPAND  BSS    1
* EXPANDS TERMS INTO 2 BITS PER LITERAL FORMAT
          SA1    X1             X1=TERM
          SX6    0              CLEAR RESULT REGISTER
          LX1    48D            SHIFT VARIABLE 10 TO BIT 3
          SA2    MASK           X2=MOVING MASK AND COUNTER
 LOOPE    BX3    X1*X2          EXTRACT APPROPRIATE BIT
          BX6    X6+X3          INSERT INTO RESULT
          LX1    59D            RIGHT SHIFT X1 ONCE
          LX2    58D            RIGHT SHIFT MASK TWICE
          PL     X2,LOOPE       BIT 1 WILL BE 1 AFTER 10 TIMES
          EQ     QMXPAND
 MASK     DATA   1000000000002000000B

**********************************************************************

 REPLACE  BSS    1
* SETS P1=P2 (FOR CHANGING FORTRAN DO-LOOP PARAMETERS)
          SA2    A1+1
          SA2    X2
          BX6    X2
          SA6    X1
          EQ     REPLACE

          END    CMSTORE        TELL LOADER TO EXECUTE CMSTORE
```

Appendix B.

USERS HANDBOOK FOR BOOLEAN OPTIMISATION PROGRAM

1. Introduction

This program finds the optimum two-level combinational logic network for single functions of up to 10 variables. It may further be used for multiple output functions where the sum of input and output variables does not exceed 10. It has ancilliary uses such as listing a function, its complement, its prime implicants, its PI table, and finding quick minimal (not optimal) solutions. A problem may be specified by its individual minterms and dontcares, or in partially minimised form.

A basic knowledge of the INTERCOM operating system is assumed in the following instructions.

2. Operation

The source deck may be compiled under FTN (FORTRAN Extended) with default parameters, but a ten per cent reduction in execution time can be achieved by using the OPT parameter. The maximum field length to be used must be set before loading: EFL (60000) is normally adequate. The binary version is then loaded, and the program causes files INPUT and OUTPUT to be connected for terminal communication and enters the basic "CONTROL" mode. In this mode, the user may initiate a wide variety of tasks by entering different commands, details of which are described below.

If the "PAUSE" mode is selected, the algorithm pauses at various strategic points and returns control to the user. A short message is displayed to indicate where the break occurred. This facility is invaluable as a debugging aid, and is also used for the self-timing feature and for extracting approximate solutions.

3. Control Commands

The format of commands consists of a keyword followed by a parameter list: keywords may be truncated to the extent indicated by underlining, parameters may be either numbers, alphanumeric strings starting with a letter or quote delimited strings of arbitrary characters, and any unambiguous delimiters may be used between parameters.

All control commands are truncated to 30 characters.

The notation "p=" represents the parameters: p1 must be the first parameter, p2 the second, etc.

An address mode of R refers to the start of blank common, and M refers to the end of blank common.

"Display" type representation uses the three characters 0,1 and - to represent terms.

CHECK checks the solution obtained to ensure the set of terms constitutes an irredundant cover of prime implicants.

p1 = S suppresses printout if no errors are detected.

CONNECT connects a file to the terminal

p1 = tape number.

CONSENSUS compute the set of prime implicants using iterative consensus - generally less efficient than HWA, but may be used when a set of covering terms already exists.

p1 = S suppresses printout.

COSTS sets the coefficients in the cost function used to obtain the solution. The actual values are entered in response to instructions issued to the terminal. Package cost coefficients have preset values which may be changed only if the P coefficient is non-zero and the FLAG is non-blank.

p1 = S suppresses issuing of instructions.

DISABLE disables the pause mode (default condition).

DISCONNECT disconnects a file from the terminal.

p1 = tape number.

DUMP prints selected locations in store, used for debugging.

p = address (octal).

p = address 2 may define a range of addresses: these values are relocated according to

p = A absolute (default), p = R relative or

p = M minus.

The format of the printed contents is determined by

p = O octal (default), p = B binary.

p = F floating point.

The format of the printed location numbers corresponds to the address relocation parameter by default, or may be selected as

p = PA absolute, p = PR relative, p = PM minus.

ENABLE enables the pause mode of operation.

END terminates program execution

HEAVY uses a heaviest first technique to quickly extract a covering set of prime implicants. The "solution" obtained may contain redundant prime implicants. PIs are selected in order of descending weight or ascending cost, according to the parameters being

p1 = W, p2 = C or p1 = C, p2 = W respectively.

HWA computes the set of prime implicants of the function using a version of Hwa's method

p1 = S suppresses printout

LASTRAN resets the flag used by REPEAT to define which routine is to generate the problem.

p1 = 1 READATA, 2 RANDOM, 3 RANDY, 4 MREAD.

LIST lists details of the problem in hand.

p = M minterms, p = D dontcares, p = P prime implicants, p = A all of these. More than one of the above options may be selected. The format of items in the minterm or dontcare lists is specified by p = O octal, p = B binary, default is decimal for minterms and dontcares.

p = RS complements the problem for listing purposes.

p = number, sets the page width, default is 72.

p = S prints output to file PRNT and not the terminal.

MREAD reads in a multiple output problem. The first input line specifies the format of the input as either INDIVIDUAL in which the output functions are specified separately in the same way as for READATA; or COMBINED, in which only (0,1,-) are used to specify the NV variables of a combined input/output table.

p1 = F, p2 = R, p = S may be used as for READATA.

NET applies the algorithm to solving the table so as to yield an optimal solution.

NO sets number of output variables for multiple output problems

p1 = value, default is zero (single output).

NODE outputs node information during a pause in the execution of NET.

p1 = address (octal) of index table entry.

p2 = address 2 may define a range of nodes to be examined.

p2 or p3 = R relocates these numbers as relative, default is M.

p3 or p4 = number, the node type to be printed (default is all types).

p = S suppresses the successor and used sibling lists.

NV sets the number of input variables for single output problems or sum of input and output variables for multiple output problems.

p1 = value.

OUT outputs the solution.

p1 = S suppresses printing of individual terms, only the statistical information is listed.

PFL resets the amount of working storage used in blank common. p1 = number of words (octal), default is maximum available.

PITABLE reproduces the PI table, either on the terminal or on file PRNT if it is too wide.

p1 = 4 forces output to file PRNT.

POS examines the tree being constructed for a wide variety of faults; also prints information relative to its current size.

p1 = number of times POS is to be called, default is one immediate call.

p2 = number of branching stages of the tree between each POS call.

p3 = S suppresses printout unless a fault is found.

QMTONET sets up the prime implicant table and computes PI costs ready for executing NET, HEAVY or PITABLE.

p1 = S suppresses printout.

QUICK is used during a pause in executing NET, and causes the algorithm to stop as soon as a PI cover is available. The "solution" thus obtained is unlikely to be optimum, but will be an irredundant cover.

RANDOM generates a random single output problem having about $2^n/2$ minterms and $2^n/8$ dontcares.

p = S suppresses printout.

RANDY as for RANDOM except that the numbers of minterms and dontcares are truly random.

p = S suppresses printout.

READATA reads in a single output problem consisting of a list of minterms followed by a slash(/) and a list of dontcares. Data is terminated by end-of-file or by a period. List items are delimited by any other characters or by end of line (80 characters). The format of the numbers may be specified by embedding a D for decimal (default), O for octal, or B for binary in the input string. Binary format accepts 0 or 1 or - .

p1 = F reads data from file DATA, default is the terminal.

p2 = R rewinds DATA first.

p = S suppresses printout.

REPEAT regenerates the most recent problem: file DATA is not used.

p1 = S suppresses printout.

RESTART resets all flags and counters and causes the program to restart.

RETURN returns to continue the algorithm after a pause.

p1 = count of pause calls to be reached before pausing again.

p2 = string, only pauses passing this string are counted.

p3 = P prints all pause strings up to the next pause, default suppresses these.

Special case: p2 = "TIME" causes pausing to cease for p1 CP seconds (p3 ignored).

REWIND rewinds a file.

p1 = tape number, default is 1, DATA.

RSWAP converts the problem to its complement. The terms of the solution set, when complemented, represent a product of sums realisation.

p1 = S suppresses printout.

RTOQM changes the format of the storage from its initial form at generation to that suitable for CONSENSUS.

SEED sets a random number seed for use by the random problem generators.

p1 = octal digit string of up to 20 characters.

SKIP skips logical files in file DATA.

p1 = number to be skipped forward (positive) or backward (negative), default is forward 1.

SOLUTION extracts a solution during execution of the algorithm. Normally used in conjunction with the timer when some form of result is required before time limit is reached.

SPECIAL repeats the problem, solves it, prints the solution and checks it. This is equivalent to the series of commands: REPEAT, HWA, QMTONET, NET, OUT, CHECK.
p1 = RS complements the problem.
p = S suppresses printout of everything except the solution statistics.

STATS lists the contents of the /STATS/ common block in compressed format. All information pertaining to the problem and its solution is provided.

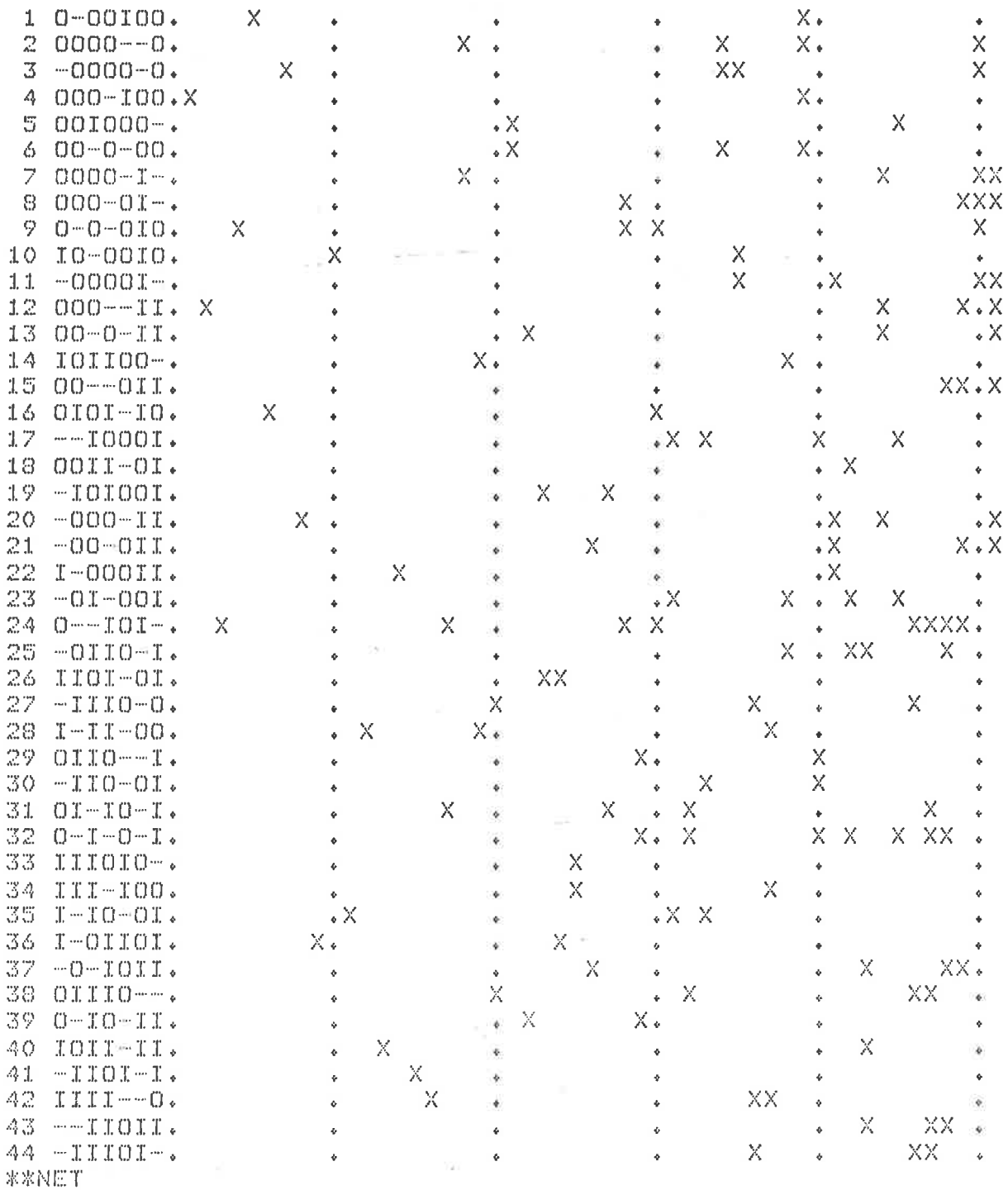
TEACH lists all recognisable commands.

TIME displays number of CP seconds elapsed.
p1 = A absolute, i.e. since LOGIN, default is time from starting NET (only meaningful during a pause).

```
PLEASE LOGIN
LOGIN,NEBLIUU,ELECENG


78/07/18    LOGGED IN AT  15.30.07.
            WITH USER-ID JV
            EQUIP/PORT 05/065
COMMAND- FETCH OPTIMIS
COMMAND- EFL 60000
COMMAND- OPTIMIS
ADDRESS OF COM(0): 34715B, CURRENT FIELD LENGTH: 060000B
CONTROL-
**TEACH
                    ---- CONTROL COMMANDS ----
                (FOR DETAILS REFER TO HANDBOOK)
 CHECK      CONNECT    CONSENSUS COSTS      DISABLE    DISCONNECT
 DUMP       ENABLE     END       HEAVY      HWA        LASTRAN
 LIST       MREAD      NET       NO         NODE       NV
 OUT        PFL        PITABLE   POS        QMTONET    QUICK
 RANDOM     RANDY      READATA   REPEAT     RESTART    RETURN
 REWIND     RSWAP      RTOQM     SEED       SKIP       SOLUTION
 SPECIAL    STATS      TEACH     TIME
**NV 7
**READATA
ENTER DATA IN BINARY(0,1,-) OR OCTAL OR DECIMAL(DEFAULT),
WITH ANY DELIMITERS INCLUDING E-O-L (80 CHARS).
NUMBER TYPES RESET WITH B,O,D RESPECTIVELY. MINTERMS
TERMINATED BY SLASH, DATA TERMINATED BY PERIOD OR E-O-R
0 2 3 4 6 7 10 11 12 15 16 17 23 25 59 58 57 56 51 49 46 43 42
41 36 34 27 26 64 66 67 71 75 77 81 82 85 88 89 91 92 95 126
124 122 119 116 113 109 105 99 / 19 20 29 53 55 117 120 123.

            3:35 P.M. 18 JULY  1978
NUMBER OF VARIABLES:  7
NUMBER OF MINTERMS:   51,  NUMBER OF DONTCARES:    8
**LIST MINTERMS DONTCARES 63
MINTERMS:      51
  0     2     3     4     6     7    10    11    12    15    16    17    23
 25    26    27    34    36    41    42    43    46    49    51    56    57
 58    59    64    66    67    71    75    77    81    82    85    88    89
 91    92    95    99   105   109   113   116   119   122   124   126
DONTCARES:      8
 19    20    29    53    55   117   120   123
**HWA
NUMBER OF PRIME IMPLICANTS:   44
**QMTONET
COST COEFFICIENTS:  4608      1      0      0    2.815E+14
**PITABLE
```



```
**NET
**OUT
SOLUTION: 25 TERMS, 132 LITERALS, 25 GATES, TOTAL COST: 115332
 0-I-0-I   -0I-00I   -00-0II   IIIOIO-   -IOIOOI   0-IO-II   00-0-00
 OIIIO--   0000--0   IIII--0   -IIOI-I   I-000II   IOII-II   I-II-00
 I-IO-0I   IO-00IO   I-0IIOI   -000-II   -0000-0   0IOI-IO   0-00I00
 0-0-0IO   0--IOI-   000--II   000-I00
**CHECK
THIS SOLUTION IS AT LEAST AN IRREDUNDANT COVER PRIME IMPLICANTS
**END
      STOP  END PROGRAM
        .574 CP SECONDS EXECUTION TIME
COMMAND- LOGOUT
CPA        .886 SEC.          .886 ADJ.
SYS TIME                     1.448
CONNECT TIME     0 HRS.      9 MIN.
 78/07/18  LOGGED OUT AT 15.39.13.
      <
```

```
EXAMPLE:  1
INPUT VARIABLES:  9
OUTPUT VARIABLES: 1
MINTERMS:      90
  4   25   29   35   38   48   50   65   66   71   78   83
 87   98  105  111  114  118  121  124  125  131  133  146
147  157  160  163  171  179  182  191  200  207  211  219
222  229  233  234  239  242  243  245  256  264  265  269
270  272  276  278  285  297  301  310  312  319  330  334
337  338  341  375  381  398  405  420  425  428  429  433
436  440  445  446  450  461  462  476  482  483  485  489
492  502  504  506  507  509
DONTCARES:     87
  7    8   27   30   33   46   53   74   80   90   92  116
129  134  141  143  145  149  151  152  154  158  165  170
172  176  177  181  189  197  198  206  221  228  237  246
252  257  262  263  274  277  279  280  283  284  287  291
296  300  314  315  316  323  326  328  339  345  347  354
359  367  371  374  376  378  386  389  391  395  402  407
416  418  447  456  468  470  473  474  480  494  495  497
503  510  511
SOLUTION:  52 TERMS,   399 LITERALS,  52 GATES.
-IOO-OIOI  OIOO--IOI  IOOOI-IO-  I-IIIOII-  OOIIIIIO-
OIIOOIII-  OOOIOOO-I  0-IOIOOII  -IOOIOOIO  II-IOIIOO
-IIIOIOOI  OOI-OOOIO  OOOOII-OI  OIOIO-OII  -IOIOOOOO
OIO--OOII  OOOIIOO-0  -IOIIOOOI  I-OIOI-OI  OOII-IOOI
I-IIIIIOI  IO-OIOOIO  IOIOIO-OI  IOO-II-II  IOO-IOIIO
IOO-OI-OI  IOOO--OOO  OOIO-OIII  OI--IOOII  0-IIIO-IO
-OIOOI-IO  OIIO-IIIO  IIIIII-I-  -IIIOOIOI  IIIIOOOI-
IIIOI-IOO  IIIOOIIOI  II--OOOIO  II-IIIII-  I--IIIOOO
IIOI-OIOO  I--OOIIIO  OI-I-OIOI  OI-IOIOIO  OIIOI-OII
-IIOOIOOO  -IOIIII-I  OI-IIOIIO  --IIOIIII  OOIOOOOOI
OOOIO-IIO  OOOOOOIOO
```

```
EXAMPLE:  2
INPUT VARIABLES:  9
OUTPUT VARIABLES: 1
MINTERMS:     101
  4     8    15    19    25    26    32    38    51    52    55    57
 65    66    69    72    74    80    88    99   101   103   109   113
114   115   119   122   127   132   137   143   144   145   146   148
149   158   164   175   177   191   195   197   201   211   221   236
251   257   259   272   274   276   285   286   287   293   303   314
317   320   321   332   333   337   341   343   346   349   360   362
363   370   386   397   400   402   406   408   411   413   418   421
431   432   446   453   455   456   458   466   471   472   474   475
478   493   494   504   509
DONTCARES:     42
  2    10    12    30    62    82    86    97   135   141   169   180
184   189   202   209   217   225   244   260   262   273   289   291
296   300   313   315   350   354   365   366   373   377   399   412
436   459   470   482   503   511
SOLUTION:  59 TERMS,    462 LITERALS,  59 GATES.
IIOOI-OOO  0-OOOOIOO  IOIOI-IOI  IIII-IIOI  IIIO-OIII
-IIOOOIOI  IIO-IOOOO  OIOI-IIII  OO-OO-OIO  IIOO-IIOI
IIO-OOOIO  IOII-OOIO  OOI-OO-OI  IO-IOIOOO  IOIOIOI-I
IO-O-OOOI  IOO-IIIOI  OOOO-IOIO  IOOOIIII-  IOOO-OIOO
OO-OOIO-O  OOIIO-IOI  OOIIOO--I  OOIOI-OOO  OI-OOIOOI
III-IIOOO  I-IIOIIIO  IIIO-IO-O  IIOIIIIIO  I-OIOOIOI
II-OIIOII  II-OIO-IO  IOIIOIOI-  I-IOII-IO  IOIOOIIO-
IOIOOOOO-  IOOIIIOI-  I-OIOIIII  I-OOIOO-O  IOO-OOO-I
OIIIIIOII  OIIIOIIOO  OIIOII-OI  OIIO-OOII  OIO-IOOOI
OIO--OIOO  O-OOIIIIO  OIOOIO-O-  -IOOIOO-O  OOIII-III
OOIII-OIO  OOII-OO-I  OO-IIO-II  O-OIIOIOO  OOOIOOIIO
OOOIOOOOO  OOO-IIOOI  OOO-IOOII  O-OOOIIII
```

```
EXAMPLE:  3
INPUT VARIABLES: 10
OUTPUT VARIABLES: 1
MINTERMS:     108
   2     12     22     26     42     43     49     52     58     59
  63     65     93    107    117    122    124    137    143    157
 158    160    162    192    198    199    202    213    215    216
 222    225    253    255    272    290    295    304    306    307
 313    314    329    350    366    374    389    390    402    407
 414    416    437    458    469    481    538    542    577    584
 597    610    619    634    643    656    669    670    698    710
 761    762    763    769    774    780    783    797    801    804
 812    814    816    818    822    824    828    848    870    871
 891    895    901    910    911    917    931    940    948    960
 962    968    974    980    986    990    993   1018
DONTCARES:     90
   9     18     24     30     33     40     71     78     84    101
 103    109    115    134    136    165    170    184    193    205
 214    229    258    261    302    327    334    337    340    355
 370    371    398    400    406    412    436    442    448    476
 492    493    498    499    517    521    525    529    535    537
 576    591    601    616    625    628    629    654    663    664
 676    694    699    711    720    724    732    733    740    741
 750    751    775    795    821    825    851    880    887    906
 929    937    944    952    956    973    979   1013   1016   1017
SOLUTION:   74 TERMS,    665 LITERALS,   74 GATES.
I-IIIIIOIO  0-OOIIIOIO  II-OII-OOO  -IIIIOOOOI  IIIIOII-IO
0-IIIOOOOI  III-OOIIIO  OOIIO-OII-  IIIOII--OO  OOOOI-IOI-
0-IIOOOOOO  OIIOOIOO-O  OIIOO--IIO  OI-OOOOIOI  OOIOIOOO-O
IIOOIOII-O  OIOI-OIIIO  II-OOOIIII  0-OOOOOOIO  OOIOOOIOO-
IOIIIIIO-I  -O-OOIIIIO  IOIOOI-OOO  OIOO-OOOIO  OIOO-IOOOO
-OOIIIIOIO  OOOII-OIOI  IOOIOO-OOO  -OOIOOOOOI  OOI-OIIIIO
I-IIOIOIOO  IIIIOO-OOO  IIIIOOOO-O  II-OI-IIOO  IIIOIOOO-I
IIIOO-OIOI  IIOIIII-II  IIOIIOOII-  IIOI-IOOOO  IIOOIIO-IO
IIOOIO-IOO  IIOOOIIIOI  IIOO-OIIOO  IIOOOOOII-  IIOO-OOOOI
-OIIOOOII-  IOI-IIIOI-  IOIOOOOOII  -OOIIOIOII  IOOIIOOOIO
IOOI-IOIOI  -OOOOII-IO  O-IIOIOIOI  OIIOIIOIO-  O-IOIOOOOO
OIIOOIOII-  OIOIIIO-IO  OIOIO-IIIO  OIOIOOIOOI  -IOOIIIOOI
OIO-IIOOI-  OIOOIOOIII  OOIIIIII-I  OOIIOIIOOO  O-IIOOIOIO
-OIOOIIIOI  OOIOOOIIII  OOOIIIIIOO  OOOIOIIIOI  OOOOIII-II
OOOOIIOIOO  OOOOI-OOOI  OOOOOI--IO  OOOOOOIIOO
```

```
EXAMPLE:  4
INPUT VARIABLES: 10
OUTPUT VARIABLES: 1
MINTERMS:       53
  34     65     74     75    120    166    172    207    235    240
 269    276    303    390    404    448    464    474    480    521
 534    535    539    555    558    582    587    630    645    660
 665    695    712    726    740    747    756    757    759    762
 765    787    843    869    889    906    911    918    942    961
 966    976   1023
DONTCARES:     642
   0      1      3      5      7      9     11     12     13     15
  16     17     19     20     21     22     24     25     26     27
  28     32     35     36     37     39     40     41     43     44
  45     50     51     52     53     54     55     56     59     60
  61     64     66     71     73     76     77     78     79     80
  81     82     85     86     87     92     93     94     95     96
  97     98     99    100    101    103    104    107    108    110
 112    116    117    118    119    121    122    123    124    125
 126    127    128    129    131    132    134    135    137    138
 139    140    141    142    143    144    145    146    149    150
 151    153    154    156    160    162    164    168    170    171
 174    175    176    177    178    179    180    181    183    184
 185    187    188    189    190    192    193    194    195    196
 198    199    200    203    204    205    209    214    215    216
 217    218    219    221    222    223    225    226    227    229
 232    234    236    238    242    244    245    246    247    249
 251    252    253    254    256    257    259    262    263    264
 266    268    270    271    273    274    275    284    285    287
 288    289    290    293    294    295    296    298    299    300
 304    305    306    308    309    310    311    312    313    315
 317    318    319    321    322    323    324    325    328    330
 332    333    334    337    338    340    341    342    343    344
 345    348    349    352    353    354    355    356    357    360
 361    362    364    366    367    368    369    371    372    373
 376    379    380    382    383    384    385    387    388    389
 391    392    393    394    395    396    397    399    402    403
 405    406    410    411    412    414    415    416    417    419
 421    422    423    424    425    428    429    432    433    434
 435    436    437    438    440    442    443    444    445    446
 450    451    452    453    454    455    456    458    460    462
 463    466    469    470    471    475    476    481    482    483
 485    487    488    490    491    492    493    495    499    501
 502    506    509    510    514    515    516    517    518    522
 523    524    525    526    527    530    533    536    540    544
 546    547    548    551    553    554    557    559    560    561
 562    564    565    567    568    569    571    572    574    575
 576    577    581    583    584    585    588    590    591    592
 595    596    597    598    599    600    601    602    603    604
 605    606    607    609    610    611    612    613    614    616
 618    619    620    622    624    626    628    629    631    633
 634    635    636    638    639    642    643    644    647    650
 654    655    656    657    659    661    663    664    666    667
 668    669    672    675    678    679    681    684    685    686
 688    689    690    691    692    693    694    696    697    698
 699    700    702    703    705    707    708    709    711    713
 716    718    719    720    721    722    724    729    730    734
 735    736    737    738    739    741    742    743    744    745
 746    748    749    750    751    752    755    758    760    761
 763    764    766    767    768    770    773    774    775    778
 779    780    781    782    783    785    786    788    789    792
```

```
 793   794   795   796   797   798   800   801   802   803
 805   807   808   809   811   812   814   815   817   818
 820   824   825   827   828   830   832   833   835   837
 838   839   841   844   846   847   850   852   853   855
 856   857   858   859   860   861   862   863   865   867
 868   872   874   876   878   879   880   883   884   885
 886   887   888   890   891   892   893   894   895   897
 899   901   903   904   905   910   912   914   915   916
 919   920   922   923   924   925   926   928   929   930
 933   934   936   937   938   939   940   941   943   944
 946   947   951   952   953   954   956   958   959   960
 962   965   967   968   970   972   973   974   975   977
 979   980   984   989   990   991   993   994   995   997
 999  1001  1003  1006  1007  1009  1011  1014  1016  1017
1018  1020
SOLUTION:  33 TERMS,   227 LITERALS,  33 GATES.
0---IOI-00  IOIII-I---  -0-IIO-0II  00--00000-  --0000IIO-
I----0IIIO  0IIIO--0IO  IIIO--I-IO  0-0-I000-0  IIOI-II---
IOIOOI--0-  I----000IOI  -IOO-0-III  I-0-0-IOII  0I--I0000-
I--I-00-0I  -IO-0I-I00  --I-00-III  II-I--IIII  0-IO-00IIO
IO-I-0I-00  -000-0IO-I  0-0II---00  -IIOOI-I-0  -I-00I00I-
IOIII--I--  -0-I-I-IIO  -0I-IIO-00  I00-0-0IIO  -IIIOO--IO
IO-0-IOI-I  000I00I-I-  -IIIO-0000
```

```
EXAMPLE:  5
INPUT VARIABLES:  5
OUTPUT VARIABLES: 5
MINTERMS:      11      (OUTPUT 1)
 3   4   8  11  12  13  15  16  20  26  29
DONTCARES:      8      (OUTPUT 1)
 7  10  18  19  22  23  27  28
MINTERMS:      16      (OUTPUT 2)
 1   4   5   8   9  10  11  13  15  21  22  25  26  27  28  30
DONTCARES:      3      (OUTPUT 2)
14  17  29
MINTERMS:      13      (OUTPUT 3)
 0   2   3   6   7   9  10  12  13  18  26  28  31
DONTCARES:      4      (OUTPUT 3)
 1   8  20  24
MINTERMS:      12      (OUTPUT 4)
 2   5   6   7   8  11  14  17  19  21  24  30
DONTCARES:      7      (OUTPUT 4)
 1  10  16  20  26  27  28
MINTERMS:      13      (OUTPUT 5)
 0   3   7   8   9  11  12  13  15  18  22  24  30
DONTCARES:      5      (OUTPUT 5)
17  20  21  23  26
SOLUTION:  20 TERMS,     75 LITERALS,  20 GATES.
I-OIO-O-O-  -IOOOOOO--  OIO-O---OO  IIIOOO--OO  OI-II--OO-
OI-O-OO-O-  OO-II-O-O-  IO-O-OOO-O  O-OOOOO-O-  I-IIOO-OO-
-IIO--OOOO  OO-IOOO--O  -IOI-O-O-O  ----OIO-OOO  IOO-IOOO-O
IO--O-OOOO  -I-IOOOO-O  OOI-IOOO-O  OOIOO--OOO  IIIIIOO-OO

EXAMPLE:  6
INPUT VARIABLES:  5
OUTPUT VARIABLES: 5
MINTERMS:      14      (OUTPUT 1)
 1   7   8  11  15  17  18  19  21  23  24  27  29  30
DONTCARES:      3      (OUTPUT 1)
 4  25  31
MINTERMS:       8      (OUTPUT 2)
 4  15  17  18  19  21  26  28
DONTCARES:      2      (OUTPUT 2)
 0  13
MINTERMS:      16      (OUTPUT 3)
 3   6   7  10  11  12  13  14  19  20  22  23  24  25  28  29
DONTCARES:      1      (OUTPUT 3)
18
MINTERMS:       7      (OUTPUT 4)
10  18  19  24  25  29  30
DONTCARES:      1      (OUTPUT 4)
 4
MINTERMS:      10      (OUTPUT 5)
 3   6   7   8  11  13  16  19  26  27
DONTCARES:     12      (OUTPUT 5)
 0   1   2   5   9  14  22  23  24  25  28  31
SOLUTION:  21 TERMS,     93 LITERALS,  21 GATES.
OIOII-O-O-  I--II-OOO-  -O-IIOO-O-  O-IIOOO-O-  OIOIOOO--O
IIOO-OO--O  IIII0-OO-O  II-OI-OO-O  O--OIOOOO-  -IIO-OO-OO
IIOIOO-OO-  IO-OI--OOO  -OOOI-OOOO  IOI-OOO-OO  --III-OOOO
IOOI---O-O  OO-OOO-OOO  OII-IO-OOO  IIIOOO-OOO  --OOOOOOO-
-IOOO-OOOO
```

```
EXAMPLE:  7
INPUT VARIABLES:  4
OUTPUT VARIABLES: 1
MINTERMS:       2
 0   6
DONTCARES:     11
 1   2   3   4   5   8   9  10  11  12  13
SOLUTION:    1 TERMS,      2 LITERALS,    1 GATES.
0--0

EXAMPLE:  8
INPUT VARIABLES:  4
OUTPUT VARIABLES: 1
MINTERMS:       9
 0   1   4   5   7   8  10  14  15
DONTCARES:      0
SOLUTION:    4 TERMS,     11 LITERALS,    4 GATES.
I-IO  IO-0  -III  0-0-

EXAMPLE:  9
INPUT VARIABLES:  4
OUTPUT VARIABLES: 1
MINTERMS:      14
 1   2   3   4   5   6   7   8   9  10  11  12  13  14
DONTCARES:      0
SOLUTION:    4 TERMS,      8 LITERALS,    4 GATES.
IO--  -IO-  --IO  0--I

EXAMPLE: 10
INPUT VARIABLES:  4
OUTPUT VARIABLES: 1
MINTERMS:       9
 1   2   3   5   8   9  11  14  15
DONTCARES:      0
SOLUTION:    5 TERMS,     14 LITERALS,    5 GATES.
-0-I  III-  IOO-  0-0I  OOI-

EXAMPLE: 11
INPUT VARIABLES:  5
OUTPUT VARIABLES: 1
MINTERMS:      12
 0   1   2   5   6   7  24  25  27  28  30  31
DONTCARES:      0
SOLUTION:    6 TERMS,     24 LITERALS,    6 GATES.
III-0  II-II  IIOO-  OOI-I  OO-IO  OOOO-

EXAMPLE: 12
INPUT VARIABLES:  5
OUTPUT VARIABLES: 1
MINTERMS:       4
 8  14  16  19
DONTCARES:     12
20  21  22  23  24  25  26  27  28  29  30  31
SOLUTION:    4 TERMS,     14 LITERALS,    4 GATES.
I--II  I--OO  -III0  -IOOO
```

```
EXAMPLE: 13
INPUT VARIABLES:  5
OUTPUT VARIABLES: 1
MINTERMS:      10
 0   3   4   6   7  15  21  23  26  28
DONTCARES:      7
 2   8  12  14  17  24  31
SOLUTION:    6 TERMS,     20 LITERALS,    6 GATES.
0-II-  0--00  IOI-I  -I-00  IIO-0  00-I-

EXAMPLE: 14
INPUT VARIABLES:  5
OUTPUT VARIABLES: 1
MINTERMS:      15
 0   2   3   4   8  10  11  12  16  20  24  26  27  28  31
DONTCARES:      0
SOLUTION:    4 TERMS,     12 LITERALS,    4 GATES.
-IO-0  II-II  ---00  0-0I-

EXAMPLE: 15
INPUT VARIABLES:  6
OUTPUT VARIABLES: 1
MINTERMS:      16
 0   1   2   5   8  13  18  19  28  34  37  39  47  56  57  61
DONTCARES:      9
 4  12  16  20  23  38  49  59  62
SOLUTION:   10 TERMS,     46 LITERALS,   10 GATES.
OIOOI-  -OOIOI  -OOOIO  III-OI  IIIOO-  IO-III  0--IOO  00-IO-
00--00  OOO-0-

EXAMPLE: 16
INPUT VARIABLES:  7
OUTPUT VARIABLES: 1
MINTERMS:      92
  1    2    3    4    5    6    7    8    9   10   11   14
 15   16   17   18   19   20   21   22   23   24   25   26
 27   28   29   30   31   32   34   35   37   39   44   47
 48   49   50   51   52   53   54   55   56   57   58   61
 62   63   64   83   84   85   86   87   88   89   90   95
 96   97   98   99  100  101  102  103  104  105  106  107
108  109  110  111  112  113  114  115  116  117  118  119
120  121  122  123  124  125  126  127
DONTCARES:      0
SOLUTION:   19 TERMS,     75 LITERALS,   19 GATES.
0--OOI-  0-I--IO  0-I--0I  00-0--I  0---III  0--0I-I  II------
--I-III  --IIO-0  --IIOO-  --IOI--  --IO-II  I-OOOOO  -IOIIOO
-I-OO-0  OOI-----  OO---I-  OO-IO--  OO-OI--
```

```
EXAMPLE: 17
INPUT VARIABLES:  7
OUTPUT VARIABLES: 1
MINTERMS:       4
 57    65    67    70
DONTCARES:      81
 10    11    12    13    14    15    26    27    28    29    30    31
 42    43    44    45    46    47    58    59    60    61    62    63
 71    72    73    74    75    76    77    78    79    80    81    82
 83    84    85    86    87    88    89    90    91    92    93    94
 95    96    97    98    99   100   101   102   103   104   105   106
107   108   109   110   111   112   113   114   115   116   117   118
119   120   121   122   123   124   125   126   127
SOLUTION:    3 TERMS,     10 LITERALS,    3 GATES.
I---II-   I---0-I   -III--I

EXAMPLE: 18
INPUT VARIABLES:  8
OUTPUT VARIABLES: 1
MINTERMS:       45
 25    40    41    55    56    57    70    71    72    73    85    86
 87    88    89   100   101   102   103   104   105   115   116   117
118   119   120   121   130   131   132   133   134   135   136   137
145   146   147   148   149   150   151   152   153
DONTCARES:    156
 10    11    12    13    14    15    26    27    28    29    30    31
 42    43    44    45    46    47    58    59    60    61    62    63
 74    75    76    77    78    79    90    91    92    93    94    95
106   107   108   109   110   111   122   123   124   125   126   127
138   139   140   141   142   143   154   155   156   157   158   159
160   161   162   163   164   165   166   167   168   169   170   171
172   173   174   175   176   177   178   179   180   181   182   183
184   185   186   187   188   189   190   191   192   193   194   195
196   197   198   199   200   201   202   203   204   205   206   207
208   209   210   211   212   213   214   215   216   217   218   219
220   221   222   223   224   225   226   227   228   229   230   231
232   233   234   235   236   237   238   239   240   241   242   243
244   245   246   247   248   249   250   251   252   253   254   255
SOLUTION:   12 TERMS,     36 LITERALS,   12 GATES.
I--I---I   I---I---   I----I--   I-----I-   -III--II   -II--I--
-I-I-I-I   -I--I---   -I---II-   --II-III   --I-I---   ---II--I
```

```
EXAMPLE: 19
INPUT VARIABLES:  9
OUTPUT VARIABLES: 1
MINTERMS:         4
  16    32    64   128
DONTCARES:     503
   0     3     5     6     7     9    10    11    12    13    14    15
  17    18    19    20    21    22    23    24    25    26    27    28
  29    30    31    33    34    35    36    37    38    39    40    41
  42    43    44    45    46    47    48    49    50    51    52    53
  54    55    56    57    58    59    60    61    62    63    65    66
  67    68    69    70    71    72    73    74    75    76    77    78
  79    80    81    82    83    84    85    86    87    88    89    90
  91    92    93    94    95    96    97    98    99   100   101   102
 103   104   105   106   107   108   109   110   111   112   113   114
 115   116   117   118   119   120   121   122   123   124   125   126
 127   129   130   131   132   133   134   135   136   137   138   139
 140   141   142   143   144   145   146   147   148   149   150   151
 152   153   154   155   156   157   158   159   160   161   162   163
 164   165   166   167   168   169   170   171   172   173   174   175
 176   177   178   179   180   181   182   183   184   185   186   187
 188   189   190   191   192   193   194   195   196   197   198   199
 200   201   202   203   204   205   206   207   208   209   210   211
 212   213   214   215   216   217   218   219   220   221   222   223
 224   225   226   227   228   229   230   231   232   233   234   235
 236   237   238   239   240   241   242   243   244   245   246   247
 248   249   250   251   252   253   254   255   257   258   259   260
 261   262   263   264   265   266   267   268   269   270   271   272
 273   274   275   276   277   278   279   280   281   282   283   284
 285   286   287   288   289   290   291   292   293   294   295   296
 297   298   299   300   301   302   303   304   305   306   307   308
 309   310   311   312   313   314   315   316   317   318   319   320
 321   322   323   324   325   326   327   328   329   330   331   332
 333   334   335   336   337   338   339   340   341   342   343   344
 345   346   347   348   349   350   351   352   353   354   355   356
 357   358   359   360   361   362   363   364   365   366   367   368
 369   370   371   372   373   374   375   376   377   378   379   380
 381   382   383   384   385   386   387   388   389   390   391   392
 393   394   395   396   397   398   399   400   401   402   403   404
 405   406   407   408   409   410   411   412   413   414   415   416
 417   418   419   420   421   422   423   424   425   426   427   428
 429   430   431   432   433   434   435   436   437   438   439   440
 441   442   443   444   445   446   447   448   449   450   451   452
 453   454   455   456   457   458   459   460   461   462   463   464
 465   466   467   468   469   470   471   472   473   474   475   476
 477   478   479   480   481   482   483   484   485   486   487   488
 489   490   491   492   493   494   495   496   497   498   499   500
 501   502   503   504   505   506   507   508   509   510   511
SOLUTION:    1 TERMS,      5 LITERALS,    1 GATES.
0----0000

EXAMPLE: 20
INPUT VARIABLES:  6
OUTPUT VARIABLES: 1
MINTERMS:        39
  2    4    7    8    9   10   11   12   14   16   17   18   19   20   21   22
 26   27   29   30   31   32   33   35   36   37   38   40   41   45   47   48
 51   52   53   55   56   58   63
DONTCARES:       0
SOLUTION:   16 TERMS,     72 LITERALS,   16 GATES.
IIOI-I   OI-OI-   I--000   III0-0   I-IIII   IO--OI   --0I00   I-00II
OIII-I   OI--IO   OIO-0-   00I--0   00IO--   IOOI-0   000III   0--0IO
```

```
EXAMPLE: 21
INPUT VARIABLES:  9
OUTPUT VARIABLES: 1
MINTERMS:     100
 19   49   50   51   79   80   81   82   83  109  110  111
112  113  114  115  139  140  141  142  143  144  145  146
147  169  170  171  172  173  174  175  176  177  178  179
199  200  201  202  203  204  205  206  207  208  209  210
211  229  230  231  232  233  234  235  236  237  238  239
240  241  242  243  259  260  261  262  263  264  265  266
267  268  269  270  271  272  273  274  275  289  290  291
292  293  294  295  296  297  298  299  300  301  302  303
304  305  306  307
DONTCARES:    100
  0    1    2    3    4    5    6    7    8    9   10   11
 12   13   14   15   16   17   18   32   33   34   35   36
 37   38   39   40   41   42   43   44   45   46   47   48
 64   65   66   67   68   69   70   71   72   73   74   75
 76   77   78   96   97   98   99  100  101  102  103  104
105  106  107  108  128  129  130  131  132  133  134  135
136  137  138  160  161  162  163  164  165  166  167  168
192  193  194  195  196  197  198  224  225  226  227  228
256  257  258  288
SOLUTION:    4 TERMS,     12 LITERALS,    4 GATES.
-00--00--   -00-0-----  0-----00--  0---0-----

EXAMPLE: 22
INPUT VARIABLES:  6
OUTPUT VARIABLES: 1
MINTERMS:      38
 2   4   7   8   9  10  11  12  14  16  17  18  19  20  21  25
26  27  29  31  32  33  35  36  37  38  40  41  45  47  48  51
52  53  55  56  58  63
DONTCARES:      0
SOLUTION:  15 TERMS,     67 LITERALS,  15 GATES.
OIOO--   IIOI-I  OIO-0-  OOIO--  IIIO-0  I--OOO  I-IIII  IO--OI
I-OOII  OII--I  --OIOO  IOOI-0  OOI--0  OOOIII  0--OIO

EXAMPLE: 23
INPUT VARIABLES:  7
OUTPUT VARIABLES: 1
MINTERMS:      63
  4    6    7    8    9   11   12   13   16   18   19   20
 23   24   25   28   30   33   34   36   37   39   40   41
 43   45   47   49   50   51   53   54   55   56   58   66
 68   72   75   77   80   82   85   86   89   90   96   97
 98  100  101  103  108  109  111  112  115  116  117  118
122  124  125
DONTCARES:     14
 15   21   29   44   48   69   70   76   79   83   94  106
113  119
SOLUTION:  23 TERMS,    111 LITERALS,  23 GATES.
--OIIO-  0--OIII  OOI--OO  -IOOOIO  -I-0-OI  --IOOII  -IO-I-I
II-0-0-  I-IIOIO  OOIII-0  --IOIOI  -OIOO-0  IO-0-IO  OII-0-0
-IIOII-  -OOOI-0  0-0I-0-  -IO-IO-  II--IO-  -OIIOOI  -OOI-II
-OOI-OO  0-OI--I
```

```
EXAMPLE: 24
INPUT VARIABLES:  7
OUTPUT VARIABLES: 1
MINTERMS:      68
  1     3     4     5    10    11    12    13    15    16    21    22
 23    25    27    28    29    30    33    34    35    37    38    40
 41    43    44    45    47    48    49    50    52    53    55    62
 64    67    69    70    71    74    76    77    78    79    80    87
 93    98   100   101   102   103   104   105   108   110   111   113
115   118   119   121   123   124   126   127
DONTCARES:   . 13
 17    20    42    46    57    66    72    83    86    88    89   116
120
SOLUTION:  21 TERMS,    102 LITERALS,  21 GATES.
I00I--0  -00II-I  I000-I-  -I0I00-  --00I0I  0I0I---  II--II-
III-0-I  II--I-0  -I00-I0  -0-IIOI  I0--000  -I-IIIO  --I0III
0I-00I0  00I-I-0  00II0-I  0-I0-0-  0-0I0I-  0-000-I  00--I0-
EXAMPLE: 25
INPUT VARIABLES:  8
OUTPUT VARIABLES: 1
MINTERMS:      33
 0    12    21    25    30    51    58    59    68    71    90    94
105   116   118   124   127   140   144   145   146   150   176   177
201   206   208   209   214   219   234   240   243
DONTCARES:    188
 1     2     3     4     5     6     7     8     9    10    11    13
14    15    16    17    18    23    24    26    27    28    29    31
32    34    36    37    38    40    41    42    43    44    46    48
49    50    52    53    54    55    56    57    60    62    63    64
66    67    69    70    72    73    74    75    77    78    80    81
82    83    84    85    86    87    88    89    91    92    93    95
96    97    98    99   100   101   102   103   104   106   108   110
111   113   114   115   117   119   120   121   122   123   125   126
128   129   130   131   132   133   134   135   136   137   138   139
141   142   143   147   148   149   151   152   153   154   155   158
159   160   161   163   164   165   166   167   168   171   172   173
174   179   180   181   182   183   185   186   187   188   189   193
194   195   196   197   198   200   202   203   204   205   207   210
211   212   215   216   218   221   222   223   225   226   228   229
230   231   232   233   235   236   238   239   242   244   245   246
247   248   249   250   251   252   253   255
SOLUTION:  11 TERMS,     47 LITERALS,  11 GATES.
-0-I0-0I  0-II--I-  0--II--0  II-0I0--  I-0---I0  -I-I--II
0---I00-  -000----  0I--0I--  I--I0-00  I-0I00--
```

```
EXAMPLE: 26
INPUT VARIABLES:  5
OUTPUT VARIABLES: 5
MINTERMS:      11      (OUTPUT 1)
 3   4   8  11  12  13  15  16  20  26  29
DONTCARES:      8      (OUTPUT 1)
 7  10  18  19  22  23  27  28
MINTERMS:      16      (OUTPUT 2)
 1   4   5   8   9  10  11  13  15  21  22  25  26  27  28  30
DONTCARES:      3      (OUTPUT 2)
14  17  29 .
MINTERMS:      13      (OUTPUT 3)
 0   2   3   6   7   9  10  12  13  18  26  28  31
DONTCARES:      4      (OUTPUT 3)
 1   8  20  24
MINTERMS:      12      (OUTPUT 4)
 2   5   6   7   8  11  14  17  19  21  24  30
DONTCARES:      7      (OUTPUT 4)
 1  10  16  20  26  27  28
MINTERMS:      13      (OUTPUT 5)
 0   3   7   8   9  11  12  13  15  18  22  24  30
DONTCARES:      5      (OUTPUT 5)
17  20  21  23  26
SOLUTION:  20 TERMS,     75 LITERALS,  20 GATES.
I-0I0-0-0-  -I000000--  0I0-0---00  III000--00  0I-II--00-
0I-0-00-0-  00-II-0-0-  I0-0-000-0  0-00000-0-  I-II00-00-
-II0--0000  00-I000--0  -I0I-0-0-0  ---0I0-000  I00-I000-0
I0--0-0000  -I-I0000-0  00I-I000-0  00I00--000  IIII100-00
```

Appendix E.

Cost Function for Classical Minimisation

Classical minimisation involves selecting a solution having the minimum number of terms and if a choice exists, the minimum number of literals.

In this case, the cost function defined in section 3.3 may be simplified to $C = Tt + Ll$, and a lower bound on the ratio T/L is given by $T/L \geq t_{max}(n - 1)$ where t_{max} is an upper limit on the number of terms in an optimum solution, and n is the number of variables.

To derive this expression, consider the unwanted situation where there exists 2 solutions such that $C_1 > C_2$ but $t_1 < t_2$:

$$C_1 > C_2$$

hence $Tt_1 + Ll_1 > Tt_2 + Ll_2$

ie $T/L < (l_1 - l_2)/(t_2 - t_1)$ $(t_2 - t_1 > 0$ by assumption)

in fact $t_2 - t_1 \geq 1$ since t_1 , t_2 are integers

$$T/L < l_1 - l_2$$
$$< nt_1 - l_2$$
$$< nt_1 - t_2 \quad (\text{since } t_2 \leq l_2)$$
$$< nt_2 - t_2 = t_2(n - 1)$$
$$< t_{max}(n - 1) \quad (\text{where } t_{max} \geq t_i \text{ for all } i)$$

Hence $T/L \geq t_{max}(n - 1)$ ensures that the lowest cost solution contains the least number of terms.

When n - 10, the value for t_{max} is 2^9 (the solution for the exclusive-or function), and so $T/L \geq 2^9(10 - 1) = 4608$.

ABHYANKAR S
"MINIMAL 'SUM OF PRODUCTS OF SUMS' EXPRESSIONS OF BOOLEAN FUNCTIONS"
IRE TRANS. ELECTRON. COMPUT., VOL.EC-7, PP.268-276 DECEMBER 1958

ABHYANKAR S
"ABSOLUTE MINIMAL EXPRESSIONS OF BOOLEAN FUNCTIONS"
IRE TRANS. ELECTRON. COMPUT., VOL.EC-8, PP.3-8 MARCH 1959

AITCHISON R E
"COMPUTER AIDED ANALYSIS AND DESIGN"
PROC. IREE AUST., VOL.31, PP.87-88 APRIL 1970

AKERS S B
"A TRUTH TABLE METHOD FOR THE SYNTHESIS OF COMBINATIONAL LOGIC"
IRE TRANS. ELECTRON. COMPUT., VOL.EC-10, PP.604-615 DECEMBER 1961

AKERS S B
"A DIAGRAMMATIC APPROACH TO MULTILEVEL LOGIC SYNTHESIS"
IEEE TRANS. ELECTRON. COMPUT., VOL.EC-14, PP.174-181 APRIL 1965

AL J
SEE: VINK H A, VAN DEN DOLDER B, AL J

ANDERSON J P
SEE: GORMAN D F, ANDERSON J P

AREVALO Z, BREDESON J G
"A METHOD TO SIMPLIFY A BOOLEAN FUNCTION INTO A NEAR MINIMAL SUM-OF-PRODUCTS FOR PROGRAMMABLE LOGIC ARRAYS"
IEEE TRANS. COMPUT., VOL.C-27, PP.1028-1039 NOVEMBER 1978

ASHENHURST R L
"THE DECOMPOSITION OF SWITCHING FUNCTIONS"
ANNALS OF THE HARVARD COMPUTATION LABORATORY, VOL.29, PP.74-116 1959

BALAS E
"AN ADDITIVE ALGORITHM FOR SOLVING LINEAR PROGRAMS WITH ZERO-ONE VARIABLES"
OPERATIONS RESEARCH, VOL.13, PP.517-546 JULY 1965

BARNARD D F, HOLMAN D F
"THE USE OF ROTH'S DECOMPOSITION ALGORITHM IN MULTI-LEVEL DESIGN OF CIRCUITS"
THE COMPUTER JOURNAL, VOL.11, PP.269-276 NOVEMBER 1968

BARTEE T C
"THE AUTOMATIC DESIGN OF LOGICAL NETWORKS"
PROC. WESTERN JOINT COMPUTER CONF., PP.103-107 MARCH 1959

BARTEE T C
"COMPUTER DESIGN OF MULTIPLE-OUTPUT LOGICAL NETWORKS"
IEEE TRANS. ELECTRON. COMPUT., VOL.EC-10, PP.21-30 MARCH 1961

BARTEE T C
"DIGITAL COMPUTER FUNDAMENTALS"
MCGRAW-HILL (4TH ED.) 1977

BARTEE T C, LEBOW I L, REED I S
"THEORY AND DESIGN OF DIGITAL MACHINES"
MCGRAW-HILL 1962

BASU A K
SEE: DE SARKAR S C, BASU A K, CHOUDHURY A K

BASU M S
SEE: CHOUDHURY A K, BASU M S

BASU M S
SEE: CHAKRABARTI K K, CHOUDHURY A K, BASU M S

BATNI R P, RUSSELL J D, KIME C R
"AN EFFICIENT ALGORITHM FOR FINDING AN IRREDUNDANT SET COVER"
J. ASSOC. COMPUT. MACH., VOL.21, PP.351-355 JULY 1974

BAUMERT L D
SEE: GOLOMB S W, BAUMERT L D

BERZTISS A T
"DATA STRUCTURES THEORY AND PRACTICE"
ACADEMIC PRESS, N.Y. 1975

BISWAS N N
"MINIMIZATION OF BOOLEAN FUNCTIONS"
IEEE TRANS. COMPUT.,VOL.C-20(2), PP.925-929 AUGUST 1971

BOWMAN R M, MCVEY E S
"A METHOD FOR THE FAST APPROXIMATE SOLUTION OF LARGE PRIME IMPLICANT CHARTS"
IEEE TRANS. COMPUT., VOL.C-19, PP.169-173 FEBRUARY 1970

BREDESON J G
SEE: AREVALO Z, BREDESON J G

BREDESON J G, HULINA P T
"GENERATION OF PRIME IMPLICANTS BY DIRECT MULTIPLICATION"
IEEE TRANS. COMPUT., VOL.C-20(1), PP.475-476 APRIL 1971

BREUER M A
"A GENERAL SURVEY OF DESIGN AUTOMATION OF DIGITAL COMPUTERS"
PROC. IEEE, VOL.54, NO.12, PP.1708-1721 DECEMBER 1966

BREUER M A
"HEURISTIC SWITCHING EXPRESSION SIMPLIFICATION"
PROC. 23RD NATIONAL CONF. OF THE ACM, PP.241-250 AUGUST 1968

BREUER M A
"SIMPLIFICATION OF THE COVERING PROBLEM WITH APPLICATION TO BOOLEAN EXPRESSIONS"
J. ASSOC. COMPUT. MACH., VOL.17, PP.166-181 JANUARY 1970

BREUER M A (ED)
"DESIGN AUTOMATION OF DIGITAL SYSTEMS"
VOL.1, PRENTICE-HALL 1972

BUBENIK V
"WEIGHTING METHOD FOR THE DETERMINATION OF THE IRREDUNDANT SET OF PRIME IMPLICANTS"
IEEE TRANS. COMPUT., VOL.C-21, PP.1449-1451 DECEMBER 1972

BURKE R E, VAN BOSSE J G
"NAND-AND CIRCUITS"
IEEE TRANS. ELECTRON. COMPUT., VOL.EC-14, PP.63-65 FEBRUARY 1965

BUTLER K J, WARFIELD J N
"A DIGITAL COMPUTER PROGRAM FOR REDUCING LOGICAL STATEMENTS TO A MINIMAL FORM"
PROC. NAT. ELECTRONICS CONF., VOL.15, PP.456-466 1959

CALDWELL S H
"SWITCHING CIRCUITS AND LOGICAL DESIGN"
WILEY 1958

CANTO D
SEE: DORMIDO B, CANTO D

CERNY E, MARIN M A
"A COMPUTER ALGORITHM FOR THE SYNTHESIS OF MEMORYLESS LOGIC CIRCUITS"
IEEE TRANS. COMPUT., VOL.C-23, PP.455-465 MAY 1974

CHAKRABARTI K K, CHOUDHURY A K, BASU M S
"COMPLEMENTARY FUNCTION APPROACH TO THE SYNTHESIS OF THREE-LEVEL NAND NETWORKS"
IEEE TRANS. COMPUT., VOL.C-19, PP.509-514 JUNE 1970

CHANG C -L
SEE: SLAGLE J R, CHANG C -L, LEE R C T

CHANG D M Y, MOTT T H
"COMPUTING IRREDUNDANT NORMAL FORMS FROM ABBREVIATED PRESENCE FUNCTIONS"
IEEE TRANS. ELECTRON. COMPUT., VOL.EC-14, PP.335-342 JUNE 1965

CHARLAND M J
"LOOK TO ASYNCHRONOUS SEQUENTIAL LOGIC"
ELECTRONIC DESIGN, VOL.22, NO.20, PP.98-103, 27TH SEPTEMBER 1974

CHOUDHURY A K
SEE: DAS S R, CHOUDHURY A K

CHOUDHURY A K
SEE: DE SARKAR S C, BASU A K, CHOUDHURY A K

CHOUDHURY A K
SEE: CHAKRABARTI K K, CHOUDHURY A K, BASU M S

CHOUDHURY A K, BASU M S
"A MECHANIZED CHART FOR SIMPLIFICATION OF SWITCHING FUNCTIONS"
IRE TRANS. ELECTRON. COMPUT., VOL.EC-11, PP.713-714 OCTOBER 1962

CHOUDHURY A K, DAS S R
"DIRECT DETERMINATION OF ALL THE MINIMAL PRIME IMPLICANT COVERS OF SWITCHING FUNCTIONS"
J. OF ELECTRONICS AND CONTROL, VOL.17, NO.5, PP.553-575 1964

COBHAM A, FRIDSHAL R, NORTH J H
'AN APPLICATION OF LINEAR PROGRAMMING TO THE MINIMIZATION OF BOOLEAN FUNCTIONS"
AIEE SYMP. ON SWITCHING CIRCUIT THEORY AND LOGICAL DESIGN, PP.3-9 1961 <OR> IBM RESEARCH REPORT RC-472 JUNE 1961

COBHAM A, FRIDSHAL R, NORTH J H
"A STATISTICAL STUDY OF THE MINIMIZATION OF BOOLEAN FUNCTIONS USING INTEGER PROGRAMMING"
IBM RESEARCH REPORT, RC-756 JUNE 1962

COBHAM A, NORTH J H
"EXTENSIONS OF THE INTEGER PROGRAMMING APPROACH TO THE MINIMIZATION OF BOOLEAN FUNCTIONS"
IBM RESEARCH REPORT, RC-915 1963

CRAWFORD B J
SEE: WAXMAN R, MCMAHON M T, CRAWFORD B J, DE ANDRADE A B

CURTIS H A
"A FUNCTIONAL CANONICAL FORM"
J. ASSOC. COMPUT. MACH., VOL.6, PP.245-258 1959

CURTIS H A
"MULTIFUNCTIONAL CIRCUITS IN FUNCTIONAL CANONICAL FORM"
J. ASSOC. COMPUT. MACH., VOL.6, PP.538-547 1959

CURTIS H A
"A GENERALIZED TREE CIRCUIT"
J. ASSOC. COMPUT. MACH., VOL.8, PP.484-496 1961

CURTIS H A
"A NEW APPROACH TO THE DESIGN OF SWITCHING CIRCUITS"
PRINCETON, N.J., VAN NOSTRAND 1962

CURTIS H A
"SYSTEMATIC PROCEDURES FOR REALIZING SYNCHRONOUS SEQUENTIAL MACHINES USING FLIP-FLOP MEMORY (PART 1)"
IEEE TRANS. COMPUT., VOL.C-18, PP.1121-1127 DECEMBER 1969

CURTIS H A
"SYSTEMATIC PROCEDURES FOR REALIZING SYNCHRONOUS SEQUENTIAL MACHINES USING FLIP-FLOP MEMORY (PART 2)"
IEEE TRANS. COMPUT., VOL.C-19, PP.66-73 JANUARY 1970

CURTIS H A
"ADJACENCY TABLE METHOD OF DERIVING MINIMAL SUMS"
IEEE TRANS. COMPUT., VOL.C-26, PP.1136-1141 NOVEMBER 1977

DAS S R
SEE: CHOUDHURY A K, DAS S R

DAS S R, CHOUDHURY A K
"MAXTERM TYPE EXPRESSIONS OF SWITCHING FUNCTIONS AND THEIR PRIME IMPLICANTS"
IEEE TRANS. ELECTRON. COMPUT., VOL.EC-14, PP.920-923 DECEMBER 1965

DAVIDSON E S
"AN ALGORITHM FOR NAND DECOMPOSITION UNDER NETWORK CONSTRAINTS"
IEEE TRANS. COMPUT., VOL.C-18, PP.1098-1109 DECEMBER 1969

DAVIDSON E S
SEE: LEE H -P, DAVIDSON E S

DAVIDSON E S, METZE G
"MODULE COMPLEXITY AND NAND NETWORK DESIGN ALGORITHMS"
PROC. 6TH. ANNUAL ALLERTON CONF. ON CIRCUIT AND SYSTEM THEORY OCTOBER 1968

DAVIDSON E S, METZE G
"COMMENTS ON 'AN ALGORITHM FOR SYNTHESIS OF MULTIPLE OUTPUT COMBINATIONAL LOGIC'"
IEEE TRANS. COMPUT., VOL.C-17, PP.1091-1092 NOVEMBER 1968

DE ANDRADE A B
SEE: WAXMAN R, MCMAHON M T, CRAWFORD B J, DE ANDRADE A B

DE SARKAR S C, BASU A K, CHOUDHURY A K
"SIMPLIFICATION OF INCOMPLETELY SPECIFIED FLOW TABLES WITH THE HELP OF PRIME CLOSED SETS"
IEEE TRANS. COMPUT., VOL.C-18, PP.953-956 OCTOBER 1969

DE TROYE N C
"CLASSIFICATION AND MINIMIZATION OF SWITCHING FUNCTIONS"
PHILIPS RESEARCH REPORTS, VOL.14, PP.151-193 AND PP.250-292 1959

DEFRANCESCO H F, LACROSSE T R
"AUTOMATED LOGICAL DESIGN"
IEEE INTERNATIONAL CONVENTION RECORD, VOL.11(1), PT.4, PP.94-101 MARCH 1963

DIETMEYER D L
SEE: SCHNEIDER F R, DIETMEYER D L

DIETMEYER D L
SEE: SU Y -H, DIETMEYER D L

DIETMEYER D L
"LOGIC DESIGN OF DIGITAL SYSTEMS"
BOSTON, MA: ALLYN AND BACON (2ND ED.) 1978

DIETMEYER D L
PERSONAL CORRESPONDENCE
30 OCTOBER 1978

DIETMEYER D L
PERSONAL CORRESPONDENCE
19 DECEMBER 1978

DIETMEYER D L, SCHNEIDER P R
"A COMPUTER-ORIENTED FACTORING ALGORITHM FOR NOR LOGIC DESIGN"
IEEE TRANS. ELECTRON. COMPUT., VOL.EC-14. PP.868-874 DECEMBER 1965

DIETMEYER D L, SCHNEIDER P R
"IDENTIFICATION OF SYMMETRY, REDUNDANCY, AND EQUIVALENCE OF BOOLEAN FUNCTIONS"
IEEE TRANS. ELECTRON. COMPUT., VOL.EC-16, PP.804-817 DECEMBER 1967

DIETMEYER D L, SU Y -H
"LOGIC DESIGN AUTOMATION OF FAN-IN LIMITED NAND NETWORKS"
IEEE TRANS. COMPUT., VOL.C-18, PP.11-22 JANUARY 1969

DOLOTTA T A
SEE: WEINER P, DOLOTTA T A

DOLOTTA T A, MCCLUSKEY E J
"THE CODING OF INTERNAL STATES OF SEQUENTIAL MACHINES"
IEEE TRANS. ELECTRON. COMPUT., VOL.EC-13, PP.549-562 OCTOBER 1964

DORMIDO B, CANTO D
"SYSTEMATIC SYNTHESIS OF COMBINATIONAL CIRCUITS USING MULTIPLEXERS"
ELECTRONICS LETTERS, VOL.14, NO.18, PP.588-590 31 AUGUST 1978

DRUFFEL L E
SEE: SCHMIDT D C, DRUFFEL L E

DU M -W
"A WAY TO FIND A LOWER BOUND FOR THE MINIMAL SOLUTION OF THE COVERING PROBLEM"
IEEE TRANS. COMPUT., VOL.C-21, PP.317-318 MARCH 1972

DUNHAM B, FRIDSHAL R
"THE PROBLEM OF SIMPLIFYING LOGICAL EXPRESSIONS"
J. SYMBOLIC LOGIC, VOL.24 PP.17-19 MARCH 1959

DUNWORTH A
"A RECURSIVE PREPROCESSING ALGORITHM FOR PRIME IMPLICANT DETERMINATION"
DIGITAL PROCESSES, VOL.3, NO.3, PP.259-267 OCTOBER 1977

DUNWORTH A
PERSONAL DISCUSSIONS
4 APRIL 1978

DUNWORTH A, HELLESTRAND G R
"THE USE OF APL IN DIGITAL SYSTEMS COURSES AT THE UNIVERSITY OF NEW SOUTH WALES"
IEEE TRANS. EDUCATION, VOL.E-20, PP.45-51 FEBRUARY 1977

DUNWORTH A, VAN DER KNAAP A
"AN EFFICIENT IMPLEMENTATION OF THE SHARP PRODUCT OPERATION FOR MULTIPLE-OUTPUT SWITCHING CUBES"
DIGITAL PROCESSES, VOL.3, NO.2, PP.161-176 JULY 1977

DWYER T F
SEE: WEINER P, DWYER T F

ELLIS D T
"A SYNTHESIS OF COMBINATORIAL LOGIC WITH NAND OR NOR ELEMENTS"
IEEE TRANS. ELECTRON. COMPUT., VOL.EC-14, PP.701-705 OCTOBER 1965

EWING A C, ROTH J P, WAGNER E G
"ALGORITHMS FOR LOGICAL DESIGN"
AIEE TRANS. COMM. ELECTRON., VOL.80(1), PP.450-458 SEPTEMBER 1961

FRIDSHAL R
SEE: DUNHAM B, FRIDSHAL R

FRIDSHAL R
SEE: COBHAM A, FRIDSHAL R, NORTH J H

FRIDSHAL R
SEE: COBHAM A, FRIDSHAL R, NORTH J H

GAVRILOV M A
"MINIMIZATION OF THE BOOLEAN FUNCTIONS CHARACTERIZING SWITCHING CIRCUITS"
AVTOMATIKA I TELEMEKHANIKA (AUTOMATION AND REMOTE CONTROL), VOL.20(2), PP.1188-1207 SEPTEMBER 1959

GEOFFRION A M
"AN IMPROVED IMPLICIT ENUMERATION APPROACH FOR INTEGER PROGRAMMING"
OPERATIONS RESEARCH, VOL.17, PP.437-454 MAY 1969

GHAZALA M J
"IRREDUNDANT DISJUNCTIVE AND CONJUNCTIVE FORMS OF A BOOLEAN FUNCTION"
IBM J. OF RES. AND DEV., VOL.1, PP.171-176 1957

GIATANIS N
SEE: HALATSIS C, GIATANIS N

GIMPEL J F
"A METHOD OF PRODUCING A BOOLEAN FUNCTION HAVING AN ARBITRARILY PRESCRIBED PRIME IMPLICANT TABLE"
IEEE TRANS. ELECTRON. COMPUT., VOL.EC-14, PP.485-488 JUNE 1965

GIMPEL J F
"A REDUCTION TECHNIQUE FOR PRIME IMPLICANT TABLES"
IEEE TRANS. ELECTRON. COMPUT., VOL.EC-14, PP.535-541 AUGUST 1965

GIMPEL J F
"THE MINIMIZATION OF TANT NETWORKS"
IEEE TRANS. ELECTRON. COMPUT., VOL.EC-16, PP.18-38 FEBRUARY 1967

GIVONE D D
"INTRODUCTION TO SWITCHING CIRCUIT THEORY"
NEW YORK: MCGRAW-HILL 1970

GOLOMB S W, BAUMERT L D
"BACKTRACK PROGRAMMING"
J. ASSOC. COMPUT. MACH., VOL.12, PP.516-524 OCTOBER 1965

GOMORY R E
"OUTLINE OF AN ALGORITHM FOR INTEGER SOLUTIONS TO LINEAR PROGRAMS"
BULL. AMER. MATH. SOC., VOL.64, PP.275-278 SEPTEMBER 1958

GORDON B B, HOUSE R W, LECHLER A P, NELSON L D, RADO T
"SIMPLIFICATION OF THE COVERING PROBLEM FOR MULTIPLE OUTPUT LOGICAL NETWORKS"
IEEE TRANS. ELECTRON. COMPUT., VOL.EC-15, PP.891-897 DECEMBER 1966

GORMAN D F, ANDERSON J P
"A LOGIC DESIGN TRANSLATION"
PROC. FALL JOINT COMPUTER CONF., PP.251-261 1962

GRASSELLI A, LUCCIO F
"A METHOD FOR MINIMIZING THE NUMBER OF INTERNAL STATES IN INCOMPLETELY SPECIFIED SEQUENTIAL NETWORKS"
IEEE TRANS. ELECTRON. COMPUT., VOL.EC-14, PP.350-359 JUNE 1965

GRASSELLI A, LUCCIO F
"SOME COVERING PROBLEMS IN SWITCHING THEORY"
NATO ADVANCED STUDY INSTITUTE ON NETWORK AND SWITCHING THEORY, VOL.1, PP.536-559 1968

GREER C R, THOMPSON R A
"COMBINATIONAL LOGIC DESIGN WITH DECODERS"
IEEE TRANS. COMPUT., VOL.C-27, PP.869-875 SEPTEMBER 1978

HALATSIS C, GIATANIS N
"IRREDUNDANT NORMAL FORMS AND MINIMAL DEPENDENCE SETS OF A BOOLEAN FUNCTION"
IEEE TRANS. COMPUT., VOL.C-27, PP.1064-1068 NOVEMBER 1978

HARRIS B
"AN ALGORITHM FOR DETERMINING MINIMAL REPRESENTATIONS OF A LOGIC FUNCTION"
IRE TRANS. ELECTRON. COMPUT., VOL.EC-6 PP.103-108 JUNE 1957

HARRISON H J
SEE: STORY J R, HARRISON H J, REINHARD E A

HARRISON M A
"INTRODUCTION TO SWITCHING AND AUTOMATA THEORY"
NEW YORK MCGRAW-HILL 1965

HELLERMAN L
"A CATALOG OF THREE-VARIABLE OR-INVERT AND AND-INVERT LOGICAL CIRCUITS"
IEEE TRANS. ELECTRON. COMPUT., VOL.EC-12, PP.198-223 JUNE 1963

HELLESTRAND G R
SEE: DUNWORTH A, HELLESTRAND G R

HILL F J, PETERSON G R
"INTRODUCTION TO SWITCHING THEORY AND LOGICAL DESIGN"
WILEY 1968

HOLMAN D F
SEE: BARNARD D F, HOLMAN D F

HONG S J
SEE: OSTAPKO D L, HONG S J

HOUSE R W
SEE: GORDON B B, HOUSE R W, LECHLER A P, NELSON L D, RADO T

HOUSE R W
SEE: ROBINSON S U, HOUSE R W

HOUSE R W, RADO T
"IMPLEMENTATION OF LOGIC"
IRE TRANS. ON MILITARY ELECTRONICS, VOL.MIL-6, NO.3, PP.297-302 JULY 1962

HOUSE R W, RADO T
"ON A COMPUTER PROGRAM FOR OBTAINING IRREDUCIBLE REPRESENTATIONS FOR TWO-LEVEL MULTIPLE INPUT-OUTPUT LOGICAL SYSTEMS"
J. ASSOC. COMPUT. MACH., VOL.10, PP.48-77 JANUARY 1963

HOUSE R W, STEVENS D W
"A NEW RULE FOR REDUCING CC TABLES"
IEEE TRANS. COMPUT., VOL.C-19, PP.1108-1111 NOVEMBER 1970

HUFFMAN D A
"THE SYNTHESIS OF SEQUENTIAL SWITCHING CIRCUITS"
J. FRANKLIN INST., VOL.257 NOS.3,4 PP.161,275 1954

HULINA P T
SEE: BREDESON J G, HULINA P T

HUMPHREY W S
"SWITCHING CIRCUITS WITH COMPUTER APPLICATIONS"
MCGRAW-HILL, NEW YORK 1958

HWA H R
"A METHOD FOR GENERATING PRIME IMPLICANTS OF A BOOLEAN EXPRESSION"
IEEE TRANS. COMPUT., VOL.C-23 PP.637-641 JUNE 1974

IBARAKI T
SEE: MUROGA S, IBARAKI T

IBARAKI T, MUROGA S
"SYNTHESIS OF NETWORKS WITH A MINIMUM NUMBER OF NEGATIVE GATES"
IEEE TRANS. COMPUT., VOL.C-20(1), PP.49-58 JANUARY 1971

KARNAUGH M
"THE MAP METHOD FOR SYNTHESIS OF COMBINATIONAL LOGIC CIRCUITS"
AIEE TRANS. COMM. ELECTRON., VOL.72, PP.593-598 NOVEMBER 1953

KARP R M
SEE: ROTH J P, KARP R M

KARP R M
"FUNCTIONAL DECOMPOSITION OF SWITCHING CIRCUIT DESIGN"
J. SOC. INDUSTRIAL AND APPLIED MATHS., VOL.11, PP.291-335 JUNE 1963

KARP R M, MCFARLIN F E, ROTH J P, WILTS J R
"A COMPUTER PROGRAM FOR THE SYNTHESIS OF COMBINATIONAL SWITCHING CIRCUITS"
PROC. 2ND ANNUAL SYMPOSIUM ON SWITCHING CIRCUIT THEORY AND LOGICAL DESIGN, PP.182-194 OCTOBER 1961

KASAMI T
SEE: NAKAMURA K, TOKURA N, KASAMI T

KAZAKOV V D
"THE MINIMIZATION OF LOGICAL FUNCTIONS OF A LARGE NUMBER OF VARIABLES"
AVTOMATIKA I TELEMEKHANIKA (AUTOM. REMOTE CONTROL), VOL.23 PP.1161-1165 MARCH 1963

KELLA J
"STATE MINIMIZATION OF INCOMPLETELY SPECIFIED SEQUENTIAL MACHINES"
IEEE TRANS. COMPUT., VOL.C-19, P.342-348 APRIL 1970

KIM J, NEWBORN M M
"THE SIMPLIFICATION OF SEQUENTIAL MACHINES WITH INPUT RESTRICTIONS"
IEEE TRANS. COMPUT., VOL.C-21, PP.1440-1443 DECEMBER 1972

KIME C R
SEE: BATNI R P, RUSSELL J D, KIME C R

KJELKERUD E
"A COMPUTER PROGRAM FOR THE SYNTHESIS OF SWITCHING CIRCUITS BY DECOMPOSITION"
IEEE TRANS. COMPUT., VOL.C-21, PP.568-573 JUNE 1972

KNUTH D E
"THE ART OF COMPUTER PROGRAMMING"
ADDISON-WESLEY 1973

KRIEGER M
"BASIC SWITCHING CIRCUIT THEORY"
MACMILLAN 1967

LACROSSE T R
SEE: DEFRANCESCO H F, LACROSSE T R

LANGDON G G
"A DECOMPOSITION CHART TECHNIQUE TO AID IN REALIZATIONS WITH MULTIPLEXERS"
IEEE TRANS. COMPUT., VOL.C-27, PP.157-159 FEBRUARY 1978

LAWLER E L
"AN APPROACH TO MULTILEVEL BOOLEAN MINIMIZATION"
J. ASSOC. COMPUT. MACH., VOL.11, PP.283-295 JULY 1964

LAWLER E L
"COVERING PROBLEMS: DUALITY RELATIONS AND A NEW METHOD OF SOLUTION"
J. SOC. FOR INDUSTRIAL AND APPLIED MATHS., VOL.14, PP.1115-1132 SEPTEMBER 1966

LEBOW I L
SEE: BARTEE T C, LEBOW I L, REED I S

LECHLER A P
SEE: GORDON B B, HOUSE R W, LECHLER A P, NELSON L D, RADO T

LEE C Y
"SWITCHING FUNCTIONS ON AN N-DIMENSIONAL CUBE"
AIEE TRANS. COMM. ELECTR., VOL.73(1), PP.289-291 SEPTEMBER 1954

LEE H -P S
"AN ALGORITHM FOR MINIMAL TANT NETWORK GENERATION"
IEEE TRANS. COMPUT., VOL.C-27, PP.1202-1206 DECEMBER 1978

LEE H -P, DAVIDSON E S
"A TRANSFORM FOR NAND NETWORK DESIGN"
IEEE TRANS. COMPUT., VOL.C-21, PP.12-20 JANUARY 1972

LEE R C T
SEE: SLAGLE J R, CHANG C -L, LEE R C T

LEE R C T
"AN ALGORITHM TO GENERATE PRIME IMPLICANTS AND ITS APPLICATION TO THE SELECTION PROBLEM"
INFORMATION SCIENCES, VOL.4, PP.251 1972

LEWIN D W
"LOGICAL DESIGN OF SWITCHING CIRCUITS"
LONDON:NELSON, 1968

LEWIN D W, WATERS M C
"COMPUTER AIDS TO LOGIC SYSTEM DESIGN"
COMPUTER BULLETIN, VOL.13, PP.382-380 NOVEMBER 1969

LOGAN J R
"LOGICAL DETERMINISM IN NETWORK SYNTHESIS"
WRIGHT-PATTERSON AIR FORCE BASE, AFAL-TR-65-72 1965

LOGAN J R
"A DETERMINISTIC NETWORK SYNTHESIZER"
PROC. NATIONAL ELECTRONICS CONF., VOL.25, PP.645-650 DECEMBER 1969

LOGAN J R, LUCAS P, WILKENING C
"DETERMINISTIC NETWORK SYNTHESIS AND LARGE SCALE INTEGRATION"
WRIGHT-PATTERSON AIR FORCE BASE, AFAL-TR-68-13 FEBRUARY 1969

LUCAS P
SEE: LOGAN J R, LUCAS P, WILKENING C

LUCCIO F
SEE: GRASSELLI A, LUCCIO F

LUCCIO F
"A METHOD FOR THE SELECTION OF PRIME IMPLICANTS"
IEEE TRANS. ELECTRON. COMPUT., VOL.EC-15, PP.205-212 APRIL 1966

LUCCIO F
SEE: GRASSELLI A, LUCCIO F

LUCCIO F
"EXTENDING THE DEFINITION OF PRIME COMPATIBILITY CLASSES OF STATES IN INCOMPLETE SEQUENTIAL MACHINE REDUCTION"
IEEE TRANS. COMPUT., VOL.C-18, PP.537-540 JUNE 1969

MAGE J M
"APPLICATION OF ITERATIVE CONSENSUS TO MULTIPLE-OUTPUT FUNCTIONS"
IEEE TRANS. COMPUT., VOL.C-19, PP.359-359 APRIL 1970

MARCUS M P
"SWITCHING CIRCUITS FOR ENGINEERS"
2ND ED. ENGLEWOOD CLIFF, N.J.: PRENTICE HALL, 1967

MARIN M A
SEE: CERNY E, MARIN M A

MCCLUSKEY E J
"MINIMIZATION OF BOOLEAN FUNCTIONS"
BELL SYST. TECH. J., VOL.35(2), PP.1417-1444 NOVEMBER 1956

MCCLUSKEY E J
SEE: PYNE I B, MCCLUSKEY E J

MCCLUSKEY E J (ED)
"A SURVEY OF SWITCHING CIRCUIT THEORY"
MCGRAW-HILL 1962

MCCLUSKEY E J
SEE: PYNE I B, MCCLUSKEY E J

MCCLUSKEY E J
"MINIMAL SUMS FOR BOOLEAN FUNCTIONS HAVING MANY UNSPECIFIED FUNDAMENTAL PRODUCTS"
AIEE TRANS. COMM. ELECTRON., VOL.81(1), PP.387-392 NOVEMBER 1962

MCCLUSKEY E J
SEE: DOLOTTA T A, MCCLUSKEY E J

MCCLUSKEY E J
"INTRODUCTION TO THE THEORY OF SWITCHING CIRCUITS"
MCGRAW-HILL 1965

MCCLUSKEY E J, SCHORR H
"ESSENTIAL MULTIPLE-OUTPUT PRIME IMPLICANTS"
PROC. SYMPOSIUM ON MATHEMATICAL THEORY OF AUTOMATA, VOL.12, PP.437-458 APRIL 1962

MCFARLIN F E
SEE: KARP R M, MCFARLIN F E, ROTH J P, WILTS J R

MCKELLAR A C
SEE: SHEN V Y -S, MCKELLAR A C

MCKINNEY M H
SEE: RHYNE V T, NOE P S, MCKINNEY M H, POOCH U W

MCMAHON M T
SEE: WAXMAN R, MCMAHON M T, CRAWFORD B J, DE ANDRADE A B

MCNAUGHTON R, MITCHELL B
"THE MINIMALITY OF RECTIFIER NETS WITH MULTIPLE OUTPUTS INCOMPLETELY SPECIFIED"
J. FRANKLIN INST., VOL.264, PP.457-460 DECEMBER 1957

MCVEY E S
SEE: BOWMAN R M, MCVEY E S

MEO A R
"ON THE MINIMAL THIRD ORDER EXPRESSION OF A BOOLEAN FUNCTION"
AIEE SYMP. ON SWITCHING CIRCUIT THEORY AND LOGICAL DESIGN, CHICAGO SEPTEMBER 1962

MEO A R
"ON THE SYNTHESIS OF MANY-VARIABLE SWITCHING FUNCTIONS"
NATO ADVANCED STUDY INSTITUTE ON NETWORK AND SWITCHING THEORY, VOL.1, PP.470-482 1968

MERWIN R E, SANBORN J L
"DIGITAL COMPUTERS FOR LOGICAL DESIGNS"
IN KLERER N, KORN G (EDS) 'DIGITAL COMPUTER USER'S HANDBOOK' NEW YORK, MCGRAW-HILL 1967

METZE G
SEE: DAVIDSON E S, METZE G

METZE G
SEE: DAVIDSON E S, METZE G

MILETO F, PUTZOLU G
"AVERAGE VALUES OF QUANTITIES APPEARING IN BOOLEAN FUNCTION MINIMIZATION"
IEEE TRANS. ELECTRON. COMPUT., VOL.EC-13, PP.87-92 APRIL 1964

MILETO F, PUTZOLU G
"STATISTICAL COMPLEXITY OF ALGORITHMS FOR BOOLEAN FUNCTION MINIMIZATION"
J. ASSOC. COMPUT. MACH., VOL.12, PP.364-375 JULY 1965

MILETO F, PUTZOLU G
"AVERAGE VALUES OF QUANTITIES APPEARING IN MULTIPLE OUTPUT BOOLEAN MINIMIZATION"
IEEE TRANS. ELECTRON. COMPUT., VOL.EC-14, PP.542-552 AUGUST 1965

MILLER R E
"SWITCHING THEORY. VOLUME 1: COMBINATIONAL CIRCUITS"
NEW YORK: WILEY, 1965

MILLS B E
SEE: SAMSON E W, MILLS B E

MITCHELL B
SEE: MCNAUGHTON R, MITCHELL B

MOORE E F (ED)
"SEQUENTIAL MACHINES: SELECTED PAPERS"
ADDISON-WESLEY 1964

MORREALE E
"PARTITIONED LIST ALGORITHMS FOR PRIME IMPLICANT DETERMINATION FROM CANONICAL FORMS"
IEEE TRANS. ELECTRON. COMPUT., VOL.EC-16, PP.611-620 OCTOBER 1967

MORREALE E
"PARTITIONED LIST TECHNIQUES FOR THE SYNTHESIS BY COMPUTER OF TWO-LEVEL AND-OR NETWORKS"
PROC. 23RD NAT. CONF. ASSOC. COMPUT. MACH., PP.355-365 1968

MORREALE E
"PARTITIONED LIST TECHNIQUES IN QUINE'S METHOD IMPLEMENTATION"
NATO ADVANCED STUDY INSTITUTE ON NETWORK AND SWITCHING THEORY, VOL.1, PP.483-495 1968

MORREALE E
"COMPUTATIONAL COMPLEXITY OF PARTITIONED LIST ALGORITHMS"
IEEE TRANS. COMPUT., VOL.C-19, PP.421-428 MAY 1970

MORREALE E
"RECURSIVE OPERATORS FOR PRIME IMPLICANT AND IRREDUNDANT NORMAL FORM DETERMINATION"
IEEE TRANS. COMPUT., VOL.C-19, PP.504-509 JUNE 1970

MOTT T H
"DETERMINATION OF THE IRREDUNDANT NORMAL FORMS OF A TRUTH FUNCTION BY ITERATED CONSENSUS OF THE PRIME IMPLICANTS"
IRE TRANS. ELECTRON. COMPUT., VOL.EC-9, PP.245-252 JUNE 1960

MOTT T H
SEE: CHANG D M Y, MOTT T H

MUELLER R K
"ON THE SYNTHESIS OF A MINIMAL REPRESENTATION OF A LOGIC FUNCTION"
AIR FORCE CAMBRIDGE RESEARCH CENTRE, TR-55-104 APRIL 1955

MUELLER R K
SEE: URBANO R H, MUELLER R K

MUKHOPADHYAY A
"DETECTION OF DISJUNCTS OF SWITCHING FUNCTIONS AND MULTI-LEVEL CIRCUIT DESIGN"
J. OF ELECTRONICS AND CONTROL, VOL.10, PP.45-55 JANUARY 1961

MULLER D E
"APPLICATION OF BOOLEAN ALGEBRA TO SWITCHING CIRCUIT DESIGN AND TO ERROR DETECTION"
IRE TRANS. ELECTRON. COMPUT., VOL.EC-3, PP.6-12 SEPTEMBER 1954

MULLER D E
"COMPLEXITY IN ELECTRONIC SWITCHING CIRCUITS"
IRE TRANS. ELECTRON. COMPUT., VOL.EC-5, PP.15-19 MARCH 1956

MUROGA S
SEE: IBARAKI T, MUROGA S

MUROGA S, IBARAKI T
"DESIGN OF OPTIMAL SWITCHING NETWORKS BY INTEGER PROGRAMMING"
IEEE TRANS. COMPUT., VOL.C-21, PP.573-582 JUNE 1972

NAGLE H T
SEE: SHIVA S G, NAGLE H T

NAGLE H T
SEE: SHIVA S G, NAGLE H T

NAGLE H T
SEE: SHIVA S G, NAGLE H T

NAGLE H T
PERSONAL CORRESPONDENCE
7 MARCH 1978

NAKAGAWA T
"A BRANCH-AND-BOUND ALGORITHM FOR OPTIMAL AND-OR NETWORKS"
DEPT. COMPUT. SCI., UNIV. ILLINOIS, URBANA, REP. 462 JUNE 1971

NAKAMURA K, TOKURA N, KASAMI T
"MINIMAL NEGATIVE GATE NETWORKS"
IEEE TRANS. COMPUT., VOL.C-21, PP.5-11 JANUARY 1972

NAM C -W
SEE: SU S Y H, NAM C -W

NECULA N N
"A NUMERICAL PROCEDURE FOR DETERMINATION OF THE PRIME IMPLICANTS OF A BOOLEAN FUNCTION"
IEEE TRANS. ELECTRON. COMPUT., VOL.EC-16, PP.687-689, OCTOBER 1967

NECULA N N
"AN ALGORITHM FOR THE AUTOMATIC APPROXIMATE MINIMISATION OF BOOLEAN FUNCTIONS"
IEEE TRANS. COMPUT., VOL.C-17(2), PP.770-782 AUGUST 1968

NELSON L D
SEE: GORDON B B, HOUSE R W, LECHLER A P, NELSON L D, RADO T

NELSON R J
"SIMPLEST NORMAL TRUTH FUNCTIONS"
J. SYMBOLIC LOGIC, VOL.20, PP.105-108 JUNE 1955

NELSON R J
"WEAK SIMPLEST NORMAL TRUTH FUNCTIONS"
J. SYMBOLIC LOGIC, VOL.20, PP.232-234 SEPTEMBER 1955

NEWBORN M M
SEE: KIM J, NEWBORN M M

NOE P S
LETTER ADDRESSED TO PROF. H T NAGLE
3 OCTOBER 1974

NOE P S
SEE: RHYNE V T, NOE P S, MCKINNEY M H, POOCH U W

NOE P S
PERSONAL CORRESPONDENCE
27 FEBRUARY 1978

NOE P S, RHYNE V T
"A MODIFICATION TO THE SHR-OPTIMAL STATE ASSIGNMENT PROCEDURE"
IEEE TRANS. COMPUT., VOL.C-23, PP.327-329 MARCH 1974

NOE P S, RHYNE V T, SURARATRUNGSI S
"COMMENTS ON 'WEIGHTING METHOD FOR THE DETERMINATION OF THE IRREDUNDANT SET OF PRIME IMPLICANTS'"
IEEE TRANS. COMPUT., VOL.C-23, PP.646-646 JUNE 1974

NORTH J H
SEE: COBHAM A, FRIDSHAL R, NORTH J H

NORTH J H
SEE: COBHAM A, FRIDSHAL R, NORTH J H

NORTH J H
SEE: COBHAM A, NORTH J H

OSTAPKO D L, HONG S J
"GENERATING TEST EXAMPLES FOR HEURISTIC BOOLEAN MINIMIZATION"
IBM J. OF RES. AND DEV., VOL.18, PP.459-464 SEPTEMBER 1974

PAULL M C, UNGER S H
"MINIMIZING THE NUMBER OF STATES IN INCOMPLETELY SPECIFIED SEQUENTIAL SWITCHING FUNCTIONS"
IEEE TRANS. ELECTRON. COMPUT., VOL.EC-8, PP.356-367 SEPTEMBER 1959

PERLMAN M
SEE: ROTH J P, PERLMAN M

PETERSON G R
SEE: HILL F J, PETERSON G R

PETRICK S R
"A DIRECT DETERMINATION OF THE IRREDUNDANT FORMS OF A BOOLEAN FUNCTION FROM THE SET OF PRIME IMPLICANTS"
AIR FORCE CAMBRIDGE RESEARCH CENTRE, TR-56-110 APRIL 1956

PHISTER M
"LOGICAL DESIGN OF DIGITAL COMPUTERS"
NEW YORK: WILEY, 1958

POOCH U W
SEE: RHYNE V T, NOE P S, MCKINNEY M H, POOCH U W

POVAROV G
"ON THE FUNCTIONAL DECOMPOSITION OF BOOLEAN FUNCTIONS"
DOKLADY AKADEMIIA NAUK SSSR, VOL.94, PP.801-803 1954

PRATHER R E
"COMPUTATIONAL AIDS FOR DETERMINING THE MINIMAL FORM OF A TRUTH FUNCTION"
J. ASSOC. COMPUT. MACH., VOL.7, PP.299-310 OCTOBER 1960

PRATHER R E
"INTRODUCTION TO SWITCHING THEORY: A MATHEMATICAL APPROACH"
BOSTON, MA: ALLYN AND BACON 1967

PUTZOLU G
SEE: MILETO F, PUTZOLU G

PUTZOLU G
SEE: MILETO F, PUTZOLU G

PUTZOLU G
SEE: MILETO F, PUTZOLU G

PYNE I B, MCCLUSKEY E J
"AN ESSAY ON PRIME IMPLICANT TABLES"
J. SOC. INDUSTRIAL AND APPLIED MATHS., VOL.9, PP.604-631 DECEMBER 1961

PYNE I B, MCCLUSKEY E J
"THE REDUCTION OF REDUNDANCY IN SOLVING PRIME IMPLICANT TABLES"
IRE TRANS. ELECTRON. COMPUT., VOL.EC-11, PP.473-482 AUGUST 1962

QUINE W V
"THE PROBLEM OF SIMPLIFYING TRUTH FUNCTIONS"
AMERICAN MATH. MONTHLY, VOL.59, PP.521-531 OCTOBER 1952

QUINE W V
"A WAY TO SIMPLIFY TRUTH FUNCTIONS"
AMERICAN MATH. MONTHLY, VOL.62, PP.627-631 NOVEMBER 1955

QUINE W V
"ON CORES OF PRIME IMPLICANTS OF TRUTH FUNCTIONS"
AMERICAN MATH. MONTHLY, VOL.66, PP.755-760 NOVEMBER 1959

RADO T
"ON THE PRESENCE FUNCTION OF GAZALE"
IBM J. OF RES. AND DEV., VOL.6, PP.268-269 APRIL 1962

RADO T
SEE: HOUSE R W, RADO T

RADO T
SEE: HOUSE R W, RADO T

RADO T
SEE: GORDON B B, HOUSE R W, LECHLER A P, NELSON L D, RADO T

REED I S
SEE: BARTEE T C, LEBOW I L, REED I S

REEVES C M
"AN INTRODUCTION TO LOGICAL DESIGN OF DIGITAL CIRCUITS"
CAMBRIDGE UNIVERSITY PRESS 1972

REINHARD E A
SEE: STORY J R, HARRISON H J, REINHARD E A

REUSCH B
"GENERATION OF PRIME IMPLICANTS FROM SUBFUNCTIONS AND A UNIFYING APPROACH TO THE COVERING PROBLEM"
IEEE TRANS. COMPUT., VOL.C-24, PP.924-930 SEPTEMBER 1975

RHYNE V T
SEE: NOE P S, RHYNE V T

RHYNE V T
SEE: NOE P S, RHYNE V T, SURARATRUNGSI S

RHYNE V T, NOE P S, MCKINNEY M H, POOCH U W
"A NEW TECHNIQUE FOR THE FAST MINIMIZATION OF SWITCHING FUNCTIONS"
IEEE TRANS. COMPUT., VOL.C-26, PP.757-764 AUGUST 1977

ROBINSON S U, HOUSE R W
"GIMPEL'S REDUCTION TECHNIQUE EXTENDED TO THE COVERING PROBLEM WITH COSTS"
IEEE TRANS. ELECTRON. COMPUT., VOL.EC-16, PP.509-514 AUGUST 1967

ROTH J P
"ALGEBRAIC TOPOLOGICAL METHODS FOR THE SYNTHESIS OF SWITCHING SYSTEMS.I."
TRANS. AMER. MATH. SOC., VOL.88, PP.301-326 JULY 1958

ROTH J P
"ALGEBRAIC TOPOLOGICAL METHODS IN SYNTHESIS"
ANNALS OF THE HARVARD COMPUTATION LABORATORY, VOL.29, PP.57-73 1959

ROTH J P
"MINIMIZATION OVER BOOLEAN TREES"
IBM J. OF RES. AND DEV., VOL.4, PP.543-558 NOVEMBER 1960

ROTH J P
SEE: EWING A C, ROTH J P, WAGNER E G

ROTH J P
SEE: KARP R M, MCFARLIN F E, ROTH J P, WILTS J R

ROTH J P
"SYSTEMATIC DESIGN OF AUTOMATA"
PROC. FALL JOINT COMPUTER CONF., PP.1093-1100 1965

ROTH J P
"METHODS IN DESIGN OPTIMIZATION-1. ARRAY LOGICS"
IBM RESEARCH REPORT RC-4943 JULY 1974

ROTH J P
"PROGRAMMED LOGIC ARRAY OPTIMIZATION"
IEEE TRANS. COMPUT., VOL.C-27, PP.174-176 FEBRUARY 1978

ROTH J P
PERSONAL CORRESPONDENCE
13 SEPTEMBER 1978

ROTH J P, KARP R M
"MINIMIZATION OVER BOOLEAN GRAPHS"
IBM J. OF RES. AND DEV., VOL.6, PP.227-238 APRIL 1962

ROTH J P, PERLMAN M
"SPACE APPLICATIONS OF A MINIMIZATION ALGORITHM"
IEEE TRANS. AEROSPACE AND ELECTRONIC SYSTEMS, VOL.AES-5,
PP.703-711 SEPTEMBER 1969

ROTH J P, WAGNER E G
"ALGEBRAIC TOPOLOGICAL METHODS FOR THE SYNTHESIS OF
SWITCHING SYSTEMS.III. MINIMIZATION OF NONSINGULAR
BOOLEAN TREES"
IBM J. OF RES. AND DEV., VOL.3, PP.326-344 OCTOBER 1959

ROTH J P, WAGNER E G
"A CALCULUS AND AN ALGORITHM FOR A LOGIC MINIMIZATION
PROBLEM TOGETHER WITH AN ALGORITHMIC NOTATION"
IBM RESEARCH REPORT, RC-2280 NOVEMBER 1968

ROTH R
"COMPUTER SOLUTIONS TO MINIMUM-COVER PROBLEMS"
OPERATIONS RESEARCH, VOL.17, NO.3, PP.455-465 MAY 1969

ROY P K S, SHENG C L
"A DECOMPOSITION METHOD OF DETERMINING MAXIMUM
COMPATIBLES"
IEEE TRANS. COMPUT., VOL.C-21, PP.309-312 MARCH 1972

RUSSELL J D
SEE: BATNI R P, RUSSELL J D, KIME C R

SAMSON E W, MILLS B E
"CIRCUIT MINIMIZATION: ALGEBRA AND ALGORITHMS FOR NEW
BOOLEAN CANONICAL EXPANSIONS"
AIR FORCE CAMBRIDGE RESEARCH CENTRE TECHNICAL REPORT 54-21
APRIL 1954

SANBORN J L
SEE: MERWIN R E, SANBORN J L

SCHEINMAN A H
"A METHOD FOR SIMPLIFYING BOOLEAN FUNCTIONS"
BELL SYST. TECH. J., VOL.41(2), PP.1337-1346 JULY 1962

SCHMIDT D C, DRUFFEL L E
"AN EXTENSION OF THE CLAUSE TABLE APPROACH TO MULTI-OUTPUT COMBINATIONAL SWITCHING NETWORKS"
IEEE TRANS. COMPUT., VOL.C-23, PP.338-346 APRIL 1974

SCHNEIDER P R
SEE: DIETMEYER D L, SCHNEIDER P R

SCHNEIDER P R
SEE: DIETMEYER D L, SCHNEIDER P R

SCHNEIDER P R, DIETMEYER D L
"AN ALGORITHM FOR SYNTHESIS OF MULTIPLE OUTPUT COMBINATIONAL LOGIC"
IEEE TRANS. COMPUT., VOL.C-17, PP.117-128 FEBRUARY 1968

SCHORR H
SEE: MCCLUSKEY E J, SCHORR H

SHANNON C E
"A SYMBOLIC ANALYSIS OF RELAY AND SWITCHING CIRCUITS"
TRANS. AIEE, VOL.57, PP.713-723 1938

SHANNON C E
"SYNTHESIS OF TWO-TERMINAL SWITCHING CIRCUITS"
BELL SYST. TECH. J., VOL.28, PP.59-98 1949

SHEN V Y -S, MCKELLAR A C
"AN ALGORITHM FOR THE DISJUNCTIVE DECOMPOSITION OF SWITHING FUNCTIONS"
IEEE TRANS. COMPUT., VOL.C-19, PP.239-248 MARCH 1970

SHENG C L
SEE: ROY P K S, SHENG C L

SHIVA S G, NAGLE H T
"BYPASS MULTIVARIABLE KARNAUGH MAPS"
ELECTRONIC DESIGN, VOL.22, NO.21, PP.86-91, 11TH OCTOBER 1974

SHIVA S G, NAGLE H T
"REDUCE STATE TABLES BY COMPUTER"
ELECTRONIC DESIGN, VOL.22, NO.22, PP.122-127, 25TH OCTOBER 1974

SHIVA S G, NAGLE H T
"LET A COMPUTER DESIGN MEMORY CIRCUITS"
ELECTRONIC DESIGN, VOL.22, NO.23, PP.122-127, 8TH NOVEMBER 1974

SLAGLE J R, CHANG C -L, LEE R C T
"A NEW ALGORITHM FOR GENERATING PRIME IMPLICANTS"
IEEE TRANS. COMPUT., VOL.C-19, PP.304-310 APRIL 1970

SMITH E J
SEE: WEINER P, SMITH E J

SMITH R A
"MINIMAL THREE-VARIABLE NOR AND NAND LOGIC CIRCUITS"
IEEE TRANS. ELECTRON. COMPUT., VOL.EC-14, PP.79-81 FEBRUARY 1965

STEVENS D W
SEE: HOUSE R W, STEVENS D W

STOFFERS K E
"PARTITIONING OF SEPARATING EDGES: A NEW APPROACH TO COMBINATIONAL LOGIC DESIGN"
IEEE TRANS. COMPUT., VOL.C-26, PP.833-836 AUGUST 1977

STORY J R, HARRISON H J, REINHARD E A
"OPTIMUM STATE ASSIGNMENT FOR SYNCHRONOUS SEQUENTIAL CIRCUITS"
IEEE TRANS. COMPUT., VOL.C-21, PP.1365-1373 DECEMBER 1972

SU
"LOGICAL DESIGN AND ITS RECENT DEVELOPMENT"
COMPUTER DESIGN, PP.81 FEBRUARY 1974

SU S Y H, NAM C -W
"COMPUTER-AIDED SYNTHESIS OF MULTIPLE-OUTPUT MULTILEVEL NAND NETWORKS"
IEEE TRANS. COMPUT., VOL.C-20(2), PP.1445-1455 DECEMBER 1971

SU Y -H
SEE: DIETMEYER D L, SU Y -H

SU Y -H
"AUTOMATED LOGIC DESIGN VIA A NEW LOCAL EXTRACTION ALGORITHM"
PROC. NATIONAL ELECTRONICS CONF., VOL.25, PP.651-656 DECEMBER 1969

SU Y -H, DIETMEYER D L
"COMPUTER REDUCTION OF TWO-LEVEL, MULTIPLE-OUTPUT SWITCHING CIRCUITS"
IEEE TRANS. COMPUT., VOL.C-18, PP.58-63 JANUARY 1969

SURARATRUNGSI S
SEE: NOE P S, RHYNE V T, SURARATRUNGSI S

SURESHCHANDER
"MINIMIZATION OF SWITCHING FUNCTIONS - A FAST TECHNIQUE"
IEEE TRANS. COMPUT., VOL.C-24, PP.753-756, JULY 1975

SVOBODA A
"AN ALGORITHM FOR SOLVING BOOLEAN EQUATIONS"
IEEE TRANS. ELECTRON. COMPUT., VOL.EC-12, PP.557-559 OCTOBER 1963

SVOBODA A
"ORDERING OF IMPLICANTS"
IEEE TRANS. ELECTRON. COMPUT., VOL.EC-16, PP.100-105 FEBRUARY 1967

TAN C -J
"STATE ASSIGNMENTS FOR ASYNCHRONOUS SEQUENTIAL MACHINES"
IEEE TRANS. COMPUT., VOL.C-20(1) PP.382-391 APRIL 1971

THOMPSON R A
SEE: GREER C R, THOMPSON R A

TISON P
"GENERALIZATION OF CONSENSUS THEORY AND APPLICATION TO THE MINIMIZATION OF BOOLEAN FUNCTIONS"
IEEE TRANS. COMPUT. VOL.EC-16, PP.446-456 AUGUST 1967

TOKURA N
SEE: NAKAMURA K, TOKURA N, KASAMI T

UNGER S H
SEE: PAULL M C, UNGER S H

UNGER S H
"INTRODUCTION TO THE THEORY OF SEQUENTIAL SWITCHING CIRCUITS"
NATO ADVANCED STUDY INSTITUTE ON NETWORK AND SWITCHING THEORY, VOL.1 PP.521-535 1968

UNGER S H
"ASYNCHRONOUS SEQUENTIAL SWITCHING CIRCUITS"
WILEY 1969

URBANO R H, MUELLER R K
"A TOPOLOGICAL METHOD FOR THE DETERMINATION OF THE MINIMAL FORMS OF A BOOLEAN FUNCTION"
IRE TRANS. ELECTRON. COMPUT., VOL.EC-5, PP.126-132 SEPTEMBER 1956

VAN BOSSE J G
SEE: BURKE R E, VAN BOSSE J G

VAN DEN DOLDER B
SEE: VINK H A, VAN DEN DOLDER B, AL J

VAN DER KNAAP A
SEE: DUNWORTH A, VAN DER KNAAP A

VANDLING G C
"THE SIMPLIFICATION OF MULTIPLE-OUTPUT SWITCHING NETWORKS COMPOSED OF UNILATERAL DEVICES"
IRE TRANS. ELECTRON. COMPUT., VOL.EC-9, PP.477-486 DECEMBER 1960

VEITCH E W
"A CHART METHOD FOR SIMPLIFYING TRUTH FUNCTIONS"
PROC. ASSOC. COMPUT. MACH., PP.127-133 1952

VINK H A
"MINIMAL TANT NETWORKS OF FUNCTIONS WITH DONT CARE'S AND SOME COMPLEMENTED INPUT VARIABLES"
IEEE TRANS. COMPUT., VOL.C-27, PP.1073-1078 NOVEMBER 1978

VINK H A, VAN DEN DOLDER B, AL J
"REDUCTION OF CC-TABLES USING MULTIPLE IMPLICATION"
IEEE TRANS. COMPUT., VOL.C-27, PP.961-966 OCTOBER 1978

WAGNER E G
SEE: ROTH J P, WAGNER E G

WAGNER E G
SEE: EWING A C, ROTH J P, WAGNER E G

WAGNER E G
SEE: ROTH J P, WAGNER E G

WARFIELD J N
SEE: BUTLER K J, WARFIELD J N

WATERS M C
SEE: LEWIN D W, WATERS M C

WAXMAN R, MCMAHON M T, CRAWFORD B J, DE ANDRADE A B
"AUTOMATED LOGIC DESIGN TECHNIQUES APPLICABLE TO INTEGRATED CIRCUIT TECHNOLOGY"
PROC. FALL JOINT COMPUTER CONFERENCE, VOL.29, PP.247-265 1966

WEINER P, DOLOTTA T A
"MIXED MEMORY TYPE REALIZATIONS OF SEQUENTIAL MACHINES"
IEEE TRANS. COMPUT., VOL.C-18, PP.272-277 MARCH 1969

WEINER P, DWYER T F
"DISCUSSION OF SOME FLAWS IN THE CLASSICAL THEORY OF TWO-LEVEL MINIMIZATION OF MULTPLE-OUTPUT SWITCHING NETWORKS"
IEEE TRANS. COMPUT., VOL.C-17(1), PP.184-186 FEBRUARY 1968

WEINER P, SMITH E J
"OPTIMIZATION OF REDUCED DEPENDENCIES FOR SYNCHRONOUS SEQUENTIAL MACHINES"
IEEE TRANS. COMPUT., VOL.EC-16, PP.835-847 DECEMBER 1967

WILKENING C
SEE: LOGAN J R, LUCAS P, WILKENING C

WILLIAMS H P
"THE SYNTHESIS OF LOGICAL NETS CONSISTING OF NOR UNITS"
THE COMPUTER JOURNAL, VOL.11, PP.173-176 AUGUST 1968

WILTS J R
SEE: KARP R M, MCFARLIN F E, ROTH J P, WILTS J R