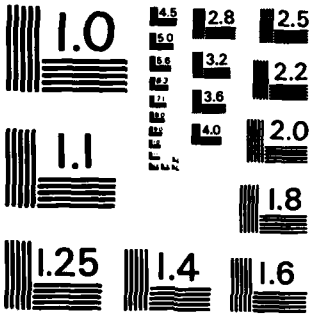
AD-A148 465 COMPLEXITY TESTABILITY AND FAULT ANALYSIS OF DIGITAL ANALOG AND HYBRID SY..(U) TENNESSEE UNIV KNOXVILLE DEPT OF ELECTRICAL ENGINEERING R C GONZALEZ ET AL. 1/2

UNCLASSIFIED 30 SEP 84 TR-EE-84-50 N00014-78-C-0311 F/G 12/1 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS - 1963 - A

COMPLEXITY, TESTABILITY, AND FAULT ANALYSIS OF DIGITAL, ANALOG, AND HYBRID SYSTEMS

Final Report

Prepared for the

Office of Naval Reseach
Arlington, Va. 22217

Contract No. N00014-78-C-0311

September 30, 1984

TR-EE-84-50

COMPLEXITY, TESTABILITY, AND FAULT ANALYSIS OF DIGITAL, ANALOG, AND HYBRID SYSTEMS

Final Report

Prepared for the

Office of Naval Reseach
Arlington, Va. 22217

Contract No. N00014-78-C-0311
September 30, 1984
TR-EE-84-50

Investigators:

R.C. Gonzalez, Electrical Engineering Dept., University of Tennessee, Knoxville, TN 37996

M.G. Thomason, Computer Science Dept., University of Tennessee, Knoxville, TN 37996

B.M.E. Moret, Computer Science Dept., University of New Mexico, Albuquerque, NM 87131

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER TR-EE-84-50	2. GOVT ACCESSION NO. AD-A148 465	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Complexity, Testability, and Fault Analysis of Digital, Analog, and Hybrid Systems		5. TYPE OF REPORT & PERIOD COVERED Final Report Feb 1, 1982 - June 30, 1984 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) R. C. Gonzalez, M. G. Thomason, B. M. E. Moret		8. CONTRACT OR GRANT NUMBER(s) N00014-78-C-0311
9. PERFORMING ORGANIZATION NAME AND ADDRESS Electrical Engineering Department University of Tennessee Knoxville, TN 37996		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research Arlington, VA 22217		12. REPORT DATE September 30, 1984 13. NUMBER OF PAGES 156
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same as 11		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same as 16

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Decision trees, decision diagrams, activity, boolean functions, fault trees, complexity, Mahalanobis distance, moments, semi-invariants, linear inequalities.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The research described in this report is divided into two major areas: (1) discrete mathematical descriptions of digital, analog, and hybrid systems, and (2) techniques for measuring parameters for characterizing certain aspects of such systems. New results are presented in decision trees, fault trees, testing complexity, optimal solution of linear inequalities, and the computation of moments and semi-invariants of the interclass Mahalanobis distance.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

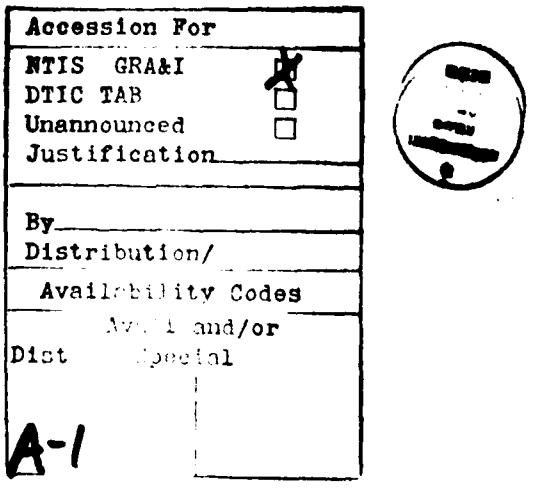
UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

TABLE OF CONTENTS

SECTION I

SUMMARY



SECTION I
SUMMARY

1. INTRODUCTION

This is the final report on the work in "complexity, testability, and fault analysis of digital, analog, and hybrid systems" carried out on ONR Contract N00014-78-C-0311. This work was performed by Drs. R. C. Gonzalez and M. G. Thomason at the University of Tennessee, Knoxville, and by Dr. B.M.E. Moret initially at the University of Tennessee, Knoxville, and subsequently at the University of New Mexico, Albuquerque. Other individuals were also involved for short periods of time. The research has produced significant theoretical and practical results which have appeared in technical journals and technical reports.

The research was divided into two major areas: discrete mathematical descriptions of aspects of digital, analog, and hybrid systems useful in the study of complexity and fault analysis; and techniques for measuring parameters to characterize certain aspects of such systems. In this summary section we give an overview of the various results. Sections II through VII contain compilations of the articles and reports resulting from this work. The material in these sections is organized in the same order as the discussion in this summary section.

2. DECISION TREES

Early in this work, decision trees and equivalent expressions were adopted as the discrete mathematical representation of functions for detailed study. There is a one-to-one correspondence between a tree for a discrete function and an expression for the same function; hence, one can select the representational form which is better suited to the manipulation required in any specific case. For instance a fault tree for a digital, analog, or hybrid system is a concept widely used to represent the interconnections of subsystems as a directed graph which clearly illustrates the hierarchical decomposition into major subsystems, then minor subsystems, then individual components; but the equivalent fault expression is often easier to manipulate when one wants to determine the criticality of a subsystem or estimate the total system's reliability as based on subsystem or component-level calculations.

The initial work on decision trees was carried out as Dr. Moret's PhD research at the University of Tennessee and has continued with a focus on the area of fault trees. The major results are these four contributions:

i) a generalization of decision trees to simple recursive functions through a process of composition which allows functional as well as hierarchical decomposition of systems, including systems with feedback;

ii) a characterization of the complexity of testing certain classes of Boolean functions, which has implications in logic design and programming;

iii) a study of the "activity" of a variable as a generalization of Chow parameters with close connections to Boolean differences, which is a useful tool in assessing subsystem importance and designing test sets;

iv) an extension of Boolean difference techniques to the analysis of time-dependent systems with applications to common-cause analysis.

Decision trees are a natural model of the sequential evaluation of discrete functions where, at each node, a variable is evaluated and a decision (to output the functional value or to look at another variable) is made. Such a model is effective for Boolean functions as well as more general, multivalued functions because it is a compact

representation with an inherent ordering of variables for evaluation.

Since the number of tree forms for a given discrete function has an exponential dependence on the number of intrinsic variables, the complexity of optimizing decision trees with respect to several criteria was examined in detail and reported in Moret [1980] and Moret et. al [1981a, 1981b]. A specific measure on discrete functions, called the activity of a variable, was defined and shown to be closely related to the evaluation cost of decision trees for the function [Moret et al. 1980]. The activity of a variable is a generalization of concepts developed in the framework of Boolean functions, such as Chow parameters and Boolean differences (cf., Moret et al. [1980]), all of which are valuable analytical tools in studies of the importance of subsystems in the overall system operation and the suseptability of total system failure to the failures of individual subsystems.

The activity of a variable also finds application in system testing. In particular, exercising those variables having the highest activities maximizes the probability of error detection in systems with equally likely faults. Moreover, the concept can be extended to sequential functions by considering the long-term, steady-state distribution of system states, so that tests can be designed to reflect average or normal operational modes. Finally, the activity is similar to previously used measures of subsystem criticality or importance; however, the activity measure readily generalizes to multi-valued models--a significant advantage where analog and hybrid systems are concerned.

Some results were obtained for proper subsets of the Boolean functions. It was shown that all symmetric and threshold Boolean functions have worst-case (i.e., total variable) testing complexity. Since these functions are commonly encountered in fault modeling, logic design, and pattern recognition, this result provides information useful in these fields. The result appears in Moret et al. [1983].

It should be noted that adopting decision trees rather than more conventional forms of discrete functions led to a unified framework in which several previously disconnected results were seen to fit together. Dr. Moret's survey article [Moret, 1982] has been used by practitioners in a variety of fields, including engineering and scientific disciplines, and has been cited frequently.

3. FAULT TREES

The most recent work has concentrated on fault trees as a special usage of decision trees as system models with emphasis on reliability and testing. A digital, analog, or hybrid system is modelled as a configuration of basic components, each of which is either working or faulty (as defined by the value of a Boolean state variable); the configuration, in turn, is described by higher-level subsystem functions, and ultimately by the overall system function (the value of which is the "top event" state in that it indicates "system working" or "system failed"). A fault tree itself is a logic-operation realization of this system function.

Such trees are widely used in areas in which very complex systems must be analyzed, for example, in the aircraft and nuclear power industries. However, as usually developed, fault trees do not account for time-dependent system reconfigurations or for non-binary component behavior (eg., partially working and satisfactory for some but not all, configurations). In order to extend the applicability of the fault tree concept, Moret and Thomason have extended an idea of Thomason and Page [1976] in using time-dependent Boolean differences for analysis of sequential fault functions for systems which undergo a reconfiguration at discrete points in time Initial work on the inclusion of probabilities was also performed so that estimates of long-run failure probabilities could be calculated for appropriate assumptions of steady-state conditions and independence of failure events.

This method also allows a study of arbitrary subconfigurations in the total system. A characterization of minimum and maximum test conditions has been developed for the sensitization of the system to an arbitrary combination of events in the subsystems. It is shown that some fundamental results in stochastic process theory can be applied to time-dependent systems with suitable transition probabilities. These results provide a basis for the qualitative and quantitative analysis of "common-causes" of simultaneous failures in several subsystems.

4. TESTING COMPLEXITY

The problem of developing test sets can be considered in two stages: a stage in which potential tests are designed and their results measured, and a stage in which the final set of tests is selected. This second stage involves the optimization of some criterion function and often reduces to selecting the smallest possible number of tests in the final collection, i.e., the "minimum test set problem." This computationally intractable problem is NP-hard, as a result of which many researchers have have worked on suboptimal strategies in the form of heuristic search methods.

The objective of the study on this contract was a theoretical and practical evaluation of various suboptimal strategies. Several minimization routines were run under different conditions for comparisons of their growth in complexity predicted by theory with the difficulties actually encountered in solving real problems (Moret and Shapiro [1982]).

Characterizing several aspects of the suboptimal algorithms required extensive experimentation. The outcome of over three thousand test runs brought to light two encouraging results. First, despite the theoretical prediction of sharply increasing complexity, the coherence or "individuality" of the real world problems caused their complexity to increase only slowly with size. Second, the experiments clearly showed that one of the algorithms was far superior to the others for the range of problems considered; this was in agreement with theoretical predictions based on bounding methods that were developed in the course of this contract.

Overall, this effort has contributed to a much better understanding of the "minimum test set problem" and its various suboptimal solutions. In particular, the best existing algorithm has been identified and characterized. Well supported by intuition and empirical evidence, an exact characterization of the best behavior of such algorithms has been conjectured but as yet not proved; should it prove true, the existing alogorithm would in fact be as good as can be achieved.

5. OPTIMAL SOLUTION OF LINEAR INEOUALITIES

Linear inequalities are applicable in digital systems in the areas of threshold functions and pattern recognition. Although the solution of consistent inequalities is straightforward (e.g., by linear programming), relatively little is known about the solution of inconsistent inequalities. The first practical algorithm is this area was reported by Warmack and Gonzalez in 1973. These results were generalized by Clark and Gonzalez [1981] as part of the work on this contract. The Clark-Gonzalez algorithm is a nonenumerative procedure guaranteed to find all optimal solutions to a set of inconsistent inequalities. (Finding the solutions of consistent inequalities is a special case of this method.) Bounds on the search carried out by the algorithm were developed, and the method was shown to be computationally superior to other methods (including the Warmack-Gonzalez algorithm) for finding minimum-error solutions.

6. MOMENTS OF THE INTERCLASS MAHALANOBIS DISTANCE

The Mahalanobis distance is a measure of similarity between multivariate Gaussian populations. In terms of the work in this contract. the Mahalanobis distance offers a robust descriptor for characterizing multivariate measurements performed in an analog system. In this context, the problem can be formulated as a pattern recognition task whose objective is to detect deviations from a normal mode of operation.

When treated as a random variable. the Mahalanobis distance has a probability density function (PDF) that can be related to the probability of error in classification (e.g., classification of normal vs. abnormal operation). When the covariance matrices are equal. obtaining this PDF is straightforward; however the more general (and practical) case involving unequal covariance matrices requires complicated numerical integration techniques for determining the PDF.

In many applications of multivariate data description. it is of interest to compute the moments of the Mahalanobis distance without having to estimate its underlying PDF as an intermediate step. In a recent paper (Gonzalez and Wagner [1983]) it was shown that the moments of the interclass Mahalanobis distance between two multivariate groups of data (also called classes) can be expressed in a simple polynomial form. The nth moment is expressible as a polynomial of order n whose variable depends upon the mean vectors and eigenvalues of the covariance matrices of the two populations. A closed form solution is also given for computing the coefficients of the expressions. The relative simplicity of these results has important implications in terms of implementation in a digital computer or dedicated hardware.

7. SEMI-INVARIANTS OF THE INTERCLASS MAHALANOBIS DISTANCE

An alternative to the technique discussed in the previous section is to compute the semi-invariants (which do not require that the eigenvectors be known) and then obtain the moments from the semi-invariants. A new approach for obtaining the semi-invariants was recently reported by Gonzalez and Wagner [1984]. The semi-invariants are given directly in terms of the mean vectors and inverse covariance matrices. It is well known that the moments and semi-invariants are related by expressions which, though theoretically simple, are quite inefficient in terms of computation. A new, iterative algorithm that is easily implemented on a computer was also reported in the same paper.

REFERENCES

Clark, D.C. and Gonzalez, R.C. [1981]. "Optimal solution of linear inequalities with applications to pattern recognition", IEEE Trans. PAMI. vol. PAMI-3, no. 6. pp. 643-655.

Gonzalez. R.C. and Wagner, C.G. [1983]. "Moments of the interclass Mahalanobis distance", IEEE Trans. SMC. vol. SMC-13. no. 6. pp. 1135-1139.

----. [1984]. "Semi-invariants of the interclass Mahalanobis distance", IEEE Trans SMC. vol. SMC-14. no. 3. pp. 534-538.

Moret. B.M.E. [1982]. "Decision trees and diagrams", ACM Comp. Surveys. vol. 14, no. 4. pp. 593-623.

----. [1980]. "The representation of discrete functions by decision trees: aspects of complexity and problems of testing", PhD dissertation. University of Tennessee. Knoxville.

Moret. B.M.E., and Shapiro. H.D. [1982]. "Experience with the minimum test set problem", UNM CS Tech. Rept. CS82-4, 32 pp. (Accepted by SIAM J. Stat. and Sci. Comp. as "On minimizing a set of tests".)

Moret. B.M.E. and Thomason. M.G. [1983]. "Boolean difference techniques for time-dependent and common-cause analysis of fault trees", UNM CS Tech. Rept. CS83-6, 18 pp. (Submitted to IEEE Trans. Reliability.)

Moret. B.M.E., Thomason. M.G., and Gonzalez, R.C. [1981a]. "Optimization criteria for decision trees", UNM CS Tech. Rept CS82-6. 17 pp.

----. [1983]. "Symmetric and threshold functions are exhaustive", IEEE Trans. Comp., vol. C-32. no. 12. pp. 1211-1212.

----. [1980]. "The activity of a variable and its relation to decision trees", ACM TOPLS. vol. 2, no. 4, pp. 580-595.

----. [1981b]. "The use of activity in testing digital and analog systems", Proc. IEEE Workshop on Automatic Test Program Generation. Philadelphia, pp. 120-127 (Invited paper).

Thomason. M.G. and Page. E.W. [1976]. "Boolean difference techniques in fault tree analysis", Int'l. J. Comp. Info. Sci., vol. 5. no. 1. pp. 81-88.

SECTION II

DECISION TREES

The Activity of a Variable and Its Relation to Decision Trees

B. M. E. MORET, M. G. THOMASON, AND R. C. GONZALEZ
University of Tennessee

The construction of sequential testing procedures from functions of discrete arguments is a common problem in switching theory, software engineering, pattern recognition, and management. The concept of the activity of an argument is introduced, and a theorem is proved which relates it to the expected testing cost of the most general type of decision trees. This result is then extended to trees constructed from relations on finite sets and to decision procedures with cycles. These results are used, in turn, as the basis for a fast heuristic selection rule for constructing testing procedures. Finally, some bounds on the performance of the selection rule are developed.

Key Words and Phrases: activity, decision diagrams, decision tables, decision trees, expected testing cost, heuristic selection, identification procedure, pattern recognition, recursiveness, sequential testing procedure, software engineering, switching theory
CR Categories: 3.63, 3.7, 4.33, 4.34, 4.6, 5.39, 6.1, 8.3

1. INTRODUCTION

A common problem in switching theory, software engineering, pattern recognition, and management is the construction of sequential testing procedures (also called decision trees or decision programs) from a given function of discrete arguments [1, 5, 9, 11, 13, 16, 20]. The problem is to select from the numerous available trees one which is an optimal tree representation with respect to some criterion. In particular, it is often desired to select a tree which has the smallest expected testing cost, that is, a tree such that the average cost of determining a value of the function (by testing some of the variables) is minimal. Variants of this problem have been studied by many researchers, who have provided search algorithms to find the optimal tree(s) [4, 7, 10, 12, 15] or proposed heuristic rules for constructing suboptimal trees [3, 5, 6, 14, 17, 19].

In this paper we introduce the concept of activity of a variable and prove a theorem relating it to the expected testing cost of decision trees with costs and probabilities. This result is then extended to trees constructed from relations on finite sets and to decision procedures with cycles (corresponding to recursive

This work was supported by the Office of Naval Research under Contract N00014-78-C-0311.
Authors' addresses: B.M.E. Moret, Department of Computer Science, University of New Mexico, Albuquerque, NM 87131; M.G. Thomason, Department of Computer Science, University of Tennessee, Knoxville, TN 37916; R.C. Gonzalez, Department of Electrical Engineering, University of Tennessee, Knoxville, TN 37916.

functions). This provides the basis for a fast heuristic selection rule, which is a generalization of criteria proposed in [4] and [15]. Finally, we examine certain conditions under which the rule performs optimally and give some bounds on its behavior.

2. PRELIMINARIES

We are given a (partial) function of n discrete-valued variables, $f(x_1, \ldots, x_n)$; each variable x_i can take on exactly m_i values, $m_i > 1$, and the determination of its value incurs cost c_i. A discrete probability distribution is also specified on the $\prod_{i=1}^{n} m_i$ points of the variables' space (combinations of variable values which do not belong to the inverse image of f do not necessarily have zero probability); the probability of a point is denoted $p(x_1, \ldots, x_n)$. It is noted that the probability $p(x_i = k)$ that variable x_i will take on value k can be computed by

$$p(x_i = k) = \sum_{x_1=1}^{m_1} \cdots \sum_{x_{i-1}=1}^{m_{i-1}} \sum_{x_{i+1}=1}^{m_{i+1}} \cdots \sum_{x_n=1}^{m_n} p(x_1, \ldots, x_{i-1}, k, x_{i+1}, \ldots, x_n). \quad (1)$$

Definition 1. If $f(x_1, \ldots, x_n) =$ constant, then the decision tree for f is a leaf labeled "constant"; otherwise, for each x_i, f has decision tree(s) composed of a root labeled "x_i" and m_i decision subtrees, corresponding to the m_i subfunctions $f_{(x_i=k)}$, $1 \le k \le m_i$.

If variables $x_{k_1}, \ldots, x_{k_{n_k}}$, in that order, are tested along path P_k, yielding values $v_1, \ldots, v_{n_k}$ and leading to leaf α_k, then the probability of reaching leaf α_k is the sum of the probabilities of all combinations of variable values leading to that leaf. Using (1) above, this can be written as

$$p(\alpha_k) = \prod_{i=1}^{n_k} p(x_{k_i} = v_i).$$

In following path P_k, we test n_k variables for a total cost of

$$C(P_k) = \sum_{i=1}^{n_k} c_{k_i}.$$

Thus the *expected testing cost* of the tree T is the quantity

$$C(T) = \sum_{k} p(\alpha_k) \cdot C(P_k),$$

where the sum is taken over all leaves α_k of T.

Any internal node of a decision tree T is associated with a subfunction of T. That subfunction itself has a probability which is the sum of the probabilities of the combinations of variable values included in the subfunction. This is equal to the probability of reaching the said internal node or, equivalently, to the sum of the probabilities of the leaves of the subtree rooted at that internal node. In the following section we shall be interested in the subfunction resulting from a combination of $n - 1$ values, that is, the case in which the values of all variables but, say, x_i are fixed, resulting in the selection of an m_i-tuple of possible combinations—the m_i values of the unspecified variable x_i.

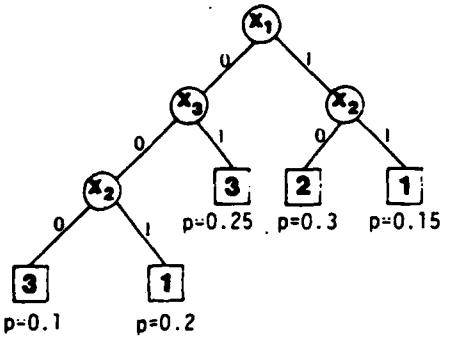


Fig. 1. A sample decision tree for Example 1.

Example 1. Let f be a partial function of three binary variables, $f: \{0, 1\}^3 \rightarrow \{1, 2, 3\}$, given by

$$(0, 0, 0) \rightarrow 3, \quad (1, 0, 0) \rightarrow 2,$$
$$(0, 1, 0) \rightarrow 1, \quad (1, 0, 1) \rightarrow 2,$$
$$(0, 1, 1) \rightarrow 3, \quad (1, 1, 0) \rightarrow 1.$$

The costs are $c_1 = 0.5$, $c_2 = 0.68$, and $c_3 = 0.25$, and the probability distribution is specified by

$$p(0, 0, 0) = 0.10, \quad p(1, 0, 0) = 0.25,$$
$$p(0, 0, 1) = 0.05, \quad p(1, 0, 1) = 0.05,$$
$$p(0, 1, 0) = 0.20, \quad p(1, 1, 0) = 0.15,$$
$$p(0, 1, 1) = 0.20, \quad p(1, 1, 1) = 0.00.$$

A possible decision tree for this function is illustrated in Figure 1, together with the probabilities of the leaves. The expected testing cost of that tree is

$$C(T) = 0.1 \cdot (0.5 + 0.25 + 0.68) + 0.2 \cdot (0.5 + 0.25 + 0.68) + 0.25 \cdot (0.5 + 0.25) + 0.3 \cdot (0.5 + 0.68) + 0.15 \cdot (0.5 + 0.68) = 1.1475. \quad \square$$

3. THE ACTIVITY OF A VARIABLE

Considering the m_t-tuple of combinations mentioned at the end of Section 2, we distinguish two cases:

(i) two of the m_t combinations are mapped to distinct values by f;
(ii) no such two combinations can be found.

In the first case, variable x_i must be tested in order to distinguish all values of the function; in the second case, this is not necessary, although it may be done in a particular tree, either as a redundant test or because at least one variable did not belong to the inverse image of f and has been arbitrarily mapped to a value distinct from the image of the other combinations. Thus, the a priori probability $p_f^+(x_i)$ that variable x_i will be needed in testing all the values of f (i.e., the probability that f will be sensitized to x_i) is equal to the sum of the probabilities of all the m_t-tuples satisfying case (i) above; conversely, the a priori probability $p_f^-(x_i)$ that x_i will be useless is equal to the sum of the probabilities of the remaining m_t-tuples, those satisfying case (ii).

The same reasoning is easily adapted to a subfunction $\bar{f}$ by normalizing the probabilities with the probability of $\bar{f}$. For any x_i and $\bar{f}$, $p_{\bar{f}}^{+}(x_i) + p_{\bar{f}}^{-}(x_i) = 1$.

Definition 2. The *activity* of variable x_i with respect to subfunction $\bar{f}$ is defined as the quantity

$$a_{\bar{f}}(x_i) = c_i \cdot p_{\bar{f}}^{+}(x_i).$$

Definition 3. The *loss* of variable x_i with respect to subfunction $\bar{f}$ is defined as the quantity

$$l_{\bar{f}}(x_i) = c_i - a_{\bar{f}}(x_i).$$

The activity of a variable is a measure of how much influence a variable has on the determination of a function's values. A related concept, known as Chow parameter [18], is discussed in [2] and [4]; when all costs are unity and all variable combinations equally likely, the activity of a variable with respect to a completely specified Boolean function of n variables reduces to the Chow parameter of the variable divided by 2^{n-1}.

The loss of a variable x is a measure of the wasted decision power associated with the choice of x as the root of the decision tree. This is intuitively obvious, since such a choice results in testing the variable with probability 1, while the a priori probability of needing x was $p_f^{+}(x)$.

Example 2. The various quantities defined above are computed for the function of Example 1 and are listed below.

$$p_f^{+}(x_1) = 0.35, \quad p_f^{+}(x_2) = 0.7, \quad p_f^{+}(x_3) = 0.4,$$
$$a_f(x_1) = 0.175, \quad a_f(x_2) = 0.476, \quad a_f(x_3) = 0.1,$$
$$l_f(x_1) = 0.325, \quad l_f(x_2) = 0.204, \quad l_f(x_3) = 0.15. \qquad \square$$

The following theorem establishes the relationship between activity, loss, and expected testing cost of decision trees. The proof technique is derived from [4], where a simplified version of this theorem using Chow parameters was proved for completely specified monotone Boolean functions of uniformly distributed variables with unity costs.

THEOREM 1. *The expected testing cost $C(T)$ of a decision tree T for the function $f(x_1, \ldots x_n)$ can be expressed as*

$$C(T) = \sum_{i=1}^{n} a_f(x_i) + \sum_{k} p(\bar{f}) \cdot l_{\bar{f}}(\beta_k), \tag{2}$$

where the second sum is taken over all internal nodes β_k and $\bar{f}$ refers to the subfunction associated with β_k.

Remark. This theorem says that the expected testing cost of a decision tree is composed of a fixed "overhead" (the first sum) and a variable amount of "loss" (the second sum) which depends on the structure of the tree.

PROOF. The proof is by induction on n, the number of variables. For $n = 1$, the basis is easily verified: the variable space is just an m-tuple, and there are only two possible tree structures. Assume that the theorem holds for all functions

of up to and including $n - 1$ variables, and let f be a function of n variables. Choose x_i to be the root of T. This determines m_i subfunctions, each of $n - 1$ variables, so that the inductive hypothesis applies and for each subfunction f_j, $j = 1, \ldots, m_i$, we have

$$C(T_j) = \sum_{\substack{k=1 \\ k \neq i}}^{n} a_{f_j}(x_k) + \sum_{\beta} p(\bar{f}_j) \cdot l_{f_j}(\beta_s),$$

where the second sum is taken over all internal nodes β_s of T_j. But $C(T) = c_i + \sum_{j=1}^{m_i} p(f_j) \cdot C(T_j)$, and after substituting and simplifying, we obtain

$$C(T) = c_i - l_f(x_i) + \sum_{j=1}^{m_i} \left(p(f_j) \cdot \sum_{\substack{k=1 \\ k \neq i}}^{n} a_{f_j}(x_k) \right) + \sum_{k} p(\bar{f}) \cdot l_f(\beta_k), \tag{3}$$

where the last sum is taken over all internal nodes β_k of T. But we know that

$$c_i - l_f(x_i) = a_f(x_i)$$

and

$$\sum_{j=1}^{n} a_f(x_j) = a_f(x_i) + \sum_{j=1}^{m_i} \left(p(f_j) \cdot \sum_{\substack{k=1 \\ k \neq i}}^{n} a_{f_j}(x_k) \right).$$

Substitution of these two equalities in (3) yields

$$C(T) = \sum_{i=1}^{n} a_f(x_i) + \sum_{k} p(\bar{f}) \cdot l_f(\beta_k),$$

where the second sum is taken over all internal nodes β_k of T. □

COROLLARY 1. *The expected testing cost of any decision tree T for the function $f(x_1, \ldots x_n)$ having x_i as root is bounded by*

$$\sum_{j=1}^{n} c_j \geq C(T) \geq l_f(x_i) + \sum_{j=1}^{n} a_f(x_j).$$

This corollary, in simplified form, was proved in [15] and is implicit in [4]. Both references use it as the basis for a branch-and-bound search algorithm to find a tree that is optimal with respect to the expected testing cost.

These results stress the importance of the sum of the activities of the variables of a function as a representation-independent measure of the cost incurred in determining the values of that function. This motivates the following definition.

Definition 4. The *intrinsic cost* $I(f)$ of the function $f(x_1, \ldots x_n)$ is defined as the quantity

$$I(f) = \sum_{i=1}^{n} a_f(x_i).$$

Example 3. Using the values of activity computed in Example 2 for the function of Example 1, we obtain the intrinsic cost of f,

$$I(f) = a_f(x_1) + a_f(x_2) + a_f(x_3) = 0.175 + 0.476 + 0.1 = 0.751.$$

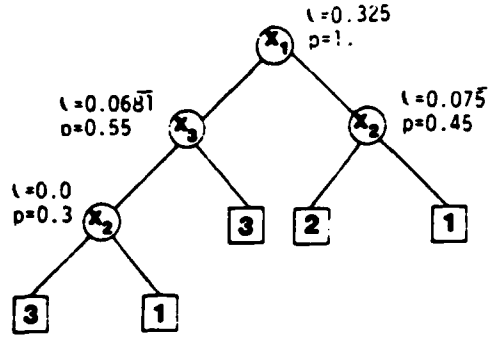


Fig. 2. The tree of Figure 1 with node losses and probabilities.

For the tree of Figure 1 we compute the loss and probability of each internal node to obtain the values shown in Figure 2. The sum of these losses, weighted by the node probabilities, and of the intrinsic cost is

$$(0.325 \cdot 1) + (0.6\overline{81} \cdot 0.55) + (0.07\overline{5} \cdot 0.45) + (0.0 \cdot 0.3) + 0.751 = 1.1475,$$

the computed expected testing cost of the tree. Corollary 1 indicates that any tree for f having x_1 as root will have a minimum cost of $0.751 + 0.325 = 1.076$; similarly, any tree with root x_2 will have a minimum cost of $0.751 + 0.204 = 0.955$, while trees rooted in x_3 have a lower bound of $0.751 + 0.15 = 0.901$. □

4. EXTENSION TO RELATIONS

We extend the definitions of activity and loss to relations on finite sets. This is of particular interest in the case of interdependent functions which must be represented by a single tree (as in [5]).

A relation R might specify no more than one output for each input combination, in which case it is a (partial) function. R may, however, specify more than one output, in which case we assume that we can arbitrarily decide to specify any particular output or leave the choice open. It is also assumed that an unspecified entry (a "don't care") is in fact related to the whole output set, so that any one output value can be selected for such input combination. We then extend the definitions of activity, loss, and intrinsic cost in the obvious way by noting that a variable is needed to differentiate the values of an m_i-tuple if and only if the intersection of the output sets specified by R for the m_i components is empty. It is readily verified that all results previously stated for partial functions remain valid for relations.

Example 4. Consider the relation R from the input set $\{0, 1\}^2 \times \{0, 1, 2\}$ to the output set $\Omega = \{a, b, c, d\}$, where all three variables have unity cost and the relation and the probability distribution are given in Figure 3. Since all variables have unity costs, $p_f^*(x_i) = a_f(x_i)$, so that $a_f(x_1) = 0.35$, $a_f(x_2) = 0.2$, and $a_f(x_3) = 0.6$. The intrinsic cost of R is $I(R) = 0.35 + 0.2 + 0.6 = 1.15$. Choosing x_3 as the root for a decision tree results in a lower bound on the cost of $1.15 + (1 - 0.6) = 1.55$. A possible decision tree T rooted in x_3 is shown in Figure 4, together with

Relation

x_3 \ x_1x_2	00	01	11	10
0	Ω	a	a	a,b,c
1	b,d	b	a	b,c,d
2	d	b	c	c

Probabilities

x_3 \ x_1x_2	00	01	11	10
0	0.05	0.10	0.20	0.10
1	0.10	0.10	0.05	0.10
2	0.00	0.05	0.10	0.05

Fig. 3. The relation and its probability distribution for Example 4.

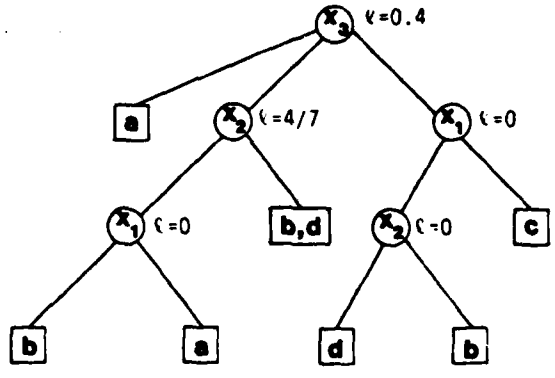


Fig. 4. A decision tree with node losses for Example 4.

the losses of its nodes. Its expected testing cost is $C(T) = 1.15 + (0.35 \cdot \frac{4}{7}) = 1.75$, and it is in fact one of the optimal trees for R. □

5. EXTENSION TO RECURSIVE FUNCTIONS AND RELATIONS

As a further extension of the foregoing concepts, we consider the case of a recursive function or relation, that is, a relation which, for certain input combinations, does not specify output values but calls for the evaluation of some relation, possibly itself. It is assumed that the same tree structure is used for all evaluations of a given relation and that an unspecified entry is not replaced by a call to a relation, but only by values.

The following discussion is restricted to immediate recursive relations, that is, those which do not call for the evaluation of any other relation than themselves. This does not diminish the generality of the development, as a hierarchy of several different relations can be analyzed in parts by considering each relation separately and then merging the results using the probabilities of each relation and of the recursive calls. Such an analysis is demonstrated in Section 6 by an example.

Given an immediate recursive relation, it is possible under the assumptions to compute the probability e that an evaluation will be made without recursive calls. If e is 1, the relation is not recursive; if e is 0, then the relation will never yield a value but will keep issuing recursive calls ad infinitum.

A first question about such relations concerns an upper bound on their testing cost. Such a bound is set by Corollary 1 for nonrecursive relations as the sum of the testing costs of the variables, but can evidently be passed by recursive relations. The following proposition provides the answer.

PROPOSITION 1. *Let R be an immediate recursive relation on n variables $x_1, \ldots, x_n$ with costs $c_1, \ldots, c_n$, and let e be as above; then the expected testing cost of R is no larger than* $(1/e) \cdot \sum_{i=1}^{n} c_i$.

PROOF. The probability of a recursive call occurring in any evaluation is $1 - e$. At worst, an evaluation results in the test of all variables, for a cost of $\sum_{i=1}^{n} c_i$; thus the total cost is no larger than

$$\sum_{k=0}^{\infty} \left((1-e)^k \cdot \sum_{i=1}^{n} c_i \right) = \left(\frac{1}{e} \right) \cdot \sum_{i=1}^{n} c_i. \qquad \square$$

A decision procedure for a recursive relation is an infinite tree that can also be represented as a diagram with cycles, each cycle leading back to the root of a subdiagram. In the case of immediate recursive relations, all cycles lead back to the root of the diagram. We can compute the probability that the relation will take on a specific value by solving a simple linear equation, subject to the convention that entries for which several values are specified are set to the specific value under consideration wherever possible.

The notion of activity of a variable is generalized to immediate recursive relations as follows.

(i) If an m_i-tuple does not include a recursive call, we count its contribution in the usual way.
(ii) If one or more recursive calls are included, the contribution is the probability of the m_i-tuple times the testing cost of the unspecified variable times the probability that the m_i-tuple will be mapped to more than one value.

We call this quantity the *tree activity*; the corresponding loss, the *tree loss*, is the testing cost minus the tree activity. The same quantities multiplied by $1/e$ will be referred to as *diagram activity* and *diagram loss*.

THEOREM 2. *Let R be an immediate recursive relation, and let a decision procedure for R be represented by a diagram D and an infinite tree T. The expected testing cost of the procedure, $C(D) = C(T)$, is equal to*

(i) the sum of the diagram activities and of the diagram losses taken over all internal nodes of the diagram, or
(ii) the sum, taken over the infinite tree, of the tree activities and of the tree losses.

Remark. The sum of the diagram activities is called the *intrinsic cost* of the relation, $I(R)$.

PROOF. The proof relies on the original theorem for nonrecursive functions and on simple considerations on the series $1, 1 - e, (1 - e)^2, (1 - e)^3, \ldots$ and its sum, $1/e$. If we replace all recursive calls in D by leaves, the cost of the resulting tree is the sum of the tree activities and the tree losses taken over all internal nodes of the tree. Introducing recursion results in a series of invocations, the probabilities of which are described by the series $(1 - e)^k$. $\square$

Corollary 1 is similarly extended.

R1

x_3 \ x_1x_2	00	01	11	10
0	R1	R1	R1	R1
1	R1	a	R2	R1
2	b	R2	Ω	R2

R2

y_3 \ y_1y_2	00	01	11	10
0	R1	a	a	R1
1	b	b	Ω	a

Values

x_3 \ x_1x_2	00	01	11	10
0	0.70	0.05	0.05	0.05
1	0.05	0.01	0.01	0.04
2	0.01	0.01	0.01	0.01

y_3 \ y_1y_2	00	01	11	10
0	0.25	0.10	0.10	0.25
1	0.10	0.05	0.05	0.10

Probabilities

$c(x_1) = 9$, $c(x_2) = 9$, $c(x_3) = 4.5$ $c(y_1) = 50$, $c(y_2) = 45$, $c(y_3) = 36$

Costs

Fig. 5. The relations for the example with their probabilities.

6. AN EXAMPLE

As mentioned above, the results can be extended to recursive hierarchies of relations, subject to our two restrictions. The following example shows how systems of relations are analyzed part by part.

Consider a situation in which a monitoring program must periodically evaluate several system variables. If the sampled values point to a satisfactory status, the program waits for a specific period of time and examines the variables again; otherwise, either a malfunction is identified and the program takes some action and stops, or further analysis is required and some additional variables are examined to determine whether the program should resume its normal cycle or take some action and stop. The first part of the examination (the normal cycle) is described by the relation R1, which includes calls both to itself and to the second relation R2 (the exception cycle), which includes calls to R1. In this example, R1 is a relation between $\{0, 1\}^2 \times \{0, 1, 2\}$ and the set of actions $\Omega = \{a, b\}$, and R2 is a relation between $\{0, 1\}^3$ and Ω, as specified in Figure 5.

The analysis treats R1 and R2 separately and considers a structure from which all recursive calls have been eliminated. Once this structure has been analyzed by the methods developed above, the results are put together using $p(\text{R2})$, the probability that R2 is called from R1 in a given evaluation. Recursion is then taken into account by multiplying the results by $1/e$, where e is the overall probability that no recursion will be needed.

We have $p(\text{R2}) = 0.01 + 0.01 + 0.01 = 0.03$; similarly, $p(\text{R1})$, the probability that R1 will be called in an evaluation of R2, is $0.25 + 0.25 = 0.5$. The probability that no recursive call will be necessary is

$$e = 0.01 + 0.01 + 0.01 + p(\text{R2}) \cdot (0.1 + 0.1 + 0.1 + 0.05 + 0.05 + 0.1) = 0.045,$$

so that $1/e = 22.2$. We can then compute the maximum probabilities of yielding

a or b as

$$p(\mathrm{R1} = a) = [0.01 + 0.01 + p(\mathrm{R2})\cdot(0.1 + 0.1 + 0.1 + 0.05)]\cdot(1/e) = 0.6\overline{7},$$
$$p(\mathrm{R1} = b) = [0.01 + 0.01 + p(\mathrm{R2})\cdot(0.1 + 0.05 + 0.05)]\cdot(1/e) = 0.5\overline{7},$$
$$p(\mathrm{R2} = a) = 0.1 + 0.1 + 0.1 + 0.05 + p(\mathrm{R1})\cdot p(\mathrm{R1} = a) = 0.6\overline{8},$$
$$p(\mathrm{R2} = b) = 0.1 + 0.05 + 0.05 + p(\mathrm{R1})\cdot p(\mathrm{R1} = b) = 0.4\overline{8}.$$

The tree activities are

$$a_{\mathrm{R1}}(x_1) = 9\cdot(0 + 0 + 0.02\cdot p(\mathrm{R2} \neq b) + 0 + 0.02\cdot p(\mathrm{R2} \neq a) + 0) = 0.148,$$
$$a_{\mathrm{R1}}(x_2) = 9\cdot(0 + 0.06\cdot p(\mathrm{R1} \neq a) + 0.02\cdot p(\mathrm{R2} \neq b) + 0 + 0.05 + 0) = 0.716,$$
$$a_{\mathrm{R1}}(x_3) = 4.5\cdot(0.76\cdot p(\mathrm{R1} \neq b) + 0.07 + 0.07 + 0.1) = 2.524.$$

Similarly, we get $a_{\mathrm{R2}}(y_1) = 10$, $a_{\mathrm{R2}}(y_2) = 10.15$, and $a_{\mathrm{R2}}(y_3) = 14.78$. Thus I, the intrinsic cost of the relations, is the sum of the tree activities of R1 and the tree activities of R2 (weighted by $p(\mathrm{R2})$) times $1/e$:

$$I = [0.148 + 0.716 + 2.524 + p(\mathrm{R2})\cdot(10 + 10.15 + 14.78)]\cdot(1/e) = 98.57\overline{5}.$$

The upper bound on the cost is

$$C_{\max} = [9 + 9 + 4.5 + p(\mathrm{R2})\cdot(50 + 45 + 36)]\cdot(1/e) = 587.\overline{3}.$$

Figure 6 describes a possible decision diagram D for the relations; the diagram losses and probabilities appear beside each internal node. The lower bound for the cost of this diagram is the sum of the intrinsic cost and of the diagram loss of x_3:

$$\mathrm{lb}(D) = 98.57\overline{5} + 43.9\overline{1} = 142.48\overline{6}.$$

The cost of the diagram can be computed from Theorem 2(i):

$$\begin{aligned} C(D) = {} & 98.57\overline{5} + 1\cdot 43.9\overline{1} + 0.11\cdot 73.\overline{93} + 0.04\cdot 148.\overline{8} \\ & + 0.02\cdot 137.\overline{7} + 0.02\cdot 97.\overline{7} + 0.02\cdot 200 \\ & + 3\cdot(0.01\cdot 471.\overline{5} + 0.007\cdot 677.\overline{7} + 0.003\cdot 370.\overline{370}) = 197. \end{aligned}$$

This can also be obtained by solving the diagram's cost equation:

$$\begin{aligned} C(D) = {} & 1\cdot 4.5 + 0.85\cdot C(D) + 0.11\cdot 9 + 0.04\cdot 9 \\ & + 0.09\cdot C(D) + 0.02\cdot 9 + 0.02\cdot 9 + 0.02\cdot 9 \\ & + 3\cdot(0.01\cdot 36 + 0.007\cdot 45 + 0.003\cdot 50 + 0.005\cdot C(D)), \end{aligned}$$

yielding $(1 - 0.955)\cdot C(D) = 8.865$, so that $C(D) = 8.865/0.045 = 197$.

7. CONSTRUCTING DECISION PROCEDURES

The construction of decision procedures with minimal expected testing costs is, in many cases, a search problem; that is, no algorithm has yet been devised that does not exhibit an exponential behavior in at least some cases. In particular, in the case of binary identification [7], the problem has been shown to be NP-complete [8]. This leads to a search for efficient rules for constructing suboptimal procedures.

As noted earlier, the loss $l_f(x_i)$ is an approximate measure of the importance of not locating x_i at the root of the subfunction f. Indeed, $l_f(x_i)$ satisfies all the

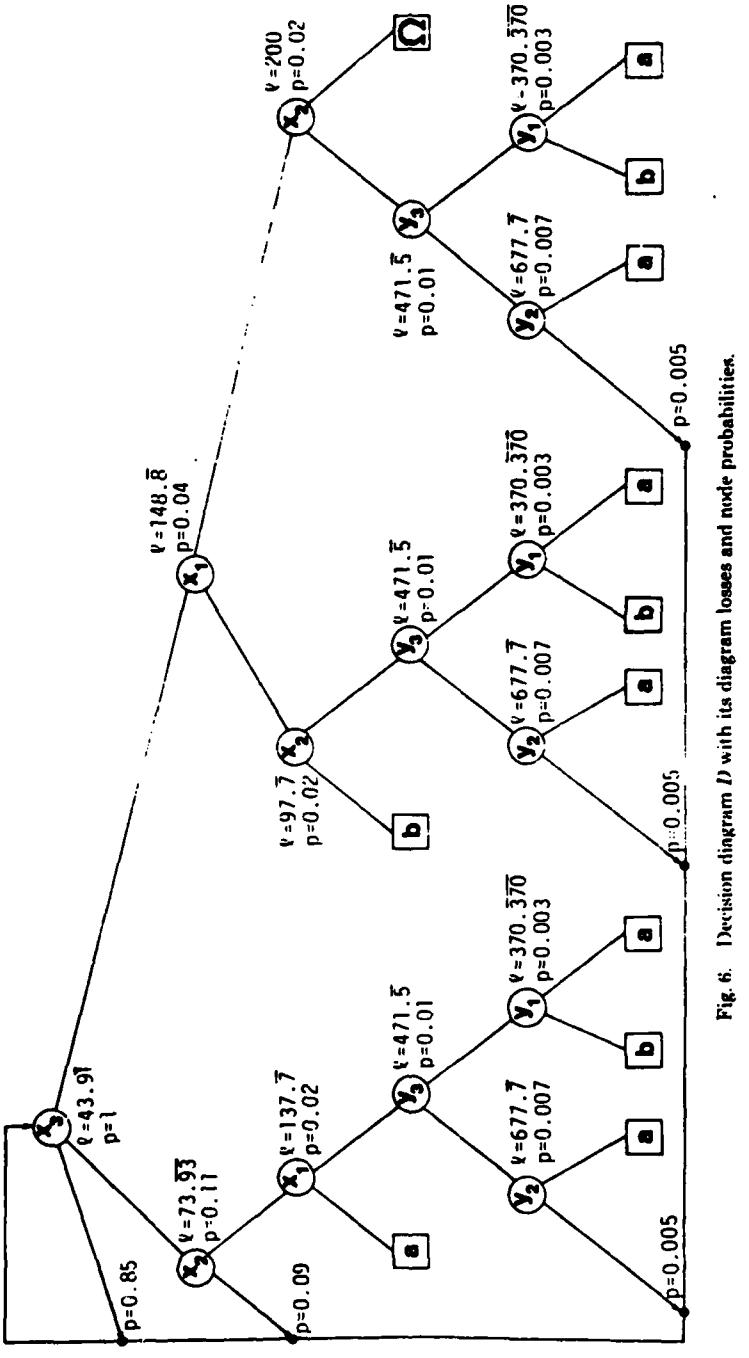


Fig. 6. Decision diagram D with its diagram losses and node probabilities.

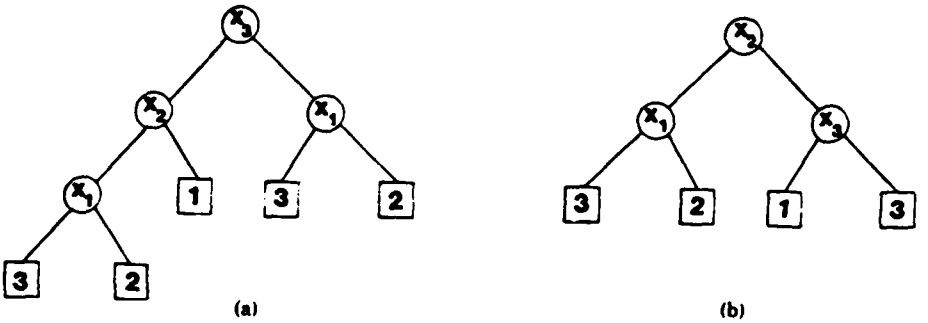


Fig. 7. (a) The tree constructed by the rule. (b) The optimal tree.

requirements set forth in [6] for a selection criterion; that is,

(i) if a variable is necessary to distinguish all of the m_i-tuples it forms, then its activity is equal to its cost, so that its loss is null and it will be tested first (this is an optimal strategy, as can easily be shown [6]);
(ii) if a variable is never needed, that is, if the relation does not intrinsically depend on that variable, then its activity is null and its loss equal to its cost; this condition is easily detected and the variable discarded, unless all other variables have the same status and the relation still specifies at least two distinct values;
(iii) the loss is directly related to the number of m_i-tuples with equal components ("dash" entries in the decision tables discussed in [6]).

This leads to the following rule for local optimization, a generalization of the branch-and-bound criteria used in [4] and [15].

Rule. When developing the decision tree for the subfunction $\bar{f}$, choose as the root the variable with the lowest loss, $l_{\bar{f}}$. In case of a tie, choose the variable with the lowest cost. If a tie subsists, choose any of the variables.

In the example of Section 6, the application of the above rule would result in the diagram of Figure 6, which is optimal in this case. However, the rule does not always result in optimal diagrams. In Example 1 we had $l_{\bar{f}}(x_1) = 0.325$, $l_{\bar{f}}(x_2) = 0.204$, $l_{\bar{f}}(x_3) = 0.15$; thus x_3 would be chosen as the root. Continuing in this manner, we would get the tree of Figure 7a with an expected testing cost of 1.051, but the optimal tree is that shown in Figure 7b, with an expected testing cost of 1.0425. Thus the tree constructed by the rule is $1.0425/1.051 \simeq 0.99$ optimal. A conservative estimate can always be made by substituting the smallest lower bound (as obtained from Corollary 1) for the unknown minimal cost. In the above example, this yields an estimate of $0.901/1.051 \simeq 0.86$.

8. DISCUSSION OF THE SELECTION RULE

An important advantage of the rule is its simplicity; compared to others [3, 6, 14, 17] it requires a minimum of computations. It is also more general, since it applies to any simple recursive or nonrecursive hierarchy of relations with costs and probabilities.

Moreover, the rule is optimal in several cases. As previously noted, it will always lead to the selection of a totally necessary variable if any such variable exists; such a choice was seen to be optimal. We also have the following result.

PROPOSITION 2. *For any recursive relation on two variables, the selection rule constructs optimal diagrams.*

PROOF. Follows immediately from the fact that the lower bound, as computed from Corollary 1, is the exact cost of the diagram. □

Our previous example showed that this result does not hold for functions of three or more variables.

A more important question is how bad the selection can be. The following example illustrates the worst case for completely specified Boolean functions with unity costs.

Let f be the Boolean function $f = x_1 + \oplus_{i=2}^{n} x_i$, where $\oplus$ denotes summation modulo 2, and assume the following probability distribution:

(i) Each point satisfying $x_1 \cdot \overline{\oplus_{i=2}^{n} x_i} = 1$ has probability $\gamma \cdot \epsilon$, for $\gamma < 1$, $\gamma \simeq 1$.
(ii) Each point satisfying $x_1 = 0$ has probability ϵ.
(iii) All other points have probability $\alpha = 2^{2-n} - (\gamma + 2) \cdot \epsilon$.

Then we get $a_f(x_1) = 2^{n-2} \cdot (\gamma + 1) \cdot \epsilon$ and $a_f(x_i) = 2^{n-1} \cdot \epsilon$ for $n \geq i \geq 2$, so that $l_f(x_i) < l_f(x_1)$. The two subfunctions resulting from the choice of some x_i, $i \neq 1$, as the root are again of the form $x_1 + \oplus x_j$, so that the trees constructed by the rule test x_1 last (on half the branches) and have cost $C(T_r) = n - 1 + 2^{n-2} \cdot (\gamma + 1) \cdot \epsilon$, while the optimal trees, rooted in x_1, have a cost of $C(T_o) = 1 + (n - 1) \cdot 2^{n-1} \cdot \epsilon$. (The case $n = 4$ is illustrated in Figure 8.) Thus, if $\epsilon \ll 1$ (e.g., if $\epsilon = 2^{-kn}$ for some $k > 1$), the asymptotic ratio of costs becomes $C(T_r)/C(T_o) = n - 1$.

By letting every point satisfying $\overline{x_1} \cdot \oplus_{i=2}^{n} x_i = 1$ be mapped to a recursive call, we obtain the worst case for recursive Boolean functions. The best diagram, D_o, has a cost of $[1 + (n - 1) \cdot 2^{n-1} \cdot \epsilon]/(1 - 2^{n-2} \cdot \epsilon)$, while the rule-constructed diagram, D_r, has a cost of $n/(1 - 2^{n-2} \cdot \epsilon)$; thus the asymptotic ratio $C(D_r)/C(D_o)$ becomes approximately n for small ϵ. That both recursive and nonrecursive cases yield the same worst case, $O(n)$, is due to the fact that the recursive factor $1/e$ is independent of tree structure and is factored out.

The rule can construct arbitrarily bad trees; however, in the above example the lower bound on the cost of the trees is $\text{lb}(f) = 1 + (n - 2) \cdot 2^{n-1} \cdot \epsilon + 2^{n-2} \cdot (\gamma + 1) \cdot \epsilon$, so that $C(T_0) - \text{lb}(f) = 2^{n-2} \cdot (1 - \gamma) \cdot \epsilon \simeq 0$. Therefore, we could have detected at an early stage that the trees constructed by the rule were costing much more than the original lower bound and revised the selection. This is not to say that the lower bound as obtained from Corollary 1 remains arbitrarily close to the cost of the optimal trees. It is easy to construct a binary identification problem [7] with n variables of unity cost and 2^{n-1} equally likely objects, so that the lower bound is always 1 while the optimal cost is $n - 1$; Figure 9 illustrates such a problem for three variables.

The determination of a general upper bound for the worst trees constructed by the heuristic rule, as well as for the lower bound obtained from Corollary 1, is an object of present study.

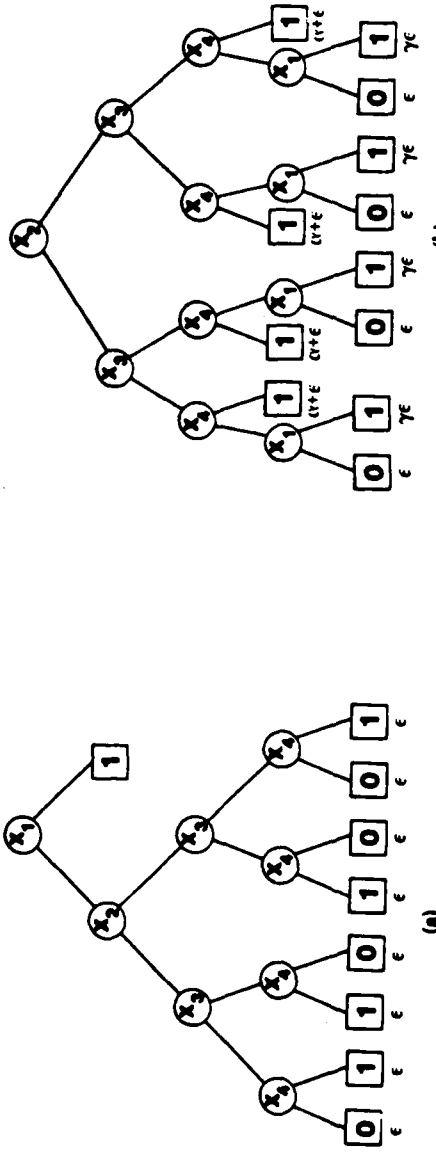


Fig. 8. The two trees for the case $n = 4$, with their leaf probabilities: (a) The optimal tree, T_0. (b) The tree constructed by the rule, T_1.

x_1x_2 / x_3	00	01	11	10
0	1		2	
1		3		4

Fig. 9. An identification problem with three variables and four objects.

9. CONCLUSION

We have introduced the concept of the activity of a variable, a global measure of the relevance of a variable in determining the values of a relation on discrete arguments. We have proved a theorem detailing the relationship of this measure to the expected testing cost of the relation. Finally, we have used this result to develop a heuristic procedure for the fast construction of suboptimal decision diagrams and have indicated some bounds on its performance.

The applicability of these results to recursive functions and decision diagrams with cycles should provide a basis for further developments in fault analysis by allowing sequential testing of time-related processes, as well as by supplying a new modeling tool. Other areas in which these results may find applications include pattern recognition, database theory, and switching theory.

REFERENCES

1. AKERS, S.B. Binary decision diagrams. *IEEE Trans. Comput. C-27*, 6 (June 1978), 509–516.
2. BOZOYAN, SH.YE. Certain properties of Boolean differentials and the activities of the arguments of Boolean functions and questions of construction of reliable schemes that have unreliable elements. *Eng. Cybern. 13*, 5 (Sept. 1975), 108–120.
3. BREITBART, Y., AND REITER, A. Algorithms for fast evaluation of Boolean expressions. *Acta Inf. 4* (1975), 107–116.
4. BREITBART, Y., AND REITER, A. A branch-and-bound algorithm to obtain an optimal evaluation tree for monotonic Boolean functions. *Acta Inf. 4* (1975), 311–319.
5. CERNY, E., MANGE, D., AND SANCHEZ, E. Synthesis of minimal binary decision trees. *IEEE Trans. Comput. C-28*, 7 (July 1979), 472–482.
6. GANAPATHY, S., AND RAJAMARAN, V. Information theory applied to the conversion of decision tables to computer programs. *Commun. ACM 16*, 9 (Sept. 1973), 532–539.
7. GAREY, M.R. Optimal identification procedures. *SIAM J. Appl. Math. 23*, 2 (1972), 173–186.
8. HYAFIL, L., AND RIVEST, R.L. Constructing optimal binary decision trees is NP-complete. *Inf. Proc. Letters 5*, 1 (1976–1977), 15–17.
9. LEE, C.R. Representation of switching circuits by binary decision programs. *Bell Syst. Tech. J. 38* (July 1959), 945–999.
10. MARTELLI, A., AND MONTANARI, U. Optimizing decision trees through heuristically guided search. *Commun. ACM 21*, 12 (Dec. 1978), 1025–1039.
11. METZNER, J.R., AND BARNES, B.M. *Decision Table Languages and Systems*. Academic Press, New York, 1977.
12. PAYNE, H.J., AND MEISEL, W.S. An algorithm for constructing optimal binary decision trees. *IEEE Trans. Comput. C-26*, 9 (Sept. 1977), 905–916.
13. PICARD, C.F. *Graphes et Questionnaires, Volume 2: Questionnaires*. Gauthier-Villars, Paris, 1972.
14. POLLACK, S.L. Conversion of limited-entry decision tables to computer programs. *Commun. ACM 8*, 11 (Nov. 1965), 677–682.
15. REINWALD, L.T., AND SOLAND, R.M. Conversion of limited-entry decision tables to computer programs I: Minimum average processing time. *J. ACM 13*, 3 (July 1966), 339–358.
16. ROUNDS, E.M. A combined non-parametric approach to feature selection and binary tree design. Proc. Conf. on Pattern Recognition and Image Processing, Chicago, Ill., 1979, pp. 38–43.

17. SHWAYDER, K. Extending the information theory approach to converting limited-entry decision tables to computer programs. *Commun. ACM 17*, 9 (Sept. 1974), 532–537.
18. WINDER, R.O. Chow parameters in threshold logic. *J. ACM 18*, 2 (April 1971), 265–289.
19. YASUI, T. Some aspects of decision table conversion techniques. *SIGPLAN Notices 6*, 9 (Sept. 1971), 104–111.
20. YOU, K.C., AND FU, K.S. An approach to the design of a linear binary tree classifier. Proc. Symp. on Machine Processing of Remotely Sensed Data, Lafayette, Ind., 1976, pp. 3A1–3A10.

Received November 1979; revised May 1980; accepted July 1980

Decision Trees and Diagrams

BERNARD M. E. MORET

Department of Computer Science, The University of New Mexico, Albuquerque, New Mexico, 87131

Decision trees and diagrams (also known as sequential evaluation procedures) have widespread applications in databases, decision table programming, concrete complexity theory, switching theory, pattern recognition, and taxonomy—in short, wherever discrete functions must be evaluated sequentially. In this tutorial survey a common framework of definitions and notation is established, the contributions from the main fields of application are reviewed, recent results and extensions are presented, and areas of ongoing and future research are discussed.

Categories and Subject Descriptors: B.6.1 [Logic Design]: Design Styles; B.6.3 [Logic Design]: Design Aids—*switching theory*; D.1.m [Programming Techniques]: Miscellaneous; F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems—*computations on discrete structures*; G.2.2 [Discrete Mathematics]: Graph Theory—*trees*; H.2.4 [Database Management]: Systems—*query processing*; I.2.8 [Artificial Intelligence]: Problem Solving, Control Methods and Search; I.5.1 [Pattern Recognition]: Models; I.5.2 [Pattern Recognition]: Design Methodology; J.3 [Computer Applications]: Life and Medical Sciences—*biology*; *health*

General Terms: Algorithms, Design, Theory

Additional Key Words and Phrases: Atomic digraph, binary identification, Boolean graph, decision program, decision table, diagnostic key, diagnostic table, evaluation, exhaustive function, feature selection, heuristics, hierarchical classifier, multiplexer network, multistage testing, NP-complete problem, sequential evaluation procedure, table splitting, taxonomy, test selection

INTRODUCTION

A decision tree or diagram is a model of the evaluation of a discrete function, wherein the value of a variable is determined and the next action (to choose another variable to evaluate or to output the value of the function) is chosen accordingly. Decision trees find many applications in decision table programming [SILB71, POOC74, METZ77], databases [WONG76, HANA77], pattern recognition [HAUS75, BELL78], taxonomy and identification [JARD71, MORS71, GARE72a, PAYR80, WILL80], machine diagnosis [KLET60, CHAN70], switching theory [LEE59, THAY81a], and analysis of algorithms [WEID77]. More recently, they have been proposed as implementation-independent models of discrete functions with a view to the development of new testing methods [AKER79, MORE81a] and complexity measures [MORE80a].

Owing to this broad applicability, results about decision trees are dispersed throughout the literature in fields such as biology, computer science, information theory, and switching theory; moreover, there is no common notation or set of definitions. Therefore this article begins by establishing a framework of notation and definitions

CONTENTS

that introduce decision trees and diagrams and the various measures associated with them. Particular attention is paid to the problem of constructing decision trees and diagrams from function descriptions and evaluating their efficiency. The complexity of such constructions is detailed, and repercussions on circuit or program design analyzed. A survey of the main fields of application and related results follows. Recently proposed extensions (to include diagram composition and recursion) and applications (e.g., to system testing) are then discussed. The article concludes with an assessment of known results and suggestions for future research.

The emphasis throughout this exposition is on Boolean functions, since they find many more applications and are more readily understood than general discrete functions. The presentation alternates formal exposition, examples, and discussion; complex proofs are avoided (the reader will find them in the references), and the mathematical content is kept to the minimum necessary for clarity and conciseness. In particular, all necessary mathematical and other background is introduced in the first section so that the paper should be accessible to any reader with a mathematical or algorithmic bent. The intent is to cover the breadth of the field, unify terminology, convey the import of the main results, and act as a guide to the literature, for which last purpose a representative, rather than exhaustive, reference list is provided. As such, this survey should be of interest to both practitioners and researchers in the areas mentioned above.

1. PRELIMINARIES

Since the evaluation of Boolean functions, the programming of decision tables, and the identification of unknown objects (biological specimens, system faults, etc.) are among the most important applications of decision trees and diagrams, we provide a succinct review of the terminology and basic concepts of Boolean functions, decision tables, and identification problems. Readers who feel comfortable with these topics may wish to skip to Section 2.

1.1 Discrete and Boolean Functions

Only a very brief review is provided; for more details, the reader is referred to DAVI80 on discrete functions and to HARR65 on Boolean functions.

By *discrete* function, we mean a (partial) function of discrete variables, $f(x_1, \ldots, x_n)$, where each variable, x_i, takes exactly m_i values, which we choose to denote $0, \ldots, m_i - 1$. A discrete function is *constant* if and only if (iff) it assumes the same value wherever it is defined; it is *null* if it is not defined in any point of its domain, *completely specified* if it is defined everywhere. When a variable is evaluated, say $x_i = k$, we are left with the *restriction*, $f(x_1, \ldots, x_{i-1}, k, x_{i+1}, \ldots, x_n)$, which we denote $f|_{x_i=k}$. A variable, x_i, is *redundant* iff

$$f|_{x_i=0} = \cdots = f|_{x_i=m_i-1}, \qquad (1)$$

where two functions are equal if they have the same domain and codomain and assume the same value wherever they are both defined; a function without redundant variables is called *intrinsic*. Finally, a variable,

x_i, is termed *indispensable* iff it is not redundant in any restriction resulting from the evaluation of any subset of the variables $\{x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n\}$. (This implies that a function can never be evaluated at any point without knowledge of the values of all indispensable variables.)

A *Boolean* function of n variables is a discrete function, f: $\{0, 1\}^n \rightarrow \{0, 1\}$, where $\{0, 1\}^n$ denotes the n-fold Cartesian product of $\{0, 1\}$, that is, the set of all binary n-tuples. Each n-tuple, $(x_1, \ldots, x_n)$, mapped to 1 by the function is a *minterm* of the function. A Boolean function can be specified by describing the mapping (giving its "truth table") or by listing its minterms and those points at which it is not defined (so-called "don't care" conditions); it can also be represented by a *Boolean formula*, usually in terms of the three operations of disjunction (+), conjunction (·), and complementation ($^-$). A Boolean function of n variables can be expressed in terms of two functions of $n-1$ variables by means of *Shannon's expansion theorem*

$$f(x_1, \ldots, x_n) = \overline{x_i} \cdot f|_{x_i=0} + x_i \cdot f|_{x_i=1} \quad (2)$$

for each choice of x_i.

Example 1

Consider the Boolean function of three variables, $f(x_1, x_2, x_3)$, given by the mapping

$$\begin{array}{lllllll} f: & (0,0,0) & \rightarrow & 0 & (1,0,0) & \rightarrow & 0 \\ & (0,0,1) & \rightarrow & 0 & (1,0,1) & \rightarrow & 0 \\ & (0,1,0) & \rightarrow & 1 & (1,1,0) & \rightarrow & 0 \\ & (0,1,1) & \rightarrow & 1 & (1,1,1) & \rightarrow & 1 \end{array}$$

Since every point in the domain is assigned a value, the function is completely specified. Other representations for f are the list of its minterms

$$\{(0, 1, 0), (0, 1, 1), (1, 1, 1)\}$$

or a Boolean formula

$$\overline{x_1}x_2\overline{x_3} + \overline{x_1}x_2x_3 + x_1x_2x_3.$$

The latter formula is equivalent to the list of minterms; it can be simplified to yield the minimum expression

$$\overline{x_1}x_2 + x_2x_3.$$

Table 1. Decision Table, Example 2

Raining? Wind condition	Yes	No Breezy	No Calm	Windy
Clean basement	X			X
Spade garden			X	
Fly kite with children		X		

It is easily verified that the function is intrinsic; expanding it around x_2 yields

$$f = \overline{x_2} \cdot 0 + x_2 \cdot (\overline{x_1} + x_3). \quad \square$$

1.2 Decision Tables

The terminology used in the following is that of METZ77; other general references are SILB71 and POOC74.

A *decision table* is an organizational or programming tool for the representation of discrete functions. It can be viewed as a matrix where the upper rows specify *sets of conditions* and the lower ones sets of *actions* to be taken when the corresponding conditions are satisfied; thus each column, called a *rule*, describes a procedure of the type "if conditions, then actions."

Example 2

Table 1 describes how to spend a Saturday afternoon in spring. It has two condition rows, three action rows, and four rules; the first condition is a binary variable (taking values "yes" or "no"), while the second is a ternary variable (taking values "calm," "breezy," or "windy"). According to normal practice [METZ77], condition and action names are used as labels on appropriate rows and a rule is specified by entering values in the condition rows (or blanks, for don't care conditions) and X's (meaning "execute") in the action rows. The four rules can be read as

"if it is raining, then clean the basement";
"if it is breezy and not raining, then fly kite with children";
"if it is calm and not raining, then spade the garden";
"if it is windy, then clean the basement." □

A pair of rules *overlaps* if a combination of condition values can be found that sat-

Table 2. Decision Table, Example 3

Raining?	Yes	No	No	
Calm?		(No)	Yes	(No)
Breezy?		Yes	(No)	(No)
Clean basement	X			X
Spade garden			X	
Fly kite with children		X		

isfies the condition sets of both rules. If two overlapping rules specify different actions, they are called *inconsistent* and the table is said to be *ambiguous*; if they specify identical actions, they are termed *redundant*.

Decision tables described so far are in so-called *extended-entry* form. Often, however, it is required that all conditions be Boolean variables; this gives rise to *limited-entry* decision tables. Although most such tables are set up in limited format from their conception, it may be necessary to convert extended-entry tables to limited-entry format; this is done by using one Boolean variable for each value (but one) of the multivalued variable to be replaced [PRES65]. This process results in tables where entries in one condition row often imply (absent) entries in others; such *implied* entries can also be present in any decision table and give rise to apparent (but nonexistent) ambiguity. Since the implications result from purely semantic considerations, they cannot be detected by an automatic processor, so that they must be explicitly specified. The impossible combinations of conditions will then be treated as inputs with unspecified mapping.

Example 3

In Table 1, Rules 1 and 4 overlap because they are both applicable when it is raining and windy. Since they specify the same action set, they are redundant, and since no other rules overlap, the table is unambiguous. Table 2 is the same table, converted to limited-entry format. It still has three action rows and four rules, but now has three condition rows. Implied entries are shown in parentheses; their absence, while not confusing to a human, would induce an automatic processor to decide that the second and third rules are inconsistent, since both could apparently apply when it is not raining and it is calm and breezy. It is noted that the specification of implied entries in the table is insufficient; while it identifies the impossible condition set (no, yes, yes), it fails to identify the equally impossible set (yes, yes, yes), which will be erroneously included in the first rule. This suggests that logical inconsistencies be separately listed [KING73]; for instance, the above table would be supplemented by the logical expression NOT (breezy AND calm). □

It should now be clear that an unambiguous extended-entry decision table is a special case of a partial function of multivalued variables, where the conditions correspond to the variables and the action sets to the function values. In particular, a *complete* decision table (one which has an applicable rule for every combination of conditions) corresponds to a completely specified function, and a limited-entry decision table corresponds to a function of binary variables. An ambiguous decision table can be modeled by a relation, as discussed later. A sequential evaluation procedure for a decision table is then of particular importance, since it corresponds to an implementation, usually in software, of the decision table; indeed, the importance of the limited-entry format is in good part due to the ease of programming binary decisions (by if-then-else constructs) [METZ77].

1.3 Identification Problems

Consider the situation where an unknown event or specimen is to be classified into one of a finite number of known categories, based upon the outcome of a number of tests. (This is a special case of the concept of questionnaires [PICA72].) Such *identification problems* arise in biology, medical diagnosis, machine trouble shooting, and numerous pattern recognition applications. A *binary identification problem* includes only binary tests.

As defined in GARE72a, a binary identification problem consists of a finite set of *objects*, $(O_1, \ldots, O_n)$, which represents the universe of possible identifications, and a finite set of *tests* $(T_1, \ldots, T_m)$, each of which is a function from the set of objects

to the set {*yes*, *no*} (i.e., each test applied to a specific object gives a specific *yes/no* answer). In a *simple* binary identification problem, all tests are of the form "is the unknown object of type *i*?"; that is, their outcome is *no* for all but one object [GARE72b]. An optional probability distribution can be specified on the set of objects, giving the a priori probability that an unknown object will be identified as each object in the set. In practical situations, the number of tests and the number of objects are often in the same range (even though, in theory, the number of tests could be arbitrarily larger than that of objects and the number of objects could grow as an exponential function of the number of tests).

In our terminology, the tests are binary variables and the objects are values of a partial bijective function from the variables' space to the set of objects. In biology, tests are also known as characters or conditions and objects as taxa or formae, while the specification of an identification problem is known as a diagnostic table; in pattern recognition, tests would be known as features and objects as classes; in questionnaire theory, the tests are questions and their outcomes are responses.

Example 4

Consider the identification problem given by a set of five objects, {a, b, c, d, e}, a set of four tests, $\{T_1 = \{a\}, T_2 = \{a, b\}, T_3 = \{a, b, c\}, T_4 = \{a, b, c, d\}\}$. The same problem can be represented as a diagnostic table (see Table 3), in which the rows correspond to objects and the columns to tests. In our terminology, we have a partial bijective function of four binary variables, defined in five points and given by the following mapping:

(yes, yes, yes, yes) → a
(no, yes, yes, yes) → b
(no, no, yes, yes) → c
(no, no, no, yes) → d
(no, no, no, no) → e □

This formulation provides a clean, but very much simplified model of the general identification problem. In pattern recognition applications, the test results are generally not as clearly defined; instead, each test may take any of the possible values, as specified by a discrete probability function. The same problem arises in biology (where it is known as probabilistic identification); however, since the probability distribution is often unknown, a confidence threshold is normally used, which dichotomizes test outcomes as "known" (taking a specific value) or "variable" (susceptible of taking any value). In the following, we shall first look at Garey's model, then generalize results to include the probabilistic identification model.

Table 3. Diagnostic Table, Example 4

	T_1	T_2	T_3	T_4
a	Yes	Yes	Yes	Yes
b	No	Yes	Yes	Yes
c	No	No	Yes	Yes
d	No	No	No	Yes
e	No	No	No	No

2. DEFINITIONS

2.1 Decision Trees and Diagrams

A decision tree (or diagnostic key, as it is known in many identification applications) can be regarded as a deterministic algorithm for deciding which variable to test next, based on the previously tested variables and the results of their evaluation, until the function's value can be determined. Several constraints are placed upon such an algorithm; all are designed to prevent clearly redundant testing. The following formal definition of a decision tree is taken from MORE80b.

Definition 1

Let $f(x_1, \ldots, x_n)$ be a (partial) function of discrete variables. If f is constant or null, then the decision tree for f is composed of a single leaf labeled by the constant value or by the null symbol. Otherwise, for each x_i, $1 \leq i \leq n$, such that at least two restrictions, say $f|_{x_i=k_1}$ and $f|_{x_i=k_2}$, are not null, f has one or more decision trees composed of a root labeled x_i, and m_i subtrees, which are decision trees corresponding to the restrictions $f|_{x_i=0}, \ldots, f|_{x_i=m_i-1}$, in that order. □

This recursive definition closely parallels the conventional definition of ordered trees,

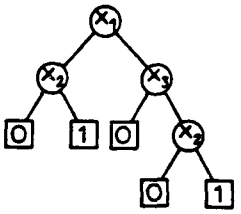


Figure 1. The decision tree of Example 5.

such as binary trees [KNUT73]; it defines decision trees as rooted, ordered, vertex-labeled trees, where each node has either m_i children for some i, $1 \leq i \leq n$, or none (and is a leaf). To an extent, this definition prevents redundant testing: a variable is tested only once on any path; no variable can be tested which would result in all restrictions but one being null; and no more testing can take place as soon as the function has been reduced to a constant.

The evaluation of a discrete function represented as a decision tree starts by ascertaining the value of the variable associated with the root of the tree. It then proceeds by repeating the process on the kth subtree, where k is the value assumed by the root variable, until a leaf is reached; the label of the leaf gives the value of the function.

Example 5

The Boolean function of Example 1 was given by the formula

$$f(x_1, x_2, x_3) = \overline{x_1}x_2 + x_2x_3.$$

A possible decision tree for that function is shown in Figure 1. Since decision trees are ordered, the left subtree of a node corresponds to the node's variable evaluating at 0, the right subtree to the variable evaluating at 1. Thus the left subtree of the root corresponds to the restriction

$$f|_{x_1=0} = x_2,$$

and the right subtree corresponds to the restriction

$$f|_{x_1=1} = x_2x_3.$$

Evaluation on the tree for the triple of values (1, 0, 0) starts by examining x_1; on finding it to be 1, it proceeds to the right subtree, there to evaluate x_3. Since x_3 is found to be 0, the left subtree is next used, thereby encountering a leaf and terminating the evaluation, having used only two of the three variables; the label of the leaf is the value of the function, that is, $f(1, 0, 0) = 0$. It is noted that a decision tree for a Boolean function is an explicit illustration of Shannon's expansion; the tree of Figure 1 represents the expansion

$$f(x_1, x_2, x_3) = \overline{x_1} \cdot [\overline{x_2} \cdot 0 + x_2 \cdot 1] + x_1 \cdot [\overline{x_3} \cdot 0 + x_3 \cdot (\overline{x_2} \cdot 0 + x_2 \cdot 1)].$$

□

We note that the same subtree may occur on several branches of the tree, in which case it may be desirable to use only one copy of that subtree by transforming the decision tree (through a process known as *reticulation* [PAYR77]) into a *simple decision diagram*, which has the structure of a rooted, directed, acyclic (hyper)graph. In the case of Boolean functions, further requiring that there be only one leaf labeled 1 (the "finish" node) yields a *free Boolean graph*. Of course, we can choose which identical subtrees to merge, if any; in particular, every decision tree is a (simple) decision diagram. To every simple decision diagram there corresponds a unique decision tree; moreover, the paths in the diagram are in one-to-one correspondence with those in the tree. Conversely, to every decision tree for a completely specified function there corresponds a unique "minimal" diagram, that is, one in which every possible merge has been accomplished. Reticulation sometimes also merges nonidentical subtrees while preserving the identity of the function; in such cases, it may happen that a variable occurs more than once along a path from the root to a leaf. Such nonsimple decision diagrams may further decrease the total number of nodes required; however, the second test of a variable is redundant and thus detracts from the diagram's "efficiency."

Decision diagrams (and their corresponding trees) can easily be programmed. LEE59 calls the result (simple) *decision programs* and has suggested a universal instruction type which implements the evaluation process taking place at an internal node:

$$L : i, g_0, \ldots, g_{m_i-1},$$

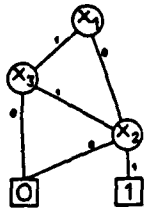


Figure 2. The minimal free Boolean graph of Example 6.

where L is a label, i identifies variable x_i, and g_k, used only when $x_i = k$, is either a value (if the restriction for $x_i = k$ is a constant) or a label. Such an instruction is executed by testing variable x_i and upon finding its value, say $x_i = k$, taking the corresponding action, g_k, that is, either transferring control to the instruction labeled g_k or assigning to the function the value g_k. Thus to each node of the diagram there corresponds one instruction in the program. Cerny [CERN79b] has investigated a special-purpose architecture for the execution of such programs.

Decision trees and diagrams for Boolean functions find yet another implementation, this time in hardware as *multiplexer* trees and networks [CERN79a, THAY81a]. In a multiplexer tree (network), each internal tree (diagram) node is represented as a 2–1 multiplexer controlled by the node variable and each leaf is implemented as a constant logical value (wired at 0 or wired at 1); the interconnection scheme is that of the decision tree (diagram). The evaluation of a function then proceeds from the "leaves" (the constant values) to the "root" multiplexer; the function variables, used as control variables, select a unique path from the root to one leaf, and the value assigned to that leaf propagates along the path to the output of the "root" multiplexer.

Example 6

Consider the tree of Example 5. It is itself a diagram; merging the identical subtrees rooted in nodes labeled x_2 and the identical leaves results in the minimal free Boolean graph pictured in Figure 2. This diagram can be implemented by the decision program (where letters are used for labels to distinguish them from values)

Start: 1, A, B
A: 2, 0, 1
B: 3, 0, A

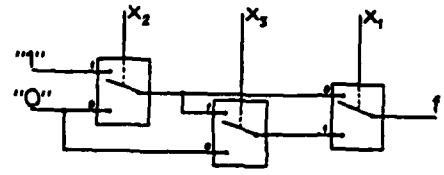


Figure 3. The multiplexer network of Example 6.

Figure 3 depicts an equivalent multiplexer network. We note that, as a consequence of our definitions, the number of multiplexers used in a network is precisely the number of instructions of an equivalent decision program; similarly, the maximum delay through a network is proportional to the maximum execution time of an equivalent program, both being dependent upon the length of the longest path through the diagram. □

2.2 Measures on Decision Trees and Diagrams

Since the root of each subtree can be labeled with any (up to the restrictions of Definition 1) of the untested variables, the number of possible decision trees for a given function is in general very large (and that of possible decision diagrams even larger). For instance, the function of Example 5 has ten distinct decision trees, as shown in Figure 4. In fact, a completely specified Boolean function of n variables can have up to

$$N_T(n) = \prod_{i=0}^{n-1} (n-i)^{2^i} \qquad (3)$$

distinct decision trees. Indeed, n choices are possible for the root, followed by $n - 1$ choices on each of the two subtrees, or $(n-1)^2$ choices; in general, up to $(n-k)^{2^k}$ choices are possible at depth k. This corresponds to the recurrence relation

$$N_T(n) = n \cdot (N_T(n-1))^2,$$

which shows that $N_T(n)$ grows faster than 2^{2^n}. The first few values of $N_T(n)$ are listed in Table 4.

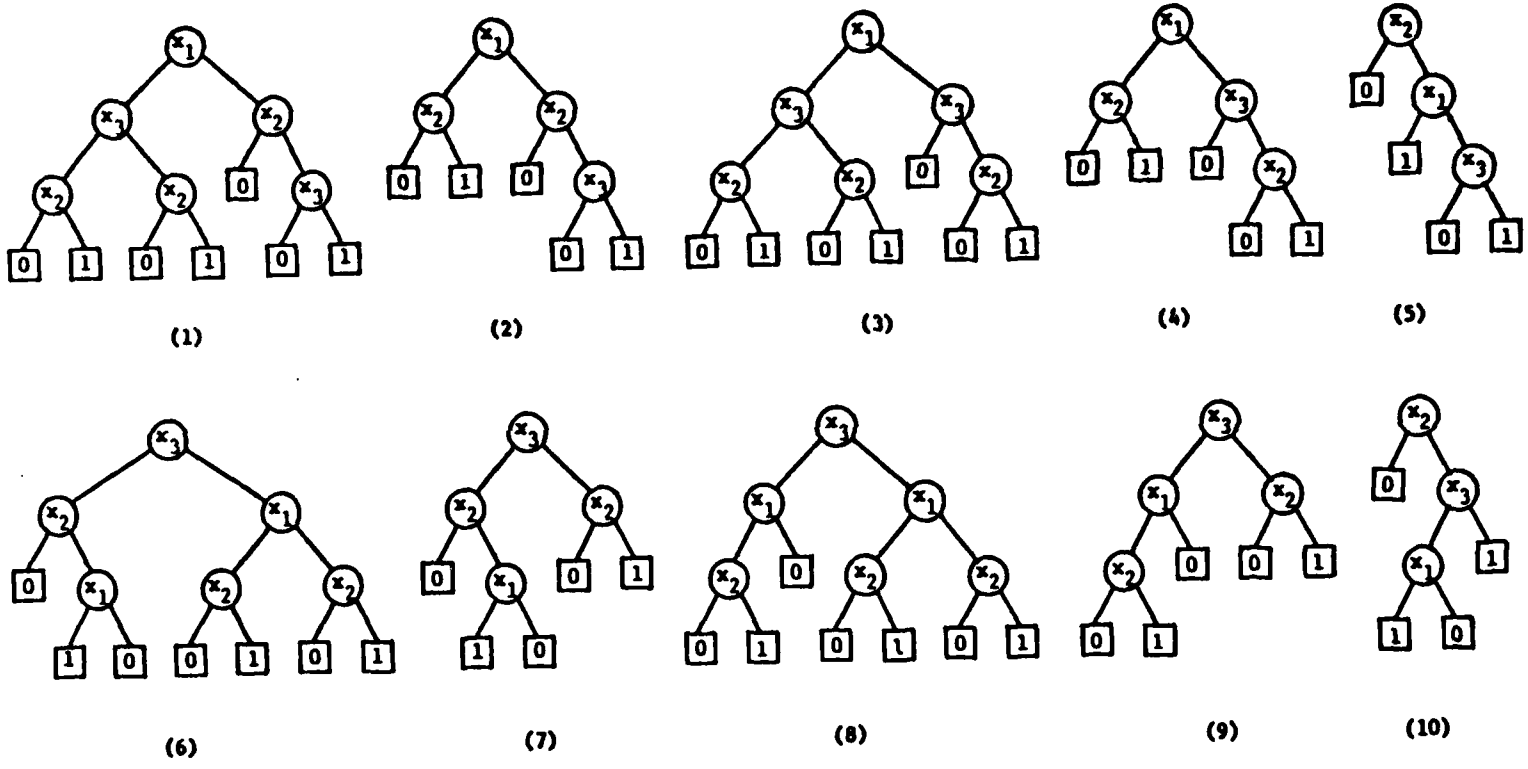


Figure 4. The ten decision trees for the function $\bar{x}_1x_2 + x_2x_3$.

Table 4. The Number of Decision Trees for Boolean Functions

n	$N_T(n)$	n	$N_T(n)$
1	1	6	$1.65 \cdot 10^{13}$
2	2	7	$1.91 \cdot 10^{27}$
3	12	8	$2.91 \cdot 10^{56}$
4	576	9	$7.64 \cdot 10^{111}$
5	1658880	10	$5.84 \cdot 10^{234}$

Not all tree or diagram representations of a function are equally desirable. Thus several criteria have been developed in order to select an appropriate representation; such criteria attempt to measure important properties of decision trees and diagrams such as their implementation and usage costs.

In the most general case, each variable has an associated *testing cost*, which measures the expense (e.g., in time) incurred each time that variable is evaluated, and a *storage cost*, which measures the expense (e.g., in memory) due to the presence of each test node labeled by that variable. In addition, a probability distribution is often specified on the variables' space, which can be assumed uniform if not otherwise known. These data allow the computation of the following measures. (These and other criteria are discussed in depth in MORE81b.)

Definition 2

(i) The worst case testing cost, h, is the maximum path testing cost. When all testing costs are unity, h reduces to the worst case number of tests, that is, the height of the tree or diagram.

(ii) The expected testing cost, E, is the expected value of the path testing cost, where the probability of a path is the sum of the probabilities of all the combinations of variables' values that select that path. When all testing costs are unity, E reduces to the expected number of tests.

(iii) The tree storage cost, α, is the sum of the storage costs of the internal nodes of the tree. When all costs are unity, α reduces to the number of internal nodes of the tree.

(iv) The diagram storage cost, β, is the sum of the storage costs of the internal nodes of the minimal diagram. □

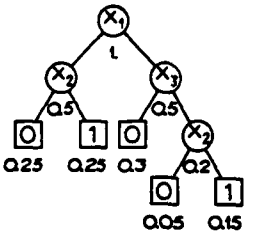


Figure 5. The decision tree of Example 7.

In the case of unity costs and uniform probability distribution, the only datum needed to compute the first three measures (those applicable to trees) is the number of leaves at each level of the tree. Thus a decision tree for a function of n variables can be characterized by an $(n + 1)$-tuple, the *leaf profile* [MORE80a], $(\lambda_0, \ldots, \lambda_n)$, where λ_i is the number of leaves at depth i. The leaf profile induces a lexicographic ordering on decision trees, thereby giving rise to two additional measures: (1) the *maximum profile*, which ranks as "best" that tree which is largest in lexicographic order (on the grounds that leaves should be encountered as soon as possible), and (2) the *minimum reverse profile*, which ranks as "best" that tree which is smallest in reverse lexicographic order (on the grounds that the number of long paths should be minimized).

Example 7

Assume storage costs s, testing costs t, and probability distribution p, for the function of Example 5:

s: $x_1 \to 1$ $x_2 \to 2$ $x_3 \to 3$

t: $x_1 \to 1$ $x_2 \to 2$ $x_3 \to 6$

p:
$(0, 0, 0) \to 0.10$ $(1, 0, 0) \to 0.05$
$(0, 0, 1) \to 0.15$ $(1, 0, 1) \to 0.05$
$(0, 1, 0) \to 0.05$ $(1, 1, 0) \to 0.25$
$(0, 1, 1) \to 0.20$ $(1, 1, 1) \to 0.15$

Figure 5 shows the tree of Figure 1 with its node probabilities. The expected testing cost, E, of the tree is

$$E = (0.25 + 0.25) \cdot (1 + 2) + 0.3 \cdot (1 + 6) + (0.05 + 0.15) \cdot (1 + 6 + 2) = 5.4.$$

The tree's profile is (0, 0, 3, 2), and its var-

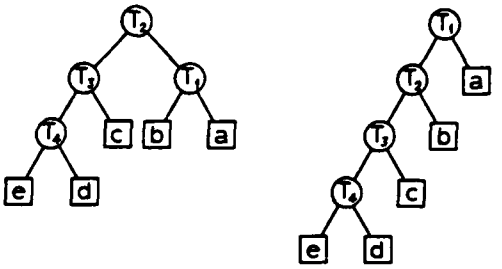


Figure 6. Two possible decision trees for the problem of Example 8.

ious measures are

Measure	Value
h (worst case testing cost)	9
Height (worst case number of tests)	3
E (expected testing cost)	5.4
Expected number of tests	2.2
α (storage cost)	8
Node count	4 □

Finally, when ascertaining the value of a variable requires a costly apparatus (or subroutine), it may be desirable to minimize the total cost incurred through the acquisition of such apparatus for each variable used in the tree; we call this criterion the *total acquisition cost*. Minimizing this cost is a common problem in biological identification [PAYR80, WILL80], where the number of tests often exceeds the number of taxa; the objective is to find the smallest subset of tests that still separates all taxa, which corresponds to minimizing the total acquisition cost when all tests have unity acquisition costs. Clearly, such a cost is fixed (and maximal) for intrinsic functions, since all variables must be tested, and thus evaluated at some point or other in the tree. In fact, this cost is better associated with the functions rather than the trees.

2.3 Binary Identification

In the simplified model of binary identification expounded by GARE72a, there is exactly one combination of test values associated with each object. Therefore decision trees for such problems have a fixed number of leaves (one per object) and thus of internal nodes (since the number of internal nodes of a binary tree is one less than the number of its leaves), so that their storage cost is simply equal to one less than the number of objects when all costs are unity. Moreover, since no two leaves are identical, there can be no common subtrees, so that decision diagrams for identification problems are decision trees.

In the even simpler case of simple binary identification, a *yes* answer immediately identifies the object and so terminates the evaluation, so that at most one path (that corresponding to the "no" answer) leads from a test to another; this corresponds to a degenerate tree with a number of internal nodes equal to its height. Only one optimization criterion, the expected testing cost, is applicable, and the optimization can be done step by step using a simple ordering of tests in terms of their cost to probability ratio [JOHN56, RIES63, SLAG64, GARE72b]. Similar conditions arise when a Boolean function is evaluated in a linear (as opposed to tree-structured) sequence [HANA77].

Example 8

Two possible decision trees for the binary identification problem of Example 4 are illustrated in Figure 6. Both trees have a storage cost of 4; their other measures are

Tree	Leaf profile	Height	Expected number of tests
Left	(0, 0, 3, 2, 0)	3	2.4
Right	(0, 1, 1, 1, 2)	4	2.8 □

3. OPTIMIZATION

In most applications, decision trees and diagrams must be constructed from function descriptions. Since, as previously observed, numerous tree representations can be built, with varying usage costs, we naturally strive to construct a decision tree which optimizes a suitable measure. This endeavor raises several questions.

(i) Since the choice of an optimization criterion can be difficult, can a tree be constructed which simultaneously optimizes several measures?
(ii) How difficult is the optimization task for each criterion?
(iii) If constructing an optimal tree is too time consuming, are there fast heuristic methods that build acceptable suboptimal trees? If so, how good are those methods?

In this section, each question is answered in turn and a survey of optimization methods provided.

3.1 Questions of Compatibility

In order to answer the first question, we examine the relationships between measures. We shall say that two optimization criteria are *compatible* if, for every function in a given family, at least one tree can be constructed that satisfies both criteria. Moret [MORE81b] has shown that, even if we restrict our attention to the family of Boolean functions with uniform costs and probabilities (the case that is least conducive to incompatibilities), all criteria are pairwise incompatible, with two exceptions: (1) the minimum height is a special case of the minimum reverse profile, and (2) the exact relationship of the number of diagram nodes with the number of tree nodes is as yet unknown. (However, it is easily shown that the minimization of one does not necessarily result in the minimization of the other.) In the more general case of discrete functions with nonuniform costs and probabilities, all criteria are pairwise incompatible [MORE81b].

Thus it appears that the six measures defined on decision trees and diagrams are essentially independent, so that we are indeed faced with a problem of choice. In order to gather more information about the possible choices, we now address the second question.

3.2 Questions of Complexity

The problem of constructing optimal decision trees and diagrams has been addressed by many researchers using branch-and-bound techniques [REIL66, REIL67, BREI75b] and dynamic programming [GARE72a, MISR72, MEIS73, BAYE73, SCHU76, PAYH77, MART78]. In the following, we review those and more specialized techniques. Each method is first introduced and its salient characteristics mentioned; a more detailed explanation follows, which the less mathematically inclined reader may wish to skip.

3.2.1 The Dynamic Programming Method

Dynamic programming is of particular interest as all of our tree measures obey the "principle of optimality," that is, they are such that an optimal solution can be built from optimal subsolutions. This is the case because the building of a decision subtree for each restriction is a separate problem that can be optimally solved independently of the others. This also tells us that our sixth measure, the diagram cost, does not obey the principle of optimality, since subdiagrams often overlap; the resulting interaction destroys the independence of the subproblems.

The general algorithm builds the optimal tree from the leaves up by identifying successively larger optimal subtrees (one for each combination of tested and untested variables). This approach is embodied in an algorithm designed to convert limited-entry decision tables to decision trees with minimal expected testing cost [BAYE73]. This solution was independently discovered by SCHU76, who generalized it to extended-entry decision tables; a refined version, using some game tree heuristics, was recently published [MART78]. A closely related procedure was developed for use in pattern recognition to minimize the worst case or the expected testing cost of binary decision trees [MEIS73, PAYH77]. The earliest version of the algorithm appears to be due to GARE70 (see also GARE72a, MISR72) in the context of binary identification problems.

For a function of n k-ary variables, the algorithm requires a number of operations proportional to $n \cdot k \cdot (k+1)^{n-1}$. Since a complete specification of the function requires $S = k^n$ items of information, the algorithm takes $O(S^{\log_k(k+1)} \cdot \log S)$ time for completely specified k-ary functions and is thus fairly efficient. For partial functions,

in particular identification problems, however; the input size may be much smaller; a binary identification problem with m tests and n objects requires the specification of m items each of size n (the answer to each test for each object) for an input size of $S = m \cdot n$. In this case, the algorithm may require time exponential in the input size, thereby being very inefficient. In fact, binary identification problems appear to be intrinsically "hard," that is, there is considerable evidence that no algorithm can be developed for them that would not require exponential time.[1]

We now examine in more detail how the dynamic programming method is applied to the optimization of decision trees and provide an example; the less mathematically inclined reader may wish to skip to the beginning of the next section.

If variable x_i, with testing cost t_i and storage cost s_i, is tested at the root of a decision tree for the function $f(x_1, \ldots, x_n)$, then the optimal values for the first three measures of Definition 2 are

$$h_{\min}(f) = t_i + \max\{h_{\min}(f|_{x_i=0}), \ldots, h_{\min}(f|_{x_i=m_i-1})\},$$

$$E_{\min}(f) = t_i + \sum_{j=0}^{m_i-1} p(x_i = j) \cdot E_{\min}(f|_{x_i=j}),$$

$$\alpha_{\min}(f) = s_i + \sum_{j=0}^{m_i-1} \alpha_{\min}(f|_{x_i=j}), \qquad (4)$$

where $p(x_i=j)$ denotes the probability that x_i takes on the value j. Similarly, the leaf profile of this tree is obtained by summing, component by component, the leaf profiles of its subtrees, then introducing an additional first component, set to 0. In a decision diagram, however, the subdiagrams representing the various restrictions usually overlap, so that the storage cost of a function is not directly related to the storage costs of its restrictions. This shows that five of our six measures indeed obey the principle of optimality.

The algorithm will generate all possible restrictions of the function. For a function of k-ary variables, each variable can be in any of $(k + 1)$ conditions (k values and the untested state) so that there are $(k + 1)^n$ distinct combinations; generating them from the bottom k^n leaves (all variables tested) to the unique top node (all variables untested) requires a number of steps equal to

$$\sum_{i=0}^{n} (n-i) \cdot \binom{n}{i} \cdot k^{n-i} = n \cdot k \cdot (k+1)^{n-1}, \qquad (5)$$

since a node with i untested variables has $n - i$ possible parents (each with one more untested variable) and can be chosen in $\binom{n}{i} \cdot k^{n-1}$ ways. Therefore, using the "big Oh" notation of algorithm analysis [WEID77], the algorithm takes $O(n \cdot k \cdot (k+1)^{n-1})$ time.

A restriction with $n - i$ untested variables determines a subspace of k^i points, which we call an *i-subcube*. The algorithm starts by considering all 0-subcubes (that is, all points in the variables' space), then forms all possible 1-subcubes by merging k 0-subcubes, in effect letting one variable be undetermined (so that a unique variable is associated with each merging). This process continues, forming all i-subcubes by merging $(i - 1)$-subcubes, until the final n-subcube (the complete space) is formed. Each subcube is identified by an n-tuple of values, $(i_1, \ldots, i_n)$, where i_j is X if the jth variable is untested at that node, and is the variable's value otherwise. For instance, the 0-subcubes identified by $(0, 1, \ldots, 1)$, $(1, 1, \ldots, 1), \ldots, (k - 1, 1, \ldots, 1)$ can be merged into the 1-subcube given by $(X, 1, \ldots, 1)$ by letting variable x_1 be untested. Thus the process builds a lattice of $(k + 1)^n$ nodes with $n \cdot k \cdot (k + 1)^{n-1}$ edges. It is noted that the same i-subcube can be formed by merging one of i distinct k-tuples of $(i - 1)$-subcubes.

As the lattice is built, each node (i.e., each subcube) is assigned a cost and a value; if we are interested in the expected testing cost, the probability of each node is also computed. The probability of an i-subcube is just the sum of the probabilities of the k merged $(i - 1)$-subcubes; the value of the function for an i-subcube is that of the k merged $(i - 1)$-subcubes, if their function values were identical, and is "?" (a special

[1] Technically, they are NP-hard [GARE79] problems, as proved in HYAF76 and LOVE79; for a detailed discussion of the exact complexity, the reader is referred to MORE81b.

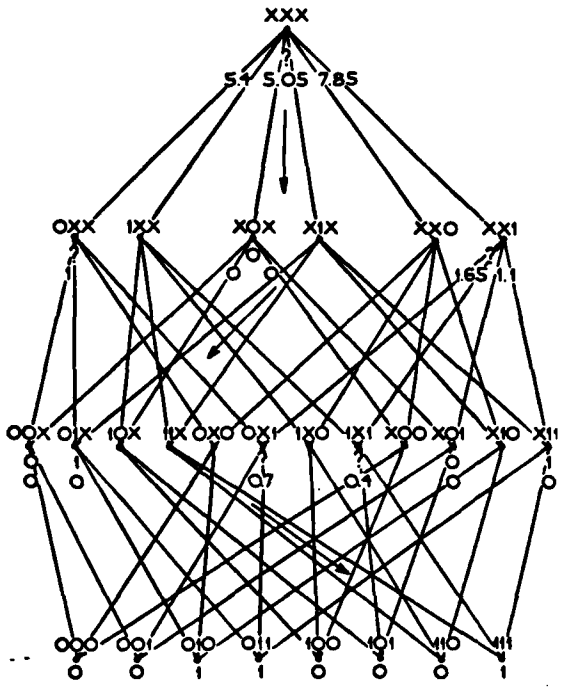


Figure 7. The dynamic programming lattice of Example 9; all initial costs zero.

symbol indicating that the value is a nonconstant function) otherwise. One easily verifies that these quantities are well defined, that is, independent of the choice of the merged subcubes. Finally, the cost assigned to an i-subcube is the minimum cost of merging, where the merging of k $(i-1)$-subcubes has cost 0 if all k subcubes have identical known values, and is equal to one of the equations (4) otherwise. Initial conditions are given by 0-subcubes with values and probabilities given by the problem. The cost of the top node (the n-subcube) is just the cost of the optimal tree for the function; the tree itself can be recovered by walking down the lattice, choosing at each step to test the variable that gave rise to the least costly merging, until nodes with a known value (leaves) are reached.

Example 9

We shall make use of the function of Example 7 to show how the dynamic programming algorithm produces a decision tree with minimal expected testing cost. With 3 variables, the variables' space has $2^3 = 8$ points, each identified by a unique combination of the 3 variables. The algorithm will build a lattice of 3^n nodes with $n \cdot 3^n$ pairs of edges, as shown in Figure 7. Since we are interested in the expected testing cost, we keep track at each node of the function's value, the node's probability, and the merging costs, where the cost of merging two $(i-1)$-subcubes is 0 if both subcubes have identical known values, and is equal to

$$c \cdot (p_1 + p_2) + c_1 + c_2$$

otherwise, where c_1, p_1 (c_2, p_2) are the cost and probability of the first (second) $(i-1)$-subcube, respectively, and c is the testing cost of the variable used in merging. Since the cost of the least expensive merging which produces the top node is 5.05, that is the cost of the optimal tree for our problem. The large arrows in Figure 7 show which merging was least expensive at each step and allow the recovery of the optimal decision tree, in this case, the linear testing sequence $x_1x_2x_3$ (the fifth tree in Figure 4). □

3.2.2 The Branch-and-Bound Method

Our sixth measure, the diagram storage cost, is not amenable to a solution by dynamic programming, since it supposes iden-

tification of common subtrees and thus a global (as opposed to dynamic programming's local) view of the subproblems. Thus optimization of diagram storage cost is done by search techniques, principally branch-and-bound [REIL67], a method which has also been applied to the optimization of the expected testing cost [REIL66, BREI75b].

Recall that a branch-and-bound algorithm proceeds by always developing that partial solution which is potentially less expensive than any other (as determined by a lower bound function), often switching from one partial solution to another when lower bounds change, until one solution has been completely developed [LAWL66]. The lower bound function used for the diagram and tree storage costs [REIL67] is based upon this simple fact: a nonredundant variable must appear at least once in any diagram. Thus a rough lower bound can be derived by simply summing the storage costs of all the nonredundant variables. The lower bound used for the expected testing cost can be derived in an analogous fashion by considering the a priori probability that each variable will be tested in any decision tree representation and modifying it to reflect the influence of the choice of a root [REIL66, BREI75b, MORE80b], or it can be derived from first principles as a complexity measure on decision trees [MORE80a].

The principal disadvantage of the branch-and-bound method is that it may result in a near-exhaustive search of the possible trees and diagrams, a process that, in view of the dimension of the search space (as previously discussed), leads to intolerably long computations. In terms of algorithm analysis, the branch-and-bound technique is an exponential-time algorithm, regardless of the input size.

We now examine in more detail the bounding functions mentioned above and illustrate the use of branch-and-bound methods by a simple example. Again, the less mathematically inclined reader may wish to skip to the next section.

A lower bound on the storage cost of a decision tree must incorporate the influence of the choice of a given variable to be of use in the branch-and-bound process. To achieve this end, the lower bound is modified by using a one-level "look-ahead"; if variable x_i is chosen for the root of the tree representing function f, then the lower bound is the sum of

(i) the storage cost of the chosen variable, x_i;
(ii) the storage cost of each x_j, $j \neq i$, times the number of restrictions, $f|_{x_i=k}$, for which x_j is nonredundant.

For diagrams, the multiplicative factor in (ii) is modified to take into account the fact that x_j may play exactly the same role for some restrictions, that is, that for every combination of values of the remaining variables, either all the restrictions are equal or at most one is not constant.

The lower bound on the expected testing cost of a decision tree can be derived from first principles by considering the development of a measure of the influence of a variable on the expected testing cost of decision tree representations. Any such measure should possess the following two properties:

(i) the measure is minimal (equal to zero) when the variable is redundant and maximal (equal to the variable's testing cost) when the variable is indispensable;
(ii) the measure is compatible with the tree structure, that is, if we denote such a measure for the variable x_i by $a_f(x_i)$, it must be the case that, for each $j \neq i$,

$$a_f(x_i) = \sum_{k=0}^{m_j-1} p(x_j = k) \cdot a_{f|_{x_j=k}}(x_i).$$

Moret [MORE80a] has shown that only one measure satisfies those two conditions: the *activity* of a variable, which is equal to the testing cost of the variable times the a priori probability that it will be tested (a concept related to the Boolean difference used in Boolean algebra [THAY81b]). The a priori probability that variable x_j will be tested is just the probability that, with all its other variables evaluated, the function still depends on x_j; this is easily computed in linear time. The lower bound used in REIL66 and BREI75b can then be defined as the sum of

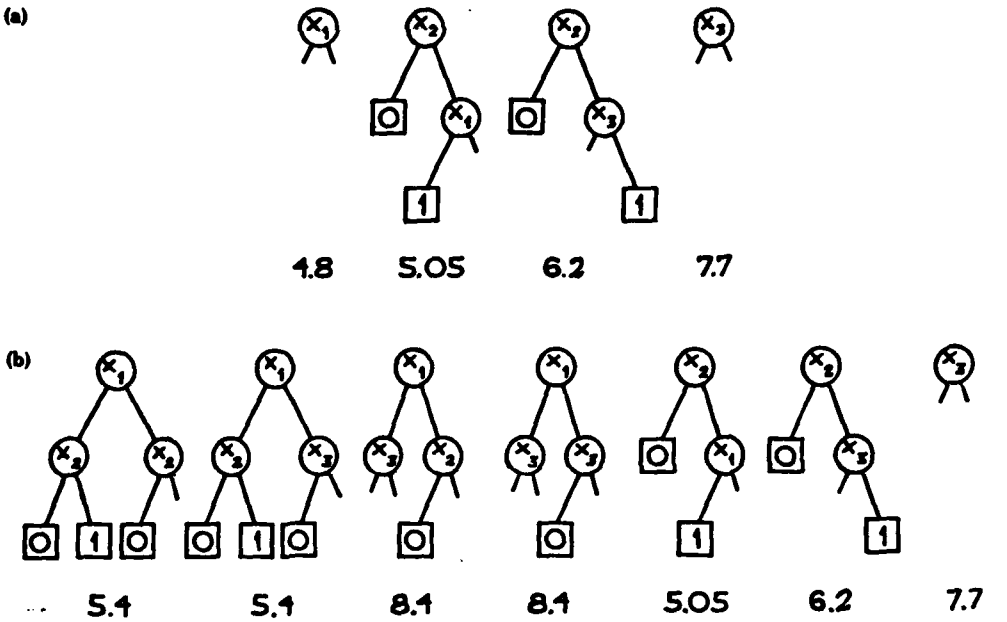


Figure 8. The partial trees as developed by the branch-and-bound method for the function of Example 10: (a) after the first stage; (b) after the second stage.

the testing cost of the root variable and the activities of the remaining variables [MORE80b].

Example 10

Consider again the Boolean function of our previous examples. The activities of the three variables are found to be

$$a_f(x_1) = 1 \cdot \text{prob}(x_2 = 1 \text{ and } x_3 = 0) = 0.3,$$

$$a_f(x_2) = 2 \cdot \text{prob}(x_3 = 1 \text{ or } x_1 = 0) = 1.4,$$

$$a_f(x_3) = 6 \cdot \text{prob}(x_1 = 1 \text{ and } x_2 = 1) = 2.4.$$

Now the lower bound on the expected testing cost of a decision tree for f with root x_i, $lb_f(x_i)$ can be computed for each variable

$$lb_f(x_1) = t(x_1) + a_f(x_2) + a_f(x_3) = 4.8,$$

$$lb_f(x_2) = t(x_2) + a_f(x_1) + a_f(x_3) = 4.7,$$

$$lb_f(x_3) = t(x_3) + a_f(x_1) + a_f(x_2) = 7.7.$$

Thus the branch-and-bound algorithm chooses to develop the tree rooted in x_2. The left subtree is then a leaf labeled 0 so that only two possible partial trees arise, depending on the choice of the variable tested at the root of the right subtree. Lower bounds are computed in turn for these. We now have four partial subtrees, pictured with their lower bounds in Figure 8a. The algorithm will choose to develop the first partial tree (rooted in x_1), since it is now the least expensive; this yields four partial trees for a total of seven partial trees, pictured with their lower bounds in Figure 8b. Now the algorithm will return to the fifth tree (rooted in x_2) and complete it; since its final cost, 5.05, is lower than the bound on any partial tree, the completed tree is optimal. □

3.2.3 *Other Methods*

In the context of logic—particularly Boolean—functions, specialized methods have been devised which attempt to use some of the standard tools of logic (such as reduction to canonical or minimal formulas and decompositions) in order to construct decision trees and diagrams with minimal

storage or expected testing cost [MICH78, THAY78, CERN79a, THAY81a].

The minimization of storage or expected testing cost for decision trees has been approached for Boolean [CERN79a] and multivalued [THAY78] logic functions by considering a special class of subfunctions, which the authors call T-terms. Starting with the terms of order 0, which are just logic cubes (in the sense of switching theory [HARR65]), terms of successively higher order are constructed by consensus operations. (That is, for each variable, x_j, one takes the consensus of a T-term of order n and one of order $k \leq n$ with respect to x_j; the result is a T-term of order $n + 1$ if it is not already a T-term of lower order and if it is independent of x_j.) Only the prime terms are kept in the construction process (where a prime term is a term not contained in any other term of the same order). A simple procedure is then used to derive a tree optimal with respect to worst-case testing cost, expected testing cost, or storage cost. Although the algorithm sheds light on the relationship between Boolean formulas and optimal decision trees, it is not of practical interest (expect for minimizing diagram storage cost) since it contains a hard problem; the prime terms of order 0 are just the *prime implicants* of the function [HARR65] and obtaining them, even from complete function descriptions, is known to be NP-hard [MASE82]. As a result, the procedures proposed may require exponential time under any input size. A similar drawback is present in the algorithm published by MICH78, which converts extended-entry decision tables to decision trees with minimal storage or expected testing cost, since the algorithm starts by establishing a minimal disjoint set cover for the function, a process known to be NP-hard [GARE79].

An extension of T-terms was recently proposed by THAY81a for the design of decision trees and diagrams with minimal storage cost. This formulation is based on a class of functions called *P-functions*, where a P-function for the Boolean function f is a pair, $\langle g, h \rangle$, of functions such that f and h are equal when restricted to the points where g evaluates to 1. A composition operation is defined that allows the building of a lattice of P-functions, from the lowest order (with $h = 1$ or $h = 0$) to the highest order (with $g = 1$). Again, only prime P-functions are retained in building the lattice. A search procedure generates optimal decision diagrams from the lattice of prime P-functions. The generation of optimal decision trees, however, requires that prime P-functions be replaced by prime P-cubes, that is, prime P-functions restricted so that they are logic cubes. Since the lowest order P-cubes comprise the prime implicants of the function and its complement, we find anew the NP-hard subproblem, so that the synthesis of optimal decision trees by P-functions requires exponential time and is therefore of little practical interest. (It must be noted that we do not imply that finding prime implicants is an unsurmountable task; in fact, several algorithms for that purpose have been studied and shown to do well in practice [SLAG70, HULM75]. Our point is that the methods described above, which incorporate this NP-hard problem as only a small part of the complete work, cannot compare with the dynamic programming method.) On the other hand, the construction of optimal diagrams, while exponential (no analysis is provided with the algorithm, but the generation of all prime P-functions may clearly require that much time), remains of interest since no substantially better solution is known.

We now proceed to a closer examination of the composition of P-functions, followed by an example. Once again, the less mathematically inclined reader may wish to skip to the next section.

If f has n variables, it has up to $2^{2^n} - 1$ P-functions on which a lattice structure can be established, from the lowest to the highest order by means of the following noncommutative composition law (designed to be compatible with Shannon's decomposition). A P-function of order $k + 1$, $\langle g, h \rangle$, is obtained from two P-functions of lower order (one of order k and the other of order no larger than k), $\langle g_0, h_0 \rangle$ and $\langle g_1, h_1 \rangle$, by using, for each x_i, the formula

$$\langle g, h \rangle = \langle g_0|_{x_i=0} \cdot g_1|_{x_i=1}, \overline{x}_i \cdot h_0 + x_i \cdot h_1 \rangle, \quad (6)$$

which we denote $\langle g_0, h_0 \rangle \odot_i \langle g_1, h_1 \rangle$. In a sense, the resulting P-function of order

$k + 1$ corresponds to our state of knowledge prior to testing variable x_i. Indeed, substituting $x_i = 0$ into $\langle g, h \rangle$ yields

$$\langle g_0|_{x_i=0} \cdot g_1|_{x_i=1}, h_0 \rangle,$$

and substituting $x_i = 1$ yields

$$\langle g_0|_{x_i=0} \cdot g_1|_{x_i=1}, h_1 \rangle,$$

which shows how the second function in the pair reflects our increased knowledge about the function f (until that second function is a constant, meaning that the evaluation of f is complete), while the first function provides us with information about the path of evaluation followed so far.

Example 11

Consider the Boolean function of our previous examples. The two prime P-functions of order 0 are

$$A_0 = \langle f, 1 \rangle \quad \text{and} \quad A_1 = \langle f, 0 \rangle.$$

From those two functions, we can form six P-functions of order 1, three of which are prime:

$$B_0 = A_0 \odot_1 A_1 = \langle x_2\overline{x_3}, \overline{x_1} \rangle,$$
$$B_1 = A_1 \odot_2 A_0 = \langle \overline{x_1} + x_3, x_2 \rangle,$$
$$B_2 = A_1 \odot_3 A_0 = \langle x_1x_2, x_3 \rangle.$$

Using now the five prime P-functions of order 0 and 1, we can form 54 P-functions of order 2 (not all distinct), four of which are prime:

$$C_0 = A_0 \odot_1 B_2 = B_0 \odot_3 A_0 = \langle x_2, \overline{x_1} + x_3 \rangle,$$
$$C_1 = A_1 \odot_2 B_1 = B_1 \odot_2 A_0 = \langle \overline{x_1} + x_3, x_2 \rangle,$$
$$C_2 = A_1 \odot_3 B_1 = \langle x_1 + \overline{x_2}, x_2x_3 \rangle,$$
$$C_3 = B_1 \odot_1 A_1 = \langle \overline{x_2} + \overline{x_3}, \overline{x_1}x_2 \rangle.$$

Finally, we can use our nine prime P-functions of orders 0, 1, and 2 in order to obtain the single prime P-function of order 3 at the top of the lattice, $D_0 = \langle 1, f \rangle$:

$$D_0 = A_1 \odot_2 C_0 = C_2 \odot_2 C_0 = C_3 \odot_2 C_0 = C_1 \odot_1 C_2 = B_1 \odot_1 C_2 = C_3 \odot_3 C_1 = C_3 \odot_3 B_1.$$

The corresponding lattice is shown in Figure 9, where a number, i, in a circle has been used to denote a composition with respect to the ith variable. A search through the lattice shows that the optimal diagrams have three nodes for a storage cost of 6; the diagram of Figure 2 was one of those. □

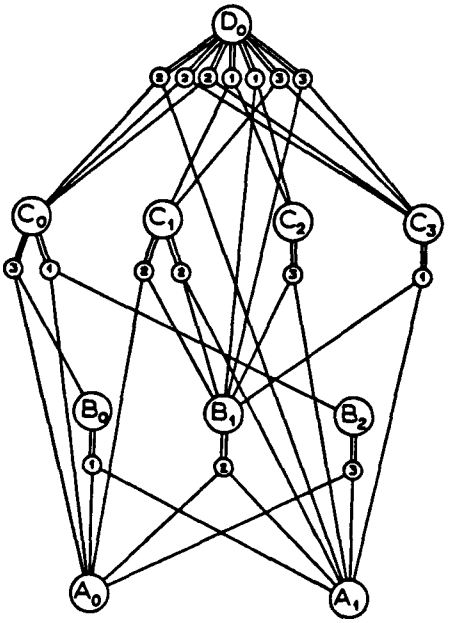


Figure 9. The lattice of prime P-functions of Example 11.

3.3 Questions of Optimality

Since decision tree optimization is an NP-hard problem under polynomial-size inputs [MORE80b], it is necessary to develop some heuristics that will allow the fast construction of good, albeit not optimal, solutions. Indeed, some such heuristics have been proposed even before an optimal algorithm was developed; it appears that the so-called "splitting" heuristic, discussed below, was known to Aristoteles and Theophrastus [Voss52], and many heuristics were proposed for the conversion of decision tables [MONT62, EGLE63, POLL65] before the publication of the branch-and-bound solution of REIL66.

All published heuristics are of the so-called *greedy* type, that is, they perform a local, step-by-step optimization. Three main types of criteria are used; the apparently large variety results from attempts to accommodate tests with variable out-

comes (as in most biological applications [BROW77, GOWE75, PAYR81]), or from minor differences in preprocessing (such as attempts to put decision tables in a canonical form [SHWA75]), or in the extent of look-ahead used (while most strategies use no look-ahead at all, SETH80 has suggested a one-step look-ahead and MICH78 proposed a range of look-aheads, from 0 to k step). These three criteria are discussed in some detail below. (Since the discussion of the first—and most important—of these criteria involves some mathematical manipulations, we once again have organized its discussion in two parts, grouping all mathematical concepts in the second.)

3.3.1 The Information-Theoretic Criterion

In this strategy, commonly used for the (near) minimization of the expected testing cost, the problem is viewed as one of refining an initial uncertainty about the function's value into a certitude. At each step, the test of a variable diminishes the universe of possibilities, thereby removing a certain amount of ambiguity. In information-theoretic terms, the initial ambiguity of a (partial) function is expressed by the *entropy* of the function (for a lucid exposition on the topic, see the original paper of SHAN48). The ambiguity remaining after testing a variable can be computed as the average ambiguity among the restrictions. This allows the computation of the ambiguity removed (or, equivalently, the information gained) by testing that variable. The *information heuristic* then chooses at each step that variable which removes the most ambiguity per unit testing cost.

The previously mentioned *splitting* heuristic is a special case of the information heuristic for identification problems. It is well known [SHAN48] that the removed ambiguity is maximized by letting all restrictions have equal probability; thus, when all variable costs are unity, the information heuristic chooses that variable which "splits" the set of objects into subsets with most nearly equal probabilities. In the case of binary identification problems with equally likely objects and unity costs, this means selecting that test which splits the objects into subsets of most nearly equal size. Yet another aspect of the splitting heuristic is a criterion used for binary identification problems [GYLL63, CHAN65], which selects that variable which separates the largest number of pairs of values: if n is the total number of values and k the number of values put into one subset, then $k \cdot (n - k)$ pairs are split. This criterion is optimized for $k = n/2$. (Choosing that variable which separates the largest number of pairs could be described as the *separation* heuristic; this criterion also appears in a variety of forms, some of which attempt to include tests with variable outcomes [BROW77, PAYR80].)

Descriptions of the heuristic and its variants abound [KLET60, RESC61, OSBO63, MAND64, GOWE71, GANA73, SHWA74], but it was not until later that the performance of the information heuristic was analyzed. Garey [GARE74] studied its application to identification problems and showed that, although it is quasi-optimal when all possible tests are available (a result dating from ZIMM59), there are problems for which it can construct trees with an expected testing cost arbitrarily larger than the optimal. This result disproved a long-standing conjecture that the splitting algorithm was optimal for identification problems with equally likely objects [KLET60, OSBO63]; such a result is, of course, predictable now in view of the NP-hardness of the problem since an optimal polynomial algorithm would disprove the widely held opinion that NP-hard problems actually require exponential solutions. However, HUNG74 proved that the heuristic is, on the average, asymptotically optimal; that is, the ratio of the average cost of trees built with the information heuristic to that of the optimal trees converges to 1 as the problems get larger. This result must be qualified by the observation that most functions have an expected testing cost fairly close to maximum, so that the asymptotic ratio used in HUNG74 is in general fairly small for any heuristic. Indeed, MORE81b showed that completely specified Boolean functions of n variables with unity costs and uniform probability distribution have an asymptotic expected testing cost of $n - 1$, so that in this case the asymptotic average ratio must be 1 for any heuristic. Recently, HART82

presented a generalization of the information heuristic which allows for more complex objective functions (including the acquisition cost) and offers a trade-off between the complexity of the construction and the upper bounds that can be placed on the size of the resulting solution.

The information heuristic is efficient: for an input size of $O(S)$, it takes time $O(S \cdot \log S)$ which, while barely faster than the optimal dynamic programming solution for exponential size inputs, compares very favorably indeed with the exponential-time algorithms used for identification problems.

We now examine the entropy computations in some detail; once again—and for the last time—the less mathematically inclined reader may wish to skip to the next section.

The expression for the entropy of a function f is [SHAN48]

$$H_f = -\sum_v p(f = v) \cdot \log_2 p(f = v), \quad (7)$$

where $p(f = v)$ is the probability that f takes the value v and the sum is taken over all the values v in the range of f (values of $p(f = v)$ are normalized so that their sum over the values of v is equal to 1). After testing variable x_i, the remaining ambiguity, $H_f(x_i)$, is the average ambiguity among the restrictions:

$$H_f(x_i) = \sum_{j=0}^{m_i-1} [p(x_i = j) \cdot H_f|_{x_i=j}].$$

Hence the ambiguity removed by testing x_i is the quantity

$$I_f(x_i) = H_f - H_f(x_i). \quad (8)$$

This quantity is computed for each variable; the information heuristic then chooses at each step that variable which has the greatest ratio, $I_f(x_i)/t_i$.

For identification problems, the expression for the removed ambiguity can be simplified to

$$I_f(x_i) = -\sum_{j=0}^{m_i-1} p(x_i = j) \cdot \log_2 p(x_i = j), \quad (9)$$

which is seen to be of the same form as (7). Thus maximizing the removed ambiguity in an identification problem involves finding that variable x_i such that the probabilities $p(x_i = j)$ are most nearly equal to each other.

Example 12

Let us use once more the function of Example 7. The entropy of the function is

$$\begin{aligned} H_f &= -[p(f = 0) \cdot \log_2 p(f = 0) \\ &\quad + p(f = 1) \cdot \log_2 p(f = 1)] \\ &= -[0.6 \cdot \log_2 0.6 + 0.4 \cdot \log_2 0.4] \cong 0.971. \end{aligned}$$

The ambiguity remaining after testing x_1 is

$$\begin{aligned} H_f(x_1) &= p(x_1 = 0) \cdot H_f|_{x_1=0} \\ &\quad + p(x_1 = 1) \cdot H_f|_{x_1=1} \\ &\cong 0.5 \cdot 1 + 0.5 \cdot 0.881 \cong 0.941. \end{aligned}$$

Thus the information gain of x_1 per unit testing cost is

$$\frac{I_f(x_1)}{t(x_1)} \cong \frac{(0.971 - 0.941)}{1} = 0.03.$$

Similarly, we find

$$\frac{I_f(x_2)}{t(x_2)} \cong \frac{0.971 - 0.625}{2} = 0.173,$$

$$\frac{I_f(x_3)}{t(x_3)} \cong \frac{0.971 - 0.652}{6} \cong 0.053,$$

so that the heuristic will choose to test x_2 first. Since the restriction for $x_2 = 0$ is constant, only the restriction for $x_2 = 1$ remains. The information gains per unit cost of x_1 and x_2 for that restriction are

$$\frac{I_f|_{x_2=1}(x_1)}{t(x_1)} \cong \frac{(0.961 - 0.382)}{1} = 0.579,$$

$$\frac{I_f|_{x_2=1}(x_3)}{t(x_3)} \cong \frac{(0.961 - 0.195)}{6} \cong 0.128,$$

so that x_1 is tested next. The completed tree is in this case the optimal tree developed previously. □

3.3.2 *The Activity Criterion*

A class of simple heuristics can be obtained for any problem by using the bounding functions of branch-and-bound algorithms and doing local optimization on their basis, in effect using branch-and-bound without backtracking. This approach is of no particular interest for the minimization of storage cost (although it has been used for that purpose [RABI71, YASU71]) since the exist-

ing bounds are too loose; it is, however, applicable to the minimization of the expected testing cost, since the lower bound based on activity is in general tighter.

The activity heuristic has the same computational requirements as the information heuristic. Moret [MORE80b] provided an analysis of its performance, showing that the worst-case ratio for completely specified Boolean functions with unity costs and uniform probabilities is limited to 2, but can be arbitrarily large if nonuniform probabilities are allowed. This heuristic appears to be of less interest than the information heuristic, since it performs best for "dense" problems, that is, those in which the function is specified on most of its domain, which are precisely those problems that can be efficiently solved by the optimal dynamic programming algorithm.

Finally, a similar approach has been taken by some authors for biological identification problems, using rough lower and upper bounds on the number of tests needed to complete an identification (see the thorough studies of BROW77 and PAYR81). In one such approach, it is postulated that the subtree will be completed optimally (i.e., following Huffman's procedure—even though only a small proportion of all tests is available); the resulting lower bound is used for deriving a selection criterion [DALL74, BROW77]. Alternatively, it is assumed that the tree will be completed by a linear sequence of simple tests; the resulting estimate is an upper bound under most conditions and allows the derivation of another selection criterion [PAYR81]. However, those criteria are based on rather simplistic bounds and thus susceptible to large errors; despite the lack of either theoretical or practical results about their performance, one can safely predict that their average performance is worse than that of the other criteria examined so far.

3.3.3 *Ad Hoc Criteria for Boolean Functions*

Since Boolean functions are conveniently expressed by formulas, special heuristics can be developed that are based on characteristics of the formulas such as number of terms or literals.

One such heuristic, proposed by HALP74 for the minimization of the expected testing cost, necessitates the generation of all prime implicants for the function and its dual; the implicants are then ranked in terms of their probability to cost[2] ratio. Variables which appear in both the best implicant for the function and that for its dual are then selected (at least one such variable must exist). Halpern [HALP74] proved that this strategy is optimal for symmetric functions (those that remain invariant under any permutation of the variables), but offers no analysis of performance in the general case.

Breitbart [BREI75a] presented a similar heuristic for monotone Boolean functions with unity costs and uniform probability distribution, which uses the minimal disjunctive form of the function (this form is unique for monotone functions [HARR65]); in a later analysis [BREI78], it was shown that trees constructed by this rule can have an expected number of tests at least $(n/\log n)$ times larger than the optimal trees for functions of n variables.

Both heuristics apply only to completely specified Boolean functions and require exponential time since the generation of all prime implicants and/or the minimization of the disjunctive form are NP-hard problems [MASE82]. Since dynamic programming offers an $O(S^{\log_2 3} \cdot \log S)$ optimal solution to the same problem, these heuristics are of interest only when the function is already specified by its prime implicants or its minimal disjunctive form.

4. APPLICATIONS

In this section, we describe the main fields of application and review related results. We distinguish four fields: (1) decision table programming; (2) diagnosis, identification, and pattern recognition; (3) logic and program design; and (4) analysis of algorithms. Of these, only the last three are treated, since decision table programming is chiefly concerned with the construction of optimal decision trees, a topic with which we dealt in the previous section.

[2] The cost of an implicant is that of the optimal tree for it; that tree is easily constructed [RIES63, SLAG64] since tree representations of conjunctions of variables are just linear test sequences.

4.1 Diagnosis, Identification, and Pattern Recognition

Garey [GARE72a] argues that most identification problems of human origin include a large number of simple tests (of the type, "Is the unknown object of type *i*?") plus a smaller number of "well-splitting" tests. Indeed, this is how most of us approach the typical identification problem of "twenty questions," starting with general, well-splitting questions (e.g., "Is it mineral?") and ending the game with simple questions (e.g., "Is it an aardvark?"). Several large identification problems approximate this description, notably in botanical and biological classification [MOLL62, PANK70, MORS71, WILL80]. Garey [GARE72a] has described a dynamic programming algorithm especially designed for this type of problems that constructs identification trees with minimum expected testing cost.

In most identification problems, many more tests are present than are needed for the identification of all objects. In consequence, several researchers have studied the problem of obtaining a minimal set of tests; we recognize in this problem the minimization of the total acquisition cost. Such an optimization is important when individual tests are time consuming and promptness in identification essential (as in medical diagnosis [PAYR80, WILL80]), so that parallel, rather than sequential, testing is used. Unfortunately, the *minimum test set* problem, as it is known, is itself NP-hard [GARE79]. As a result, splitting heuristics based on the number of split pairs have been proposed for the construction of suboptimal solutions [GYLL63, CHAN65]; however, no performance analysis was supplied. (The general analysis of the splitting algorithm presented in the previous section does not apply here since the goals are quite distinct.) Some preliminary analytical results as well as extensive experimental data can be found in MORE82. This problem is closely related to that of finding prime implicants for functions of Boolean variables, since each minimal set of prime implicants determines an irredundant set of tests for the problem. Hence methods have been proposed that first find all prime implicants and then attempt to find a set of prime implicants that minimizes the number of distinct variables used (see the review of PAYR80, pp. 261–263).

Since decision trees model multistage branching decision processes, they find a particularly important application in sequential, or more precisely, hierarchical, pattern recognition (see KANA79 for some general considerations about the advantages of hierarchical approaches). In the simplest case, a pattern recognition problem is deterministic and reduces to an identification problem. In general, however, a type (called a *class*) of objects is not absolutely characterized by selected combinations of test values (called *features*); rather, the problem is of a statistical nature such that each combination of features is distributed among all the classes. (This model of probabilistic identification also corresponds to the fuzzy decision tables discussed in KAND80.) If the set of features has sufficient power of discrimination, the probability distribution of each combination of features will exhibit one strong peak for some class, so that an object possessing this combination of features can be classified in that class with a low probability of error. At times, however, it may be advantageous to trade accuracy for speed and allow an object to be classified in a patently wrong class in order to gain on response time. As a consequence of this additional freedom, decision trees for pattern recognition purposes are subject to yet another optimization criterion: minimum overall probability of misclassification.

Example 13

Consider the following simplified pattern recognition problem with three classes, C_1, C_2, C_3, and three binary features, x_1, x_2, x_3. The testing costs of the three variables (features) are

$$t: \quad x_1 \to 1 \quad x_2 \to 2 \quad x_3 \to 3,$$

and the probability distribution on the variables' space is given by

$$\begin{array}{lll} p_v: & (0,0,0) \to 0.10 & (1,0,0) \to 0.20 \\ & (0,0,1) \to 0.20 & (1,0,1) \to 0.10 \\ & (0,1,0) \to 0.10 & (1,1,0) \to 0.10 \\ & (0,1,1) \to 0.05 & (1,1,1) \to 0.15 \end{array}$$

The distribution of each combination of features among the classes is given below,

where the probability that a given combination corresponds to class i is given by the ith value of the triple:

p_c: (0, 0, 0) → (0.10, 0.85, 0.05)
(0, 0, 1) → (0.01, 0.98, 0.01)
(0, 1, 0) → (0.20, 0.10, 0.70)
(0, 1, 1) → (0.80, 0.10, 0.10)
(1, 0, 0) → (0.80, 0.10, 0.10)
(1, 0, 1) → (0.90, 0.05, 0.05)
(1, 1, 0) → (0.05, 0.05, 0.90)
(1, 1, 1) → (0.10, 0.00, 0.90)

The two distributions allow us to compute the a priori probability of each class, that is, the probablity that a unknown object belongs to that class:

$$p(C_1) = 0.342$$
$$p(C_2) = 0.326$$
$$p(C_3) = 0.332$$

Since each combination of features must be classified in some class, the strategy that minimizes the probability of misclassification is obviously to classify a combination of features in the class for which it shows the largest probability; thus we get the assignment

f: (0, 0, 0) → C_2 (1, 0, 0) → C_1
(0, 0, 1) → C_2 (1, 0, 1) → C_1
(0, 1, 0) → C_3 (1, 1, 0) → C_3
(0, 1, 1) → C_1 (1, 1, 1) → C_3

Hence the probability of misclassification of an object with the combination of features (0, 0, 0) is 0.10 + 0.05 = 0.15; since that combination of features occurs with a probability of 0.10, it contributes a total of 0.10·0.15 = 0.015 to the overall minimal probability of misclassification. Working similarly with the other combinations of features, the latter probability is found to be

$$p_{e_{min}} = 0.134.$$

A decision tree with that overall probability of misclassification is shown in Figure 10a, together with the probability of its leaves; this tree has an expected testing cost of 4. A different tree is pictured in Figure 10b; this tree classifies the combination (0, 1, 1) in class C_3—clearly not an optimal choice in terms of classification accuracy. However, this tree has an expected testing cost of only 2.6, and its overall probability of misclassification is found to be 0.164, barely larger than optimal. Thus it is a difficult task to decide which tree is best; other criteria must be used, such as penalties due to misclassification or maximum permissible response time. □

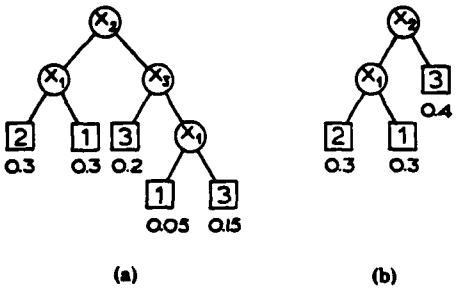


Figure 10. The two decision trees for the pattern recognition problem of Example 13: (a) with minimum possibility of error; (b) with trade-off for more efficiency.

Faced with such a variety of design criteria, researchers in the field have explored different routes. A tree is often synthesized directly from the problem without attempt to optimize its testing cost, but using heuristics designed to minimize the probability of misclassification [YOU76, ROUN79]. Hauska [HAUS75] and Wu [WU75] described a local optimization algorithm, based on measures of interclass separation, for the semiautomatic design of a decision tree from a known set of features. When the features are known in advance, a procedure based on dynamic programming due to DATT81 can be used to build a decision tree with minimal cost, in which the cost criterion includes the expected testing cost and the cost of misclassification; a similar approach based on game theory was described in SLAG71 and a third in KULK76. When the class assignments are made (and the probability of misclassification therefore fixed), the pattern recognition problem reduces to the description of a (partial) function; BELL78 models this case by decision tables and discusses their conversion to sequential testing procedures (although the optimal algorithm of BAYE73, PAYH77 is not mentioned). The accuracy of decision tree classifiers depends upon such consid-

erations as sample statistics (e.g., sample size) and the number and intercorrelation of features; KULK78 studied the problem under simplified assumptions and concluded that hierarchical classifiers, as their single-stage counterparts, may suffer from the "dimensionality" problem, that is, show decreasing performance if the number of features is increased beyond a certain threshold (which depends on the sample size).

4.2 Logic and Program Design

We mentioned previously that decision trees and diagrams for Boolean functions naturally give rise to multiplexer implementations. Such implementations are attractive since the resulting circuits have few interconnections and lend themselves well to large-scale integration (for instance, binary trees form an efficient interconnection pattern [HORO81]). Moreover, multiplexer networks can be used as universal logic modules (ULMs) [TABL76, VOLT77], thereby reducing the number of basic components needed for logic design.

Decision tree representations of Boolean functions exhibit several advantages over Boolean formulas. Lee [LEE59] showed that at most $O(2^n/n)$ diagram nodes are required to represent any Boolean function of n variables, which compares very favorably with the $O(2^n/\log n)$ operators that may be needed by an unfactored Boolean formula [SAVA76]. Moreover, every operator of the Boolean formula must be carried out in order to evaluate the formula, so that up to $O(2^n/\log n)$ operations may be performed, while a decision diagram will never require more than n variable evaluations. Thus decision diagrams express Boolean functions at least as compactly as Boolean formulas and are greatly more efficient as an evaluation tool. (The latter property is used, e.g., for the repeated evaluation of Boolean queries in a large database [WONG76].) Finally, decision diagrams lend themselves to composition and recursion, as are shown in the next section, while the same operations are very difficult to carry out using formulas. Some of those advantages were rediscovered and some pointed out for the first time by AKER78, who investigated the use of binary decision diagrams (with a slightly different definition) in testing digital systems.

In conjunction with the evaluation of Boolean functions, it must be noted that almost all Boolean functions have a pessimal worst-case testing cost (i.e., all variables are tested on at least one path). This result, due to RIVE76a, was later complemented by MORE81b, who proved that all symmetric and all linearly separable (also called threshold) Boolean functions possess this property. An important consequence of these results is that synchronous multiplexer implementations of Boolean functions in most cases cannot be optimized with respect to their propagation delay, since this delay is determined by the longest path through the network.

Finally decision diagrams can be used to model the control structure of a program (as advocated in PRAT78, where they are called "atomic digraphs"), in particular, in relation in Ianov's schemata [IANO60]. It then becomes important to recognize identical structures, that is, to decide whether or not two decision diagrams are equivalent. This problem is known to be NP-hard [FORT78]; however, BLUM80 described an efficient algorithm that solves the equivalence problem probabilistically (i.e., which provides an answer correct within a given—and refinable—percentage of error).

4.3 Analysis of Algorithms

The worst-case number of tests (the height) of a decision tree indicates a minimum number of argument evaluations that must be performed in order to compute a function. As such, finding the minimum height of any tree representation of a function is a useful technique for deriving bounds to be used in the worst-case analysis of algorithms. The previously mentioned results of RIVE76a and MORE81b, showing that large classes of Boolean functions require any decision tree representation to have maximal height, are an example of such analysis; indeed, RIVE76b built upon these results to prove some lower bounds on the complexity of graph algorithms based upon a matrix representation.

The worst-case number of evaluations must be at least equal to the ratio of the initial ambiguity of the function to the upper bound on the information supplied by each evaluation. Since the evaluation of a k-ary variable provides at most $\log_2 k$ bits of information, the height of any tree for a function, f, or k-ary variables, must obey the relation

$$h_{\min} \geq H_f/\log_2 k. \qquad (10)$$

This relation has been used to provide lower bounds on the complexity of several combinational problems, such as sorting [KNUT71] and various set operations [REIE72]. The decision tree approach has recently been generalized to handle probabilistic, nondeterministic, and alternating models of complexity [MANB82].

5. RECENT DEVELOPMENTS

5.1 Composition and Recursion

The decision diagram model of representation as described so far is limited to (partial) functions. Several extensions have recently been proposed to include composition of diagrams [AKER78, MORE80b] and modeling of relations and simple recursion [MORE80b].

Whereas a function assigns at most one value to each combination of variables, a relation may assign any number of values, that is, any subset of the set of values. Thus, in particular, a relation accurately models an ambiguous decision table, where the same rule (the same combination of variables) may specify more than one set of actions. In accordance with KING73, we assume that, when inconsistent rules apply, any of the assigned action sets may actually be chosen for execution. In terms of relations, a decision tree implementation can choose to specify for each combination of variables any (or some, or all) of the values from the subset assigned to that combination by the relation. Since combinations of variables for which the function (relation) is not defined are usually allowed to take any convenient value, we may assume that such combinations are in fact related to the whole set of values. Another consequence of our assumptions is that a relation is constant exactly when the intersection of the subsets of values assigned to all the combinations of its variables is not empty (for one can choose to use any of the values present in the intersection for each of the combinations of variables, thereby effectively transforming the relation into a constant function). Under such assumptions, all of the results discussed so far apply to the representation and evaluation of relations [MORE80b].

A particularly important tool in the analysis of problems is decomposition, which tries to simplify a problem by partitioning it into smaller parts (the rationale being that the complexity of the whole is more than the sum of the complexities of its parts); "divide and conquer" is a time-honored aspect of decomposition. Conversely, composition is an important tool in synthesis. We have seen that decision trees induce a natural decomposition—Shannon's decomposition; thus it remains to demonstrate how to compose decision trees and diagrams. Two such compositions can be distinguished: *leaf* composition and *node* composition.

Leaf composition stems from the simple hierarchical idea of a "tree of trees." As an example, consider a pattern recognition problem such as bird identification. To most of us, identifying a bird as a "sparrow" or a "dove" is sufficient; to a bird watcher or an ornithologist, however, this is a vague classification that must be greatly refined to include species and subspecies. This suggests a two-step identification, in which a rough classification is first made, followed by a highly specialized procedure for further refinements. This offers several advantages: (1) it is practical since it can be of use to both uninitiated and specialists; (2) it is efficient since the evaluation can be profitably (for the uninitiated) stopped after the first stage; and (3) it can be greatly optimized in the second stage since it is then possible to design highly specialized tests. The first step in such a design consists of a single decision tree; most of the leaves of the tree, however, do not give a value, but rather designate another decision tree to be used in the second step. Thus one can stop the evaluation upon reaching a leaf of the first tree, taking the "name" of the second tree as the result of the evaluation,

or continue evaluation by proceeding to the second tree. The latter choice results in the composition of the two trees, and it is called leaf composition since it replaces a leaf by another tree.

Formally, then, leaf composition of two trees is the process of attaching the second tree in place of appropriate leaves in the first tree. (The analog in the software world is a transfer of control between modules without transfer of information, such as chaining.) Clearly, the second tree cannot share variables with the first lest the composed tree test the same variable twice on some path. A special case of interest is the composition of decision trees for Boolean functions, where the second tree is attached in place of every leaf with the same label in the first tree. One easily verifies that such a composition results in a logical OR of the two functions when the leaves labeled "0" are replaced, and in a logical AND when the leaves labeled "1" are used instead. Moreover, the composition is then commutative (in terms of the function it yields), just as the logical operation that it implements. Figure 11a shows two trees, which are OR composed in Figure 11b and AND composed in Figure 11c.

The behavior of the various optimization criteria under leaf composition is simple. The storage cost of the composition is the sum of the cost of the first tree and, for each replaced leaf, of the cost of the second tree. This can be simplified for diagrams; since only one leaf of each label can exist, the storage cost of the composed diagram is just the sum of the costs of the component diagrams. The expected testing cost of the composition is the sum of the cost of the first tree and of the cost of the second tree, the latter being multiplied by the probability that one of the replaced leaves will be reached. Thus all the techniques used for the optimization of decision trees are applicable to the optimization of composed trees (see MORE80b). In connection with the composition of Boolean decision trees mentioned above, PERL76 proved that, on the average, the order of composition is irrelevant to the expected testing cost of the composed tree. (Notice, however, that this is clearly not the case for the OR compositions illustrated in Figure 11b.)

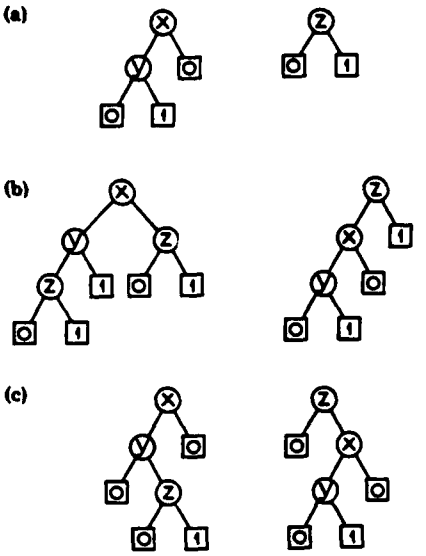


Figure 11. An example of leaf composition: (a) the two functions; (b) left and right OR compositions; (c) left and right AND compositions.

Whereas the rationale behind the leaf composition was the progressive refinement of function values, node composition introduces a refinement of the values of the variables. Thus the former is associated with an explicit tree hierarchy, while the latter induces a functional hierarchy. In a node composition, a k-ary variable is replaced by a tree with at least one leaf labeled with each value from 0 to $k - 1$; this tree specifies how to evaluate the variable it replaces. (Thus the software analog is the use of an auxiliary procedure that returns a value, i.e., a subroutine call.) The effect of node composition on functions expressed by formulas is just the substitution in the first function's formula of the second function's formula for each appearance of the node variable. In terms of trees, the composed tree is obtained by using the following procedure for each occurrence of the specified node: replace the node by the second tree and attach the jth subtree of the node to each leaf labeled j in the second tree. As for leaf composition, the trees used in node composition must not share any variable. Figure 12 illustrates a node composition. Akers [AKER78] proposed this

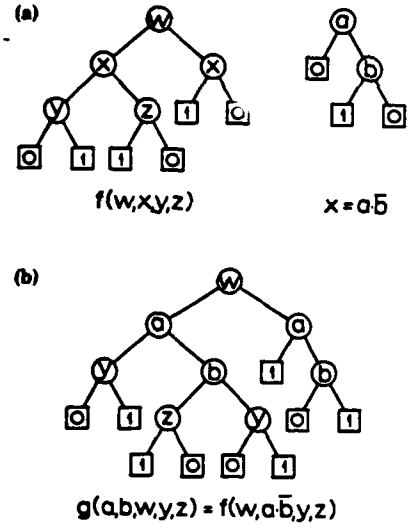


Figure 12. An example of node composition: (a) the trees for the function (left) and the variable (right); (b) the composed tree.

composition and studied its use in the design and analysis of logic functions.

It is noted that the behavior of the optimization criteria under node composition is somewhat complex. While the diagram storage cost of a composition is simply the sum of the cost of the first diagram and, for each replaced node, of the cost of the second diagram, the other costs depend on exact structure of the second tree. In particular, it is necessary to know how many leaves of the second tree share the same label, say label j, since the jth subtree of the replaced node will be attached to each of these leaves. However, the optimization methods described in this article can be applied with suitable modifications.

Both modes of composition can be used at once. Recent research on the problem of bacteriological identification in a clinical environment [SHAP81] suggests as the initial model a user-specified identification tree that simply describes a hierarchy of classes and subclasses of bacteria (as determined by local factors such as common mode of treatment in initial stages, likelihood of occurrence in the geographical area, etc.). The hierarchy involves leaf composition, while the actual tests to be used for identification are specified by node composition.

In discussing both modes of composition, we have purposefully avoided a delicate problem: What if a tree is composed with itself? In such a case, the leaves or nodes of the first tree are replaced by the identical tree, so that more replacements are possible, and so on. This creates an infinite recursion, giving rise to an infinite tree (or a diagram with cycles). Although composing a tree with itself may appear contrived, recursion is a sufficiently fundamental phenomenon that a study of its effects on decision trees is warranted. Unfortunately, very little work has been done in this area.

Moret [MORE80b] studied the recursion due to a single leaf composition and showed that the concepts of expected testing cost, diagram storage cost, and activity can be extended to such simple recursive trees. Although the tree storage cost is clearly infinite, as is the worst case testing cost, the expected testing cost is generally finite. This stems from the fact that there usually is a nonzero probability, call it e, of reaching a nonreplaceable leaf in the component tree, so that the probability of recursing one level, $1 - e$, is less than one. Now, the probability of recursing k levels is just $(1 - e)^k$, so that the average number of recursion levels used before termination is

$$r_{av} = \sum_{k=0}^{\infty} (1 - e)^k = \frac{1}{e}.$$

This factor can be used for transforming the expected testing cost of a single, nonrecursive copy of the tree into the expected testing cost of the recursive tree; the activities of the variables can be computed similarly.

5.2 Applications to Testing

Decision trees have long been used for purposes of fault diagnosis, as previously seen. Such uses, however, apply decision trees to the analysis of a discrete function which is not that which they represent. Akers [AKER78, AKER79] proposed that binary decision diagrams be used as the basis for developing tests for the Boolean function which they represent. Some of the reasons given have been discussed above, such as

compactness, good structure, and existence of composition. Other reasons include the close relationship between binary decision diagrams and standard logic design and the ease of automation in handling diagrams.

Moret [MORE81a] advocated the use of decision trees as alternate models for system analysis. A standard tool in the analysis of system reliability is the fault tree [BARL75], which is just a graphical representation (using AND and OR gates) of the Boolean function describing a system's state in terms of the state of its components (where the only possible values are "working" and "failing"). Fault trees are used for the assessment of the overall reliability of a system and of its sensitivity to the state of various components. The same analysis can be performed using decision trees and activities, with added advantages; (1) as seen, decision trees are more efficient than formulas (which is essentially what fault trees are); (2) decision trees can be used for multivalued functions, whereas fault trees are restricted to Boolean functions; and (3) decision trees can include recursion and general composition operations, while fault trees are limited to an equivalent of node composition. A possible drawback, however, is that the manipulation of Boolean formulas, while inherently inefficient, is well understood and has been successfully implemented (e.g., see WORR75), whereas that of decision trees is still in its infancy.

6. CONCLUSION

This article has provided a unified framework of definitions and notation for decision trees and diagrams; it has examined the problem of optimization, reviewed the main applications and contributions, and described some recent developments. While often difficult to optimize, decision trees and diagrams emerge as efficient representations of discrete functions, of particular interest in pattern recognition, logic design, programming methodology, and system analysis. Although several research programs are actively concerned with decision trees, further areas of study have been identified, notably the quality of optimization heuristics and the use of general recursion.

ACKNOWLEDGMENTS

The research leading to this work was supported by the Office of Naval Research, Arlington, Virginia, under Contract No. 0014-78-C-0311.

The author wishes to thank Professors R. C. Gonzalez and M. G. Thomason at the University of Tennessee for their advice and encouragement, as well as for introducing the author to decision trees and collaborating on his research; Professor H. D. Shapiro at the University of New Mexico, for many fruitful discussions and a very careful review of the manuscript; and the editor and referees, for detailed and constructive comments.

REFERENCES

AKER78 AKERS, S.B. "Binary decision diagrams." *IEEE Trans. Comput.* TC-27, 6 (1978), 509–516.

AKER79 AKERS, S. B. "Probabilistic techniques for test generation from functional descriptions." In *Proc. 9th Symp. Fault-Tolerant Computing*. IEEE, New York, 1979, pp. 113–116.

BARL75 BARLOW, R. E., FUSSELL, J. B., AND SINGPURWALLA, N. D., EDS. *Reliability and Fault Tree Analysis*. SIAM Press, Philadelphia, Pa., 1975.

BAYE73 BAYES, A. J. "A dynamic programming algorithm to optimize decision table code." *Austral. Comput. J.* 5, 2 (1973), 77–79.

BELL78 BELL, D. A. "Decision trees, tables, and lattices." In *Pattern Recognition: Ideas in Practice*, B. G. Batchelor, Ed. Plenum Press, New York, 1978. Chap. 5.

BLUM80 BLUM, M., CHANDRA, A. K., AND WEGMAN, M. N. "Equivalence of free Boolean graphs can be decided probabilistically in polynomial time." *Inf. Process. Lett.* 10, 2 (1980), 80–82.

BREI75a BREITBART, Y., AND REITER, A. "Algorithms for fast evaluation of Boolean expressions." *Acta Inf.* 4 (1975), 107–116.

BREI75b BREITBART, Y., AND REITER, A. "A branch-and-bound algorithm to obtain an optimal evaluation tree for monotonic Boolean functions." *Acta Inf.* 4 (1975), 311–319.

BREI78 BREITBART, Y., AND GAL, S. "Analysis of algorithms for the evaluation of monotonic Boolean functions." *IEEE Trans. Comput.* TC-27, 11 (1978), 1083–1087.

BROW77 BROWN, P. J. "Functions for selecting tests in diagnostic key construction." *Biometrika* 64 (1977), 589–596.

CERN79a CERNY, E., MANGE, D., AND SANCHEZ, E. "Synthesis of minimal binary decision trees." *IEEE Trans. Comput.* TC-28, 7 (1979), 472–482.

CERN79b CERNY, E., AND MOREAU, T. "Test evaluation with a decision machine." In *Proc. 9th Symp. Fault-Tolerant Com-*

puting. IEEE, New York, 1979, pp. 117–120.

CHAN65 CHANG, H. Y. "An algorithm for selecting an optimal set of diagnostic tests." *IEEE Trans. Comput.* EC-14, 5 (1965), 706–711.

CHAN70 CHANG, H. Y., MANNING, E., AND METZE, G. *Fault Diagnosis of Digital Systems*. Wiley, New York, 1970.

DALL74 DALLWITZ, M. J. "A flexible computer program for generating identification keys." *Syst. Zool.* 23 (1974), 50–57.

DATT81 DATTATREYA, G. R., AND SARMA, V. V. S. "Bayesian and decision tree approaches for pattern recognition including feature measurement costs." *IEEE Trans. Patt. Anal. Mach. Intell.* PAMI-3, 3 (1981), 293–298.

DAVI80 DAVIO, M., DESCHAMPS, J.-P., AND THAYSE, A. *Discrete and Switching Functions*. McGraw-Hill, New York, 1978.

EGLE63 EGLER, J. F. "A procedure for converting logic table conditions into an efficient sequence of test instructions." *Commun. ACM* 6, 9 (Sept. 1963), 510–514.

FORT78 FORTUNE, S., HOPCROFT, J., AND SCHMIDT, E. M. *The Complexity of Equivalence and Containment for Free Single Variable Schemes*. Lecture Notes in Computer Science, vol. 62, Springer-Verlag, New York, 1978, pp. 227–240.

GANA73 GANAPATHY, S., AND RAJARAMAN, V. "Information theory applied to the conversion of decision tables to computer programs." *Commun. ACM* 16, 9 (Sept. 1973), 532–539.

GARE70 GAREY, M. R. "Optimal binary decision trees for diagnostic identification problems." Ph.D. dissertation, Univ. of Wisconsin, Madison, 1970.

GARE72a GAREY, M. R. "Optimal binary identification procedures." *SIAM J. Appl. Math.* 23, 2 (1972), 173–186.

GARE72b GAREY, M. R. "Simple binary identification problems." *IEEE Trans. Comput.* TC-21, 6 (1972), 588–590.

GARE74 GAREY, M. R., AND GRAHAM, R. L. "Performance bounds on the splitting algorithm for binary testing." *Acta Inf.* 3 (1974), 347–355.

GARE79 GAREY, M. R., AND JOHNSON, D. S. *Computers and intractability: A guide to the theory of NP-completeness*. Freeman, San Francisco, Calif., 1979.

GOWE71 GOWER, J. C., AND BARNETT, J. A. "Selecting tests in diagnostic keys with unknown responses." *Nature* 232 (1971), 491–493.

GOWE75 GOWER, J. C., AND PAYNE, R. W. "A comparison of different criteria for selecting binary tests in diagnostic keys." *Biometrika* 62 (1975), 665–672.

GYLL63 GYLLENBERG, H. G. "A general method for deriving determination schemes for random collections of microbial isolates." *Ann. Acad. Sci. Fenn. A IV*, 69 (1963), 1–23.

HALP74 HALPERN, J. "Evaluating Boolean functions with random variables." *Int. J. Syst. Sci.* 5, 6 (1974), 545–553.

HANA77 HANANI, M. Z. "An optimal evaluation of Boolean expressions in an online query system." *Commun. ACM* 20, 5 (May 1977), 344–347.

HARR65 HARRISON, M. A. *Introduction to Switching and Automata Theory*. McGraw-Hill, New York, 1965.

HART82 HARTMANN, C. R. P., VARSHNEY, P. K., MEHROTRA, K. G., AND GERBERICH, C. L. "Application of information theory to the construction of efficient decision trees." *IEEE Trans. Inf. Theory* IT-28, 4 (1982), 565–577.

HAUS75 HAUSKA, H., AND SWAIN, P. "The decision tree classifier: Design and potential." In *Proc. 2nd Symp. Machine Processing of Remotely Sensed Data* (Lafayette, Ind.). IEEE, New York, 1975, pp. 1A38–1A48.

HORO81 HOROWITZ, E., AND ZORAT, A. "The binary tree as an interconnection network: Applications to multiprocessor systems and to VLSI." *IEEE Trans. Comput.* TC-30, 4 (1981), 247–253.

HULM75 HULME, B. L., AND WORRELL, R. B. "A prime implicant algorithm with factoring." *IEEE Trans. Comput.* TC-24, (1975), 1129–1131.

HUNG74 HUNG, T. Q. "La construction des clefs de diagnostic." M.S. thesis, University of Montreal, 1974.

HYAF76 HYAFIL, L., AND RIVEST, R. L. "Constructing optimal binary decision trees is NP-complete." *Inf. Process. Lett.* 5, 1 (1976), 15–17.

IANO60 IANOV, I. "The logical schemes of algorithms." In *Problems and Cybernetics*, vol. 1. Pergamon Press, New York, 1960, pp. 82–140.

JARD71 JARDINE, N., AND SIBSON, R. *Mathematical Taxonomy*. Wiley, New York, 1971.

JOHN56 JOHNSON, S. M. "Optimal sequential testing." Project RAND Res. Mem. RM-1652, 1956.

KANA79 KANAL, L. N. "Problem-solving models and search strategies for pattern recognition." *IEEE Trans. Patt. Anal. Mach. Intell.* PAMI-1, 2 (1979), 193–201.

KAND80 KANDEL, A., AND FRANCIONI, J. H. "On the properties and applications of fuzzy-valued switching functions." *IEEE Trans. Comput.* TC-29, 11 (1980), 986–993.

KING73 KING, P. J. H., AND JOHNSON, R. G. "Some comments on the use of ambiguous decision tables and their conversion to computer programs." *Commun. ACM* 16, 5 (May 1973), 287–290.

KLET60 KLETSKY, E. J. "An appliction of the information theory approach to failure

diagnosis." *IRE Trans. Reliability Quality Control* RQC-9 (1960), 29–39.

KNUT71 KNUTH, D. E. "Mathematical analysis of algorithms." In *Proc. IFIP Congress 71*, vol. 1. Elsevier-North Holland, New York, 1971, pp. 135–143.

KNUT73 KNUTH, D. E. *The Art of Computer Programming, Vol. 3: Searching and Sorting.* Addison-Wesley, Reading, Mass., 1973.

KULK76 KULKARNI, A. V., AND KANAL, L. N. "An optimization approach to hierarchical classifier design." In *Proc. 3rd Int. Jt. Conf. Pattern Recognition* (San Diego, Calif.). IEEE, New York, 1976.

KULK78 KULKARNI, A. V. "On the mean accuracy of hierarchical classifiers." *IEEE Trans. Comput.* TC-27, 8 (1978), 771–776.

LAWL66 LAWLER, E., AND WOOD, D. "Branch-and-bound methods: A survey." *Oper. Res.* 14, 4 (1966), 699–719.

LEE59 LEE, C. Y. "Representation of switching circuits by binary-decision programs." *Bell Syst. Tech. J.* 38 (1959), 985–999.

LOVE79 LOVELAND, D. W. "Selecting optimal test procedures from incomplete test sets." Tech. Rep. CS, 1979–7, Duke Univ., Durham, N. C., 1979.

MANB82 MANBER, U., AND TOMPA, M. "Probabilistic, nondeterministic, and alternating decision trees." In *Proc. 14th Ann. Symp. Theory of Computing* (San Francisco, May 5–7). ACM, New York, 1982, pp. 234–244.

MAND64 MANDELBAUM, D. "A measure of efficiency of diagnostic tests upon sequential logic." *IEEE Trans. Comput.* EC-13 (1964), 630.

MART78 MARTELLI, A., AND MONTANARI, U. "Optimizing decision trees through heuristically guided search." *Commun. ACM* 21, 12 (Dec. 1978), 1025–1039.

MASE82 MASEK, W. J. "Some NP-complete set covering problems" (to appear in *Theor. Comput. Sci.*).

MEIS73 MEISEL, S. W., AND MICHALOPOULOS, D. A. "A partitioning algorithm with application in pattern classification and the optimization of decision trees." *IEEE Trans. Comput.* TC-22, 1 (1973), 93–103.

METZ77 METZNER, J. R., AND BARNES, B. H. *Decision Table Languages and Systems.* Academic Press, New York, 1977.

MICH78 MICHALSKI, R. S. "Designing extended-entry decision tables and optimal decision trees using decision diagrams." Tech. Rep. UIUCDCS-R-78-898, Univ. of Illinois at Urbana-Champaign, 1978; also IEEE Comput. Rep. R78-126, 1978.

MISR72 MISRA, J. "A study of strategies for multistage testing." Ph.D. dissertation, The Johns Hopkins Univ., Baltimore, Md., 1972.

MOLL62 MOLLER, F. "Quantitative methods in the systematics of the Actinomycetales IV. The theory and application of a probabilistic identification key." *G. Microbiol.* 10 (1962), 29–47.

MONT62 MONTALBANO, M. "Tables, flow charts, and program logic." *IBM Syst. J.* 1 (1962), 51–63.

MORE80a MORET, B. M. E. "The representation of discrete functions by decision trees: aspects of complexity and problems of testing." Ph.D. dissertation, Univ. of Tennessee, Knoxville, 1980.

MORE80b MORET, B. M. E., THOMASON, M. G., AND GONZALEZ, R. C. "The activity of a variable and its relation to decision trees." *ACM Trans. Program. Lang. Syst.* 2, 4 (Oct. 1980), 580–595.

MORE81a MORET, B. M. E., THOMASON, M. G., AND GONZALEZ, R. C. "The use of activity in testing digital and analog systems." In *Proc. 1st Automatic Test Program Generation Workshop* (Philadelphia, Pa.,). IEEE, New York, pp. 120–127.

MORE81b MORET, B. M. E., THOMASON, M. G., AND GONZALEZ, R. C. "Optimization criteria for decision trees." Tech. Rep. CS81-6, Dep. of Computer Science, Univ. of New Mexico, Albuquerque, N. Mex., 1981.

MORE82 MORET, B. M. E., AND SHAPIRO, H. D. "Experience with the minimum test set problem." Tech. Rep. CS82-4, Dep. of Computer Science, Univ. of New Mexico, Albuquerque, N. Mex., 1982.

MORS71 MORSE, L. E. "Specimen identification and key construction with time-sharing computers." *Taxon* 20 (1971), 269–282.

OSBO63 OSBORNE, D. V. "Some aspects of the theory of dichotomous keys." *New Phytol.* 62 (1963), 111–160.

PANK70 PANKHURST, R. J. "A computer program for generating diagnostic keys." *Comput. J.* 13, 2 (1970), 145–151.

PAYH77 PAYNE, H. J., AND MEISEL, W. S. "An algorithm for constructing optimal binary decision trees." *IEEE Trans. Comput.* TC-26, 9 (1977), 905–916.

PAYR77 PAYNE, R. W. "Reticulation and other methods of reducing the size of printed diagnostic keys." *J. Gen. Microbiol.* 98 (1977), 595–597.

PAYR80 PAYNE, R. W., AND PREECE, D. A. "Identification keys and diagnostic tables: a review." *J. R. Statist. Soc. A* 143, 3 (1980), 253–292 (with discussion).

PAYR81 PAYNE, R. W. "Selection criteria for the construction of efficient diagnostic keys." *J. Statist. Plann. Inf.* 5 (1981), 27–36.

PERL76 PERL, Y., AND BREITBART, Y. "Optimal sequential arrangement of evaluation trees for Boolean functions." *Inf. Sci.* 11 (1976), 1–12.

PICA72 PICARD, C. F. *Graphes et questionnaires. Vol. 2: Questionnaires.* Gauthier-Villars, Paris, 1972.

POLL65 POLLACK, S. L. "Conversion of limited-entry decision tables to computer programs." *Commun. ACM* 8, 11 (Nov. 1965), 677–682.

POOC74 POOCH, U. W. "Translation of decision tables." *ACM Comput. Surv.* 6, 2 (June 1974), 125–151.

PRAT78 PRATHER, R. E., AND CASSTEVENS, H. T. "Realization of Boolean expressions by atomic digraphs." *IEEE Trans. Comput.* TC-27, 8 (1978), 681–688.

PRES65 PRESS, L. I. "Conversion of decision tables to computer programs." *Commun. ACM* 8, 6 (June 1965), 385–390.

RABI71 RABIN, J. "Conversion of limited-entry decision tables into optimal decision trees: Fundamental concepts." *SIGPLAN Not. (ACM)* 6, 8 (Sept. 1971), 68–74.

REIE72 REINGOLD, E. M. "On the optimality of some set agorithms." *J. ACM* 19, 4 (Oct. 1972), 649–659.

REIL66 REINWALD, L. T., AND SOLAND, R. M. "Conversion of limited-entry decision tables to optimal computer programs I: Minimum average processing time." *J. ACM* 13, 3 (July 1966), 339–358.

REIL67 REINWALD, L. T., AND SOLAND, R. M. "Conversion of limited-entry decision tables to optimal computer programs II: Minimum storage requirement." *J. ACM* 14, 4 (Oct. 1967), 742–756.

RESC61 RESCIGNO, A., AND MACCACARO, G. A. "The information content of biological classifications." In *Information Theory: Fourth London Symposium*, C. Cherry, Ed. Butterworth, London, 1961, pp. 437–445.

RIES63 RIESEL, H. "In which order are different conditions to be examined." *BIT* 3 (1963), 255–256.

RIVE76a RIVEST, R. L., AND VUILLEMIN, J. "On the number of argument evaluations required to compute Boolean functions." Mem. ERL-M476, Electronics Research Laboratory, Univ. of California at Berkeley, 1976.

RIVE76b RIVEST, R. L., AND VUILLEMIN, J. "On recognizing graph properties from adjacency matrices." *Theor. Comp. Sci.* 3 (1976), 371–384.

ROUN79 ROUNDS, E. M. "A combined non-parametric approach to feature selection and binary tree design." In *Proc. Conf. Pattern. Recognition and Image Processing*, (Chicago, Ill.). IEEE, New York, 1979, pp. 38–43.

SAVA76 SAVAGE, J. E. *The Complexity of Computing.* Wiley, New York, 1976.

SCHU76 SCHUMACHER, H., AND SEVCIK, K. C. "The synthetic approach to decision table conversion." *Commun ACM.* 19, 6 (June 1976), 343–351.

SETH80 SETHI, I. K., AND CHATTERJEE, B. "Conversion of decision tables to efficient sequential testing procedures." *Commun. ACM* 23, 5 (May 1980), 279–285.

SHAN48 SHANNON, C. E. "A mathematical theory of communication." *Bell Syst. Tech. J.* 27 (1948), 379–423.

SHAP81 SHAPIRO, H. D. Private communication. Dep. Computer Science, Univ. of New Mexico, Albuquerque, N. Mex., 1981.

SHWA74 SHWAYDER, K. "Extending the information theory approach to converting limited-entry decision tables to computer programs." *Commun. ACM* 17, 9 (Sept. 1974), 532–537.

SHWA75 SHWAYDER, K. "Combining decision rules in a decision table." *Commun. ACM* 18, 8 (Aug. 1975), 476–480.

SILB71 SILBERG, B. ED. "Decision tables." *SIGPLAN Not. (ACM)* 6, 8 (Sept. 1971).

SLAG64 SLAGLE, J. R. "An efficient algorithm for finding certain minimum-cost procedures for making binary decisions." *J. ACM* 11, 3 (July 1964), 253–264.

SLAG70 SLAGLE, J. R., CHANG, C. L., AND LEE, R. C. T. "A new algorithm for generating prime implicants." *IEEE Trans. Comput.* TC-19, 4 (1970), 304–310.

SLAG71 SLAGLE, J. R., AND LEE, R. C. T. "Application of game tree searching techniques to sequential pattern recognition." *Commun. ACM* 14, 2 (Feb. 1971), 103–110.

TABL76 TABLOSKI, T. F., AND MAULE, F. J. "Numerical expansion technique for minimal multiplexer logic circuits." *IEEE Trans. Comput.* TC-25, 7 (1976), 684–702.

THAY78 THAYSE, A., DAVIO, M., AND DESCHAMPS, J. P. "Optimization of multivalued decision algorithms." In *Proc. Int. Symp. on Multivalued Logic* (Rosemont, Pa.), 1978, pp. 171–178.

THAY81a THAYSE, A. "P-functions: A new tool for the analysis and synthesis of binary programs." *IEEE Trans. Comput.* TC-30, 2 (1981), 126–134.

THAY81b THAYSE, A. *Boolean Calculus of Differences,* Lecture Notes in Computer Science, vol. 101. Springer-Verlag, New York, 1981.

VOIT77 VOITH, R. "ULM implicants for minimization of universal logic modules circuits." *IEEE Trans. Comput.* TC-26, 5 (1977), 417–424.

VOSS52 VOSS, E. G. "The history of keys and phylogenetic trees in systematic biology." *J. Sci. Denison Univ.* 43 (1952), 1–25.

WEID77 WEIDE, B. "A survey of analysis techniques for discrete algorithms." *ACM Comput. Surv.* 9, 4 (Dec. 1977), 291–313.

WILL80 WILLCOX, W. R., LAPAGE, S. P., AND HOLMES, B. "A review of numerical

methods in bacterial identification." *Antonie van Leeuwenhoek* 46, 3 (1980), 233–299.

WONG76 WONG, E., AND YOUSSEFI, K. "Decomposition—A strategy for query processing." *ACM Trans. Database Syst.* 1, 3 (Sept. 1976), 223–241.

WORR75 WORRELL, R. B. "Using the Set Equation Transformation System in fault tree analysis." In *Reliability and Fault Tree Analysis*, R. E. Barlow et al., Eds. SIAM Press, Philadelphia, Pa., 1975, pp. 165–186.

WU75 WU, C., LANDGREBE, D., AND SWAIN, P. "The decision tree approach to classification." Tech. Rep. TR-EE 75-17, Purdue Univ., Lafayette, Ind., 1975.

YASU71 YASUI. T. "Some aspects of decision table conversion techniques." *SIGPLAN Not. (ACM)* 6, 8 (Sept. 1971), 104–111.

YOU76 YOU, K. C., AND FU, K. S. "An approach to the design of a linear binary tree classifier." In *Proc. Symp. Machine Processing of Remotely Sensed Data* (Lafayette, Ind.). IEEE, New York, 1976, pp. 3A1–3A10.

ZIMM59 ZIMMERMAN, S. "An optimal search procedure." *Am. Math. Mon.* 66 (1959), 690–693.

Received August 1981; final revision accepted September 1982.

The use of activity in testing digital and analog systems.*

Bernard M.E. Moret†, Michael G. Thomason‡, and Rafael C. Gonzalez‡

Abstract

With the advent of large scale integration, testing methods must be developed which rely solely on the input-output behavior of systems, thereby requiring an implementation-independent model of system behavior. Such a model, the decision tree, which has proved of great use in many areas of Computer Science, is briefly presented. Using this model, a measure of the complexity of multivalued discrete functions is developed, as well as a measure of the contribution of individual variables to the overall complexity. The latter concept, the activity of a variable, is shown to have considerable potential for the design of incomplete testing procedures. In particular, exercising those variables which have the largest activity maximizes the probability of error detection in systems with equally likely faults. Finally, activity is shown to be a powerful tool for the analysis of multivalued fault-trees, thereby allowing the application of some digital testing techniques to analog systems modelled by multivalued functions.

* This research is sponsored by the Office of Naval Research, Arlington, Va, under contract No. N00014-78-C-0311.

† Department of Computer Science, University of New Mexico, Albuquerque, New Mexico 87131.

‡ Department of Computer Science and Department of Electrical Engineering, University of Tennesse, Knoxville, TN 37916.

In: Proceedings of the 1st IEEE Workshop on Automatic Test Program Generation, Philadelphia, 1981, pp.120-127.

Introduction

As the size and complexity of new integrated systems increase, the need arises for methods of analysis, testing, and design which are implementation-independent, using only input-output specifications. Of particular importance is the ability to evaluate the complexity of a problem, as well as how individual variables contribute to it, in order to select an appropriate set of analytical tools and establish guidelines for testing and design procedures.

One implementation-independent model of discrete function evaluation, the decision tree, has long been used in Computer Science for establishing lower bounds on the complexity of problems (e.g., Knuth 71), designing switching circuits (e.g., Cerny 79), or establishing classification procedures for pattern recognition (Bell 78) and machine diagnosis (Chang 70). A decision tree is essentially a sequential evaluation procedure, whereby the value of a variable (test) is determined and the next action (to select another variable to evaluate or to output the value of the system's function) is chosen accordingly. In particular, decision trees can be used to determine the state of a system (Halpern 74); Figure 1a shows a possible decision tree for the state of the simple system pictured in Figure 1b.

In a previous investigation (Moret, Thomason, and Gonzalez 80, 81), the authors generalized the decision tree model to include composition and simple recursion, thus allowing the modelling of hierarchical systems with simple feedback. They also developed a new complexity measure for discrete functions, the intrinsic cost, as well as a measure of the contribution of a variable to the complexity of the function, and showed the close relationship existing between these measures and the decision tree model. As detailed below, the concept of activity shows considerable potential as a tool for system testing.

Activity and incomplete testing

In an input-output system, a failure is characterized by a deviation from the expected output signal (that is, a different value for discrete systems and a value

out of tolerance for analog systems]. This approach is known as signal reliability (Koren 79), in contrast with the conventional functional reliability, which considers all internal (and possibly non-critical) system faults. Signal reliability is thus more accurate and better suited to large integrated systems.

The thorough testing of a system can only be done by exhaustion; such an approach, however, is unfeasible for all but the simplest systems. Thus one is forced to use some methods of incomplete testing. In the case of combinational (i.e., memoryless) discrete circuits, (Losq 78) has shown that random compact testing, a method which applies a sequence of random input vectors to a system and compares some output statistics with those gathered from a perfect ("gold") unit, can yield very reliable estimates at only a small fraction of the cost of exhaustive testing. Often, however, such a method is inapplicable, because not all inputs are controllable; in a system with memory (feedback), for instance, the values of the feedback variable often cannot be either examined or modified (as illustrated in Figure 2).

When only a fraction of the variables is accessible or when only the most "important" variables must be tested, exhaustive testing can be used with a selected subset of variables. Such a subset must be chosen such that the probability of detecting a malfunction is maximized. The authors have shown that when all malfunctions are equally likely, such a subset must consist of these variables which have the largest activity (Moret 80). Thus the activity of a variable measures, in some sense, how important a variable is to the correct functioning of a system.

Activity and fault trees.

A complex system is rarely specified as a whole, but is conceived as a structure of simpler subsystems which interact by communicating the values of variables. Reliability analysis is then carried out on the structural relationships by representing each subsystem by a single variable qualifying its operating state; when such variables are binary, taking the values "works" or "fails", this leads to the fault tree model, which has found widespread use in industry (Fussell 79, Reactor Safety Study 75, Lapp 77).

A fault tree is essentially a Boolean function describing the set of conditions (on the subsystems) necessary to make a complete system fail. Figure 3 shows a possible fault tree for the system of Figure 1b. Obviously, each subsystem can in turn be decomposed and modelled in the same way. Fault trees are used to determine the probability of failure of a system as well as for the study of the role of individual subsystems. A tool commonly used for the latter purpose is the Boolean differential calculus (Bennetts 75, Thomason and Page 76). The Boolean difference of a Boolean function, f, with respect to one of its variables, x, is the function:

$$df/dx = f|_{x=0} \oplus f|_{x=0}$$

where $\oplus$ denotes summation modulo 2. It is well known that df/dx=1 exactly when critically depends on x, so that the probability that a system represented by f fails due to the failure of subsystem x is

$$prob(df/dx=1) \cdot prob(x \text{ fails}).$$

The Boolean difference is closely related to the activity of a variable (Moret, Thomason, and Gonzalez 80): when all probabilities are equal, the activity of variable x reduces to $prob(df/dx=1)$.

Thus the activity of a variable is a natural extension to Boolean difference analysis; unlike the latter, it is applicable to arbitrary multivalued functions (as opposed to multi-valued logic (Bell, Page, and Thomason 72)), which makes it the tool of choice for the analysis of multivalued fault trees.

Apolications to Analog Systems

In analog systems, it is often difficult to decompose a system so that its components can be characterized as either perfect or faulty. Rather, the observed output signal deviates in some measure from the ideal output. Small deviations are potentially acceptable and large ones probably not, but there is an intermediate zone in which a signal can be just above tolerance without falling in either category. This situation is summarized in Figure 4.

It is conceivable that a cascade of two subsystems, each of which is above

tolerance but not faulty, results in a faulty system. In order to model this situation, a subsystem must be described not by a binary variable, but by a multivated variable, taking for instance the values "fault", "below tolerance", "above tolerance", and "perfect." Then the failure function is not a Boolean function but a general discrete function; fault tree models must be generalized and Boolean calculus is no longer applicable, so that activity becomes the main tool for analysis.

Conclusion

The activity of a variable, a new concept which measures the contribution of a variable to the (testing) complexity of a discrete function, has been introduced. It has been shown to be of great potential as a tool for the analysis and the testing of both discrete and analog systems.

References

(1) Bell, D.A. Decision trees, tables, and lattices. In Batchelor, B.G. Pattern Recognition: Ideas in Practice. Plenum Press, NY 1978.

(2) Bell, N.J., Page, E.W., and Thomason, M.G. Extension of the Boolean difference concept to multivalued logic systems. Proc. Symp. on Theory and Applications of Multiple-Valued Logic Design, Buffalo 1972.

(3) Bennetts, R.G. On the analysis of fault trees. IEEE Trans. Rel. R-24, 3(1975), 175-185.

(4) Cerny, E., Mange, D., and Sanchez, E. Synthesis of minimal binary decision trees. IEEE Trans. Comp. TC-28, 7(1979), 472-482.

(5) Chang, H.Y., Manning, E., and Metze, G. Fault Diagnosis of Digital Systems. Wiley & Sons, NY 1970.

(6) Fussell, J.B., Powers, G.J., and Bennetts, R.G. Fault trees - a state of the art discussion. IEEE Trans. Rel. R-23, 1(1979), 51-55.

(7) Halpern, J. A sequential testing procedure for a system's state identification. IEEE Trans. Rel. R-23, 4(1974), 267-272.

(8) Knuth, D.E. Mathematical analysis of algorithms. Proc. IFIP Congress 71, Vol. I(1971), 135-143.

(9) Koren, I., and Kohavi, Z. Sequential fault diagnosis in combinational networks. IEEE Trans. Comp. TC-26, 4(1977), 334-342.

(10) Koren, I. Analysis of the signal reliability measure and an evaluation procedure. IEEE Trans. Comp. TC-28, 3(1979), 244-249.

(11) Lapp, S.A., and Powers, G.J. Computer-aided synthesis of fault trees. IEEE Trans. Rel. R-26(1977), 2-13.

(12) Losg, J. Efficiency of random compact testing. IEEE Trans. Comp. TC-27, 6(1978), 516-525.

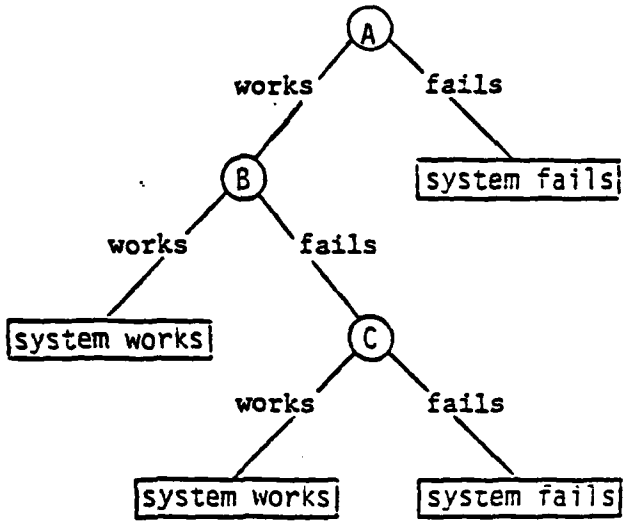
(13) Moret, B.M.E. The representation of discrete functions by decision trees: aspects of complexity and problems of testing. Ph.D. thesis, University of Tennessee, Knoxville 1980.

(14) Moret, B.M.E., Thomason, M.G., and Gonzalez, R.C. The activity of a variable and its relation to decision trees. ACM Trans. Progr. Lang. & Syst. TOPLAS 2, 4(1980), 580-595.

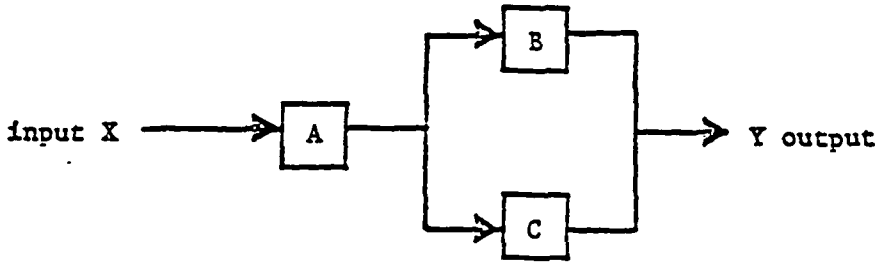
(15) Moret, B.M.E., Thomason, M.G., and Gonzalez, R.C. Optimization criteria for decision trees. To appear.

(16) Reactor Safety Study - An Assessment of Accident Risks in U.S. Commercial Nuclear Power Plants. WASH-1400. *NUREG-75/014. U.S. Nuclear Regulatory Commission, Washington, D.C., 1975.

(17) Thomason, M.G., and Page, E.W. Boolean difference techniques in fault tree analysis. Int'l J. Comp. & Inf. Sc. 5, 1(1976), 81-88.



(a)



(b)

Figure 1. A possible decision tree (a) for a simple system (b).

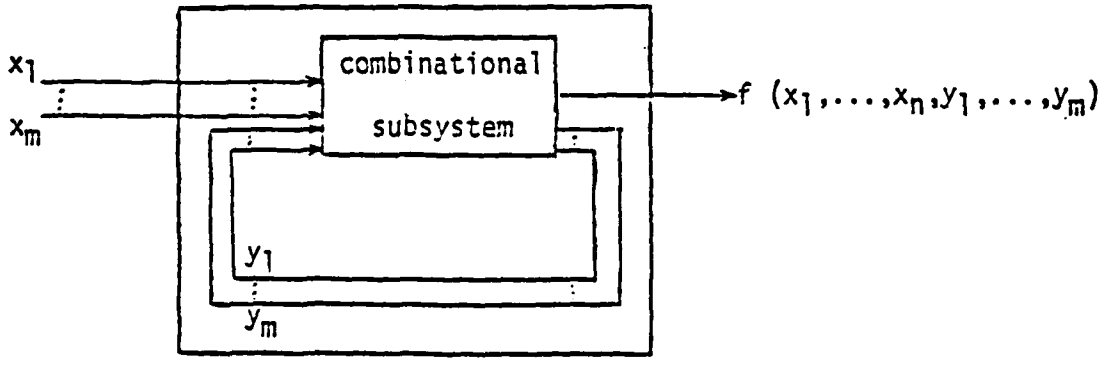


Figure 2. A system with memory (feedback) showing inaccessible internal variables.

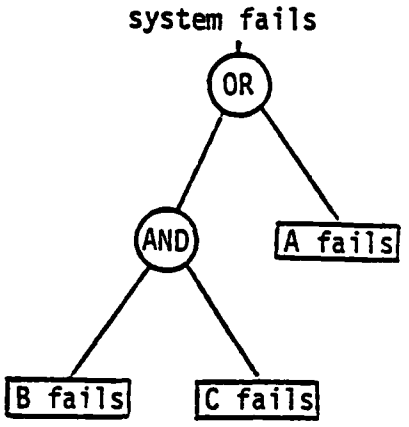


Figure 3. A fault tree for the system of Figure 1b.

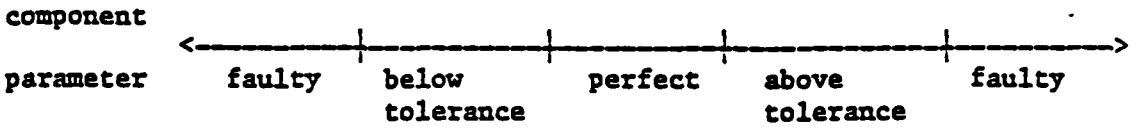


Figure 4. Subdivisions of the range of an analog signal.

OPTIMIZATION CRITERIA FOR DECISION TREES

Bernard M. E. Moret †
Michael G. Thomason ‡
Rafael C. Gonzalez ‡

ABSTRACT

Decision trees are a model of the sequential evaluation of discrete functions that have widespread applications in pattern recognition, taxonomy, decision table programming, databases, switching theory, and concrete complexity theory. Since a function in general has numerous decision tree representations, it is necessary to adopt some selection criterion in order to obtain the most appropriate representation. Several such optimization criteria have been proposed in the literature, but few have been studied together and the choice of a criterion has not often been directly addressed.

This paper regroups those criteria into a common, generalized framework, and examines their interrelationships. It is shown that, even in the simplest cases, most criteria cannot be optimized simultaneously, thereby disproving some conjectures found in the literature. Two new results are presented concerning the worst-case number of argument evaluations for Boolean functions. On the basis of the accumulated results, it is argued that two optimization criteria have widespread relevance; the computational complexity of these criteria is examined in detail.

Key Words and Phrases: computational complexity, decision diagram, decision tree, exhaustive Boolean function, identification tree, optimization criterion, sequential evaluation procedure, storage cost, testing cost.

CR Categories: 3.63, 3.7, 4.33, 4.34, 4.6, 5.39, 6.1, 8.3.

This work was supported by the Office of Naval Research, Arlington, Virginia, under contract No. 0014-78-C-0311.

† *Department of Computer Science, University of New Mexico, Albuquerque, NM 87131.*

‡ *Departments of Computer Science and of Electrical Engineering, University of Tennessee, Knoxville, TN 37916.*

1. Introduction

A decision tree is a model of the evaluation of a discrete function, wherein the value of a variable is determined and the next action (to choose another variable to evaluate or to output the value of the function) is chosen accordingly. Decision trees have many applications in pattern recognition [20,25], taxonomy and identification [5,6,11], decision table programming [1,15,16,21,22,24,26,28], switching theory [2,3,4,13], and analysis of algorithms [23,27]. More recently, they have been proposed as implementation-independent models of discrete functions with a view to the development of new complexity measures [18].

Since variables can be tested in arbitrary order during the sequential evaluation procedure, a given discrete function has, in general, numerous decision tree representations. Thus, it is necessary to develop some criterion for the selection of an appropriate tree, that is, to develop some measure on decision trees. Several mesures have been proposed in the literature [8,13,21,22], and the multiplicity of criteria presents the user with a problem of choice.

This article discusses several of these measures within a common framework of definitions and notation. After providing a formal definition of decision trees and expressing the various proposed criteria in the established framework, we briefly review the published optimization algorithms to place the optimization problem in perspective. The relationships between measures are then studied, beginning with the simplified case of Boolean functions (the case that is least conducive to incompatibilities). It is shown that, even in this case, almost all criteria are pairwise incompatible, that is, they cannot be simultaneously optimized for all functions. This disproves some conjectures found in the literature [2,28]. The special case of binary identification is then separately examined. Finally, we discuss each criterion in turn and examine its computational complexity. Several criteria are found to have limited applicability due to their specific behavior; in particular, we extend a result of Rivest [23] by showing that all symmetric and all linearly separable Boolean functions are exhaustive, i.e., have maximal worst-case testing cost. We conclude by suggesting that two specific measures, related to run-time cost and retention cost of trees, are the most generally useful optimization criteria at this time.

2. Preliminaries

The following formal definition of a decision tree appears in [18,19].

Definition 1. Let f be a (partial) function of discrete variables, $x_1, \ldots, x_n$, where variable x_i takes on m_i values (denoted $0, \ldots, m_i-1$). If f is a constant, then the decision tree for f consists of a single leaf labelled by that constant; otherwise, for each $x_i, 1 \leq i \leq n$, f has decision tree(s) composed of a root labelled x_i and m_i subtrees corresponding to the restrictions (hereafter called subfunctions) $f|_{x_i=0}, \ldots, f|_{x_i=m_i-1}$, in that order. □

It is noted that the same subtree may occur on several branches of the tree, in which case it may be desirable to use only one copy of that subtree by transforming the decision tree into a decision diagram with a rooted directed acyclic graph structure. To every decision diagram there corresponds a unique decision tree with a one-to-one correspondence between its paths and those in the tree.

Two costs are usually associated with each variable of a function: a *testing cost* measures the expense (in time or any resource associated with evaluation of that variable) incurred each time that variable is evaluated; and a *storage cost* measures the expense (in storage or any resource associated with the presence of that test) due to the presence of each test node labelled by that variable. In addition, a probability distribution is often specified on the variables' space and can be assumed uniform if not otherwise known. These data allow the computation of the following six measures.

Definition 2.

i) The total testing cost, η, is the sum, taken over all the paths from the root to the leaves, of the path testing costs, where the testing cost of a path is the sum of the testing costs of the variables evaluated on that path. When all testing costs are unity, η reduces to the external path length [12], itself a special case of the tree path entropy defined in [8].

ii) The normalized testing cost, H, is the total testing cost divided by the number of paths. When all testing costs are unity, H reduces to the average path length, itself a special case of the normalized tree path entropy [8].

iii) The worst-case testing cost, h, is the maximum path testing cost. When all testing costs are unity, h reduces to the worst-case number of tests, that is, the height of the tree or diagram.

iv) The expected testing cost, E, is the expected value of the path testing cost, where the probability of a path is the sum of the probabilities of all the combinations of variables' values that select that path.

v) The tree storage cost, α, is the sum, taken over all the internal nodes of the tree, of the storage costs of the associated variables. When all storage costs are unity, α is the total number of internal nodes of the tree.

vi) The diagram storage cost, β, is the same sum as in (v) taken over all the internal nodes of the diagram. □

It is noted that, in the case of unity testing and storage costs and uniform probability distribution, the only datum needed to compute the first five measures is the number of leaves at each level of the tree. Thus, a decision tree for a function of n variables can be entirely characterized by an (n+1)-tuple, $(\lambda_0, \ldots, \lambda_n)$, where λ_i is the number of leaves at level i. This notation is called the *leaf profile* [18] by analogy with a similar notation defined in [17]. The leaf profile induces a lexicographic ordering of decision trees, which in turn gives rise to two additional measures: the *maximum profile*, which ranks as best that tree which is largest in lexicographic order (on the grounds that leaves should be

encountered as soon as possible); and the *minimum reverse profile*, which ranks as best that tree which is smallest in reverse lexicographic order (on the grounds that the number of long paths should be minimized).

As an example of the above concepts, consider the Boolean function of four variables given by the formula,

$$f(x_1,x_2,x_3,x_4) = x_1x_2 + x_1x_4 + \bar{x}_2x_3.$$

Figure 1 shows a decision diagram and its corresponding decision tree for f; the various measures are as follows:

external path length, $\eta = 17$;
average path length, $H = 2.83$;
expected number of tests, $E = 2.375$;
tree node count, $\alpha = 5$;
diagram node count, $\beta = 4$;
leaf profile = (0,0,3,1,2).

In [18,19], decision trees and diagrams are extended to include composition and recursiveness, and it is shown that the measures defined above can be applied to this generalized case.

An interesting application of discrete functions is that of *binary identification*. As defined in [6], an identification problem consists of a set of objects, a set of binary questions, and an injective map from the set of objects to the power set of the set of questions; the image of an object is then the unique combination of positively answered questions which identifies that object. In the context of this paper, the questions are binary variables and the objects are values of a bijective partial function from the variables' space to the set of objects. It is readily seen that all decision trees for such a function have exactly one leaf for each object and thus have all the same number of nodes. Moreover, the fact that no two leaves have the same label means that there cannot exist common subtrees, so that all decision diagrams are in fact trees and, for each diagram, $\beta = \alpha$. By reason of these and other peculiarities, the case of binary identification will be treated independently in Section 5.

3. The Construction Of Optimal Decision Trees And Diagrams

The problem of constructing decision trees and diagrams that are optimal with respect to the various criteria has been addressed by numerous researchers using branch-and-bound techniques, dynamic programming, and various heuristics. A survey of their efforts can be found in [18].

Dynamic programming is of particular interest because several measures obey the "principle of optimality," that is, they have the property that an optimal solution can be built from optimal subsolutions. Indeed, if variable x_i, with testing cost t_i and storage cost s_i, is tested at the root of a decision tree for the function $f(x_1, ..., x_n)$, then the optimal values for such a tree for three of the measures are

$$h_{\min}(f) = t_i + \min\{h_{\min}(f|_{x_i=0}), \ldots, h_{\min}(f|_{x_i=m_i-1})\},$$

$$E_{\min}(f) = t_i + \sum_{j=0}^{m_i-1} p(x_i=j) \cdot E_{\min}(f|_{x_i=j}),$$

$$\alpha_{\min}(f) = s_i + \sum_{j=0}^{m_i-1} \alpha_{\min}(f|_{x_i=j}),$$

where $p(x_i=j)$ denotes the probability that x_i takes on the value j. Similarly, the leaf profile of this tree is the sum, component by component, of the leaf profiles of its subtrees. Hence, five out of the eight proposed measures obey the principle of optimality.

This approach is embodied in an algorithm designed to convert limited-entry decision tables into decision trees with minimal expected testing cost [1], later rediscovered [24] and refined [15]; a closely related procedure appears in [20]. This algorithm is easily adapted to any of the five possible criteria and to the most general type of decision tree [18]. For a function of n k-ary variables, the algorithm requires $O(n \cdot (k+1)^{n-1})$ steps; since a complete specification of the function necessitates an input of size $s = O(k^n)$, the time complexity is $O(s^{\log_k(k+1)} \cdot \log s)$. Dynamic programming offers an efficient solution to the optimization problem for those measures in the case of completely specified functions.

In the case of binary identification (and, more generally, of partial functions), however, a similar dynamic programming algorithm [6] is of exponential complexity because the specification of the problem is very much shorter than for complete functions, resulting in a much smaller input. Indeed, it has been proved [10,14] that the problem of constructing binary identification trees with minimal expected testing cost is NP-hard.

The remaining three measures, $\beta, \eta,$ and H, are not easily optimized. Branch-and-bound techniques, used for the optimization of E [3,21], have also been applied to the minimization of storage cost [22] for both trees and diagrams; however, such procedures are of exponential complexity. Little work appears to have been done on the optimization of η and H.

4. Compatibility Between Optimization Criteria

In this section, attention is focused on relationships between existing optimizaion criteria for the construction of decision trees.

4.1 Definitions

Given a function, f, and an optimization criterion, ω, let T_f^ω denote the set of all tree representations for f which optimize ω.

Definition 3. Let F be a class of functions and ψ, ω two optimization criteria. Then we say that, for that class of functions:

i) ψ and ω are equivalent, denoted $\psi \Leftrightarrow \omega$, if $\forall f \in F$, $T_f^\psi = T_f^\omega$.

ii) ψ is a special case of ω, denoted $\psi <= \omega$, if $\forall f \in F$, $T_f^{\psi} \supset T_f^{\omega}$.

iii) ψ and ω are compatible if $\forall f \in F$, $T_f^{\psi} \cap T_f^{\omega} \neq \emptyset$. □

Moreover, we shall say that ψ and ω are *strictly equivalent* if they are equivalent and the ordering of all the tree representations for any function in F is the same under both criteria.

It can now be shown that, in almost all cases, criteria are pairwise incompatible even in severely restricted classes of functions.

4.2 Results

We first examine the case that is least conducive to incompatibilities, namely that of completely specified Boolean functions with uniform probability distribution and unity costs. Figure 2 summarizes the findings for that class of functions (a $\emptyset$ entry means that the respective criteria are incompatible and a blank entry indicates that the exact relationship is unknown). Thus, most criteria are pairwise incompatible. In particular, α is not a special case of E; this can be seen by examining the trees for the Boolean function of four variables given by the formula

$$f(x_1,x_2,x_3,x_4) = x_1x_2 + \overline{x}_1x_3 + x_2\overline{x}_3x_4,$$

and thereby disproves conjectures found in [2, p.115 and 28, p.104].

We proceed to prove some of the relationships; other relationships appearing in Figure 2, but not explicitly proved in the text, are easily established by similarly constructed counterexamples.

Proposition 1. The maximum profile is incompatible with any other measure.
Proof: Two counterexamples will be used. First, let f_a be the Boolean function of four variables given by the formula

$$f_a(x_1,x_2,x_3,x_4) = \overline{x}_1x_3 + x_1x_2x_4 + x_2\overline{x}_3\overline{x}_4.$$

The trees with maximum profile have as the first test either x_1 or x_3 and also have minimal expected testing cost; the optimal tree for all other measures, however, tests x_4 first and is unique (except for the minimum diagram storage cost, which can also be attained by testing x_1 or x_3 first, but with a structure different from that of the maximum profile trees). The various measures for the three types of trees are listed in Table 1. The maximum profile (and, incidentally, the minimum expected testing cost) is thus incompatible with the minimum reverse profile, the diagram storage cost, and the total, normalized, and worst-case testing costs. Secondly, let f_b be the Boolean function of five variables given by the formula

$$f_b(x_1,x_2,x_3,x_4,x_5) = x_1x_5 + \overline{x}_1\overline{x}_2\overline{x}_5 + \overline{x}_2x_5\cdot(\overline{x}_3x_4 + x_3\overline{x}_4) + (x_2 + \overline{x}_5)\cdot(\overline{x}_3\overline{x}_4 + x_3x_4).$$

The trees with the maximum profile test x_5 first, while those optimal with respect to all other measures test x_1 first, with the results shown in Table 2. Hence the maximum profile is also incompatible with the minimum tree storage and expected testing costs. □
The function f_a above also shows that minimizing the tree or diagram storage costs

Table 1. First counterexample for Proposition 1.

First test	leaf profile	α_{min}	β_{min}	η_{min}	H_{min}	h_{min}	E_{min}
x_4	(0,0,0,8,0)	7	6	24	3.	3	3.
x_1 *or* x_3	(0,0,1,4,4)	8	6	30	$3.\overline{3}$	4	3.
x_1 *or* x_3	(0,0,2,2,4)	7	7	26	3.25	4	2.75

does not optimize any other criterion, while f_b yields the same conclusion for the minimum worst-case testing cost.

Proposition 2. The normalized testing cost is incompatible with any other measure (except, possibly, the worst-case testing cost); moreover, minimizing the normalized testing cost may involve the introduction of redundant tests.
Proof: Let f_c be the Boolean function of five variables given by the formula

$$f_c(x_1,x_2,x_3,x_4,x_5) = x_1x_2 + x_2 \oplus x_3 \oplus x_4 \oplus x_5,$$

where $\oplus$ stands for summation modulo 2. The optimal trees for all measures except H test x_1 or x_2 first and use no redundant test, while the trees with minimum normalized testing cost may test any variable first and, in case x_1 *or* x_2 is chosen, use a redundant test. Two diagrams rooted in x_1 are shown in Figure 3, the left being optimal with respect to all criteria but H, and the right being optimal for H; the corresponding measures are listed in Table 3. It is noted that the test of x_5 as the right child of the root is totally redundant. □

It is conjectured that, among the relationships with unknown status, several implications hold, most particularly $\eta \Rightarrow \alpha$. Clearly, however, the introduction of non-uniform probability distributions or non-unity costs renders all measures pairwise incompatible.

Before discussing the class of all discrete functions with arbitrary probability distributions and costs, we note the results for the class of partial Boolean functions with unity costs and uniform probability distribution on the domain, which are shown in Figure 4. Except for the obvious case of tree height and minimum reverse profile, no measure is a special case of any other; in fact, almost all measures are pairwise incompatible. The three counterexamples used to establish results beyond those of Figure 2 are omitted here for the sake of conciseness.

In the general case, it is easily seen that all measures are pairwise incompatible. In other words, one must face the problem of the choice of a criterion since, even for the simplest types of functions, it is not generally possible to optimize two criteria simultaneously.

Table 2. Second counterexample for Proposition 1.

First test	leaf profile	α_{min}	β_{min}	η_{min}	H_{min}	h_{min}	E_{min}
x_1	(0,0,1,1,8,4)	13	8	57	4.07	5	3.5
x_5	(0,0,1,2,2,12)	16	9	76	4.47	5	3.625

5. The Case Of Binary Identification

The various applicable measures will be examined first under the assumption of unity costs and uniform probability distribution of objects. Under these conditions, the storage cost of a tree reduces to its number of nodes (which is fixed, as noted in Section 2), and its expected testing cost is equal to its normalized testing cost, of which the path length is a fixed multiple. Hence it follows that only four criteria are applicable, namely, the height, the external path length, and the minimum reverse and maximum leaf profiles. The known relationships between the four measures are summarized in Figure 5 and can be established with a single couterexample as follows. Consider the identification problem with five objects, {a,b,c,d,e}, and four tests, T_1={a}, T_2={a,b}, T_3={a,b,c}, and T_4={a,b,c,d}. The trees with maximum profile test T_1 *or* T_4 first; those with minimum reverse profile and minimum path length test T_2 *or* T_3 first; and those with minimum height use any test first (but with different results if the chosen test is T_1 *or* T_4). The resulting measures are listed in Table 4. The exact relationship between the reverse profile and the path length criteria is not known; it is a simple matter, however, to construct an example which shows that they are not strictly equivalent. The introduction of non-uniform probabilities results in further incompatibilities and one more measure, the expected testing cost. In fact, the only two measures that are not incompatible are, trivially, the reverse profile and the height. Storage and testing costs impair the usefulness of leaf profiles (which do not reflect such data), but give rise to another criterion, the storage cost; all measures are then pairwise incompatible.

Even with unity costs and uniform distribution, the decision problem for the path length measure is known to be NP-complete [10]. The construction in [10] is a straightforward reduction from the exact cover by three sets (cf. [7, p.53]) and can be used to show that the decision problems for the reverse profile and the worst-case testing costs are also NP-complete. Finally, the decision problem for the maximum profile is clearly in NP, but it is not known to be NP-complete. Using standard extension and search techniques as developed in [7], one can show that the optimization problems for the storage cost and the total, expected, and worst-case testing costs are all NP-equivalent. The optimization problems for the profiles are both NP-easy since, although no polynomial-time algorithm is known for ranking profiles, one can simply use successive binary searches (one for each tree level) in order to establish the optimal profile. (In such a process, only the number of nodes at the searched level is important, so that one may set arbitrary values at the unknown levels.) Table 5 summarizes the known results about the complexity of decision tree optimization in binary identification problems.

Table 3. Counterexample for Proposition 2.

Tree	leaf profile	α_{min}	β_{min}	η_{min}	H_{min}	h_{min}	E_{min}
left	(0,0,2,0,0,16)	17	8	84	4.6	5	3.5
right	(0,0,0,4,0,16)	19	9	92	4.6	5	4.

6. An Assessment Of Optimization Criteria

Since the optimization problem for binary identification is NP-hard for most criteria, it follows that the general problem of optimization for (partial) functions is also NP-hard. However, there are large classes of functions for which the optimization problem is well-solved by the dynamic programming algorithm mentioned in Section 3, namely those functions, the specification of which requires an input of length exponential in the number of variables. Table 6 shows the complexity of optimization of each criterion in both cases of exponential- and polynomial-length inputs. It is noted that the optimization of normalized and expected testing costs under polynomial-length inputs is not known to be NP-easy: the arbitrary probability distribution and number of test outcomes enormously increases the number of possible values, to the point where even a binary search requires exponential time.

The difficulty of optimizing the normalized testing cost and its erratic behavior (as exemplified in Proposition 2, where the addition of a redundant test actually lowers the normalized testing cost), make it an undesirable criterion. Both leaf profile criteria lack generality, in that they cannot easily take into account arbitrary costs or probability distributions; therefore, they too are inappropriate measures, except in special situations. Finally, the tree storage cost is not an accurate reflection of actual memory or hardware requirements, because the diagram storage cost is never larger and often much smaller. For instance, a modulo 2 sum of n binary variables requires $O(2^n)$ tree nodes, but only O(n) diagram nodes. The diagram storage cost is a more relevant measure of implementation problems.

Of the three measures of testing cost, only h and E are concerned with the performance of a tree representation. The total testing cost, η, does not make use of the probability distribution, nor does it measure a worst-case extreme. Although it is of interest for binary identification problems as a measure of the cost incurred in producing each output of the function exactly once, it does not generally correspond to practical concerns that one has about a function. The worst-case testing cost, h, can be efficiently minimized and is certainly relevant in practical problems; however, it lacks discrimination power. Rivest [23] has shown that almost all (in the asymptotic sense) Boolean functions are exhaustive, i.e., have maximal worst-case testing cost, a result that we strengthen in the Appendix by proving that all symmetric and all linearly separable Boolean functions are exhaustive. Thus the worst-case testing cost does not discriminate between most Boolean functions and, within the sets of symmetric and threshold functions, it does not discriminate between any functions.

Table 4. The counterexample for binary identification.

First test	Leaf profile	Height	Path length
T_1 *or* T_4	(0,1,1,1,2)	4	14
T_2 *or* T_3	(0,0,3,2,0)	3	12
T_1 *or* T_4	(0,1,0,4,0)	3	13

The preceding considerations indicate that the expected testing cost, E, is the more generally useful measure of decision tree performance, while the diagram storage cost, β, is a relevant measure of decision tree implementation costs. These measures are examined in further detail in the following section.

6.1 The expected testing cost E

Given an intrinsic function of n variables, $f(x_1,...,x_n)$, for which testing variable x_i incurs cost c_i, the expected testing cost of any tree representation, T, of f is bounded by

$$\min \{c_i \mid 1 \le i \le n\} \le E(T) \le \sum_{i=1}^{n} c_i$$

These rather loose bounds can be tightened [19] to the following:

$$I(f) + \min \{l_f(x_i) \mid 1 \le i \le n\} \le E(T) \le \sum_{i=1}^{n} c_i - \max \{l_f(x_i) \mid 1 \le i \le n\},$$

where I(f) is the *intrinsic cost* of the function and $l_f(x_i)$ is the *loss* of variable x_i with respect to the function [19].

In fact, Boolean functions with unity costs and uniform probability distributions require an expected number of tests that converges to n; this can be shown as follows. Let B(n) be the number of Boolean functions of n variables and let J(n) be the number of those that are intrinsic; then

$$F(n) = 2^{2^n} \quad \text{and} \quad J(n) = \sum_{i=0}^{n} (-1)^{n-i} \cdot \binom{n}{i} \cdot F(i).$$

But almost all Boolean functions are intrinsic, so that we have

$$\lim_{n \to \infty} J(n)/F(n) = 1$$

with rapid convergence. Now, the expected value of E for a function of n variables, E(n), must be at least as large as
E(n-1) for non-intrinsic functions and equal to 1+E(n-1) otherwise; hence, we have the recurrence

$$E(n) \ge [(F(n)-J(n)) \cdot E(n-1) + J(n) \cdot (1+E(n-1))]/F(n) = E(n-1) + J(n)/F(n).$$

Since J(n)/F(n) rapidly converges to 1, the expected value of E is essentially equal to n for large values of n.

Table 5. The complexity of optimal binary identification.

Criterion	α_{min}	η_{min}, H_{min}	E_{min}	h_{min}	reverse profile	maximum profile
Complexity	NP-eqvlnt	NP-eqvlnt	NP-eqvlnt	NP-eqvlnt	NP-eqvlnt	NP-easy

This result, however, does not indicate that minimizing the expected testing cost is useless, because the presence of non-uniform costs and probabilities results in the large range of values described by the bounds given above. Moreover, the expected testing cost can be efficiently minimized, as indicated in Table 6.

The expected testing cost is the most frequently used criterion in the literature. In software applications, in particular, it is often of more interest to optimize the running time of a routine than to minimize its memory requirements. More generally,. one expects to find this criterion useful whenever a premium is placed on performance as opposed to acquisition cost.

6.2 The diagram storage cost β

The number of internal nodes of a binary decision diagram has been extensively studied in [13], where diagrams are called programs. It is shown that $O(2^n/n)$ nodes are sufficient to represent any Boolean function of n variables as compared with $O(2^n)$ for trees. This result is easily extended to show that $O(k^n/n)$ nodes are sufficient to represent any function of n k-ary variables (versus $O(k^n)$ for trees) [18]. Thus, in particular, a decision diagram is as succinct a representation of Boolean functions as is a simplified and factored Boolean formula.

The minimization of the diagram storage cost, however, is a difficult task. It cannot be done on a leaves-to-root scan because it requires that all subtrees be examined simultaneously. This precludes the use of dynamic programming and necessitates some form of top-down, backtracking method. Hence it must be suspected that the problem, which is clearly NP-easy, is also NP-hard (and thus NP-equivalent) even under exponential-length inputs. Indeed, the only existing algorithm, a branch-and-bound procedure [22], may exhibit exponential behavior by searching through almost all possible diagrams for a function.

The diagram storage cost has mostly been used in connection with hardware implementation of decision trees, such as multiplexer networks for Boolean functions [4,13]. In general, one expects to use this criterion whenever a premium is placed on acquisition or construction costs or when special constraints decrease the value of other measures. (An example of the latter is a synchronicity constraint, which requires all evaluations to take the same time and thus reduces the expected testing cost to the worst-case testing cost, h. When h is known to be maximal, as is the case with most Boolean functions, performance measures become altogether irrelevant.)

Table 6. Complexity of optimization criteria.

Criterion	Input size in function of number of variables	
	exponential	polynomial
α_{min}	low polynomial	NP-equivalent
β_{min}	?	NP-equivalent
η_{min}	?	NP-equivalent
H_{min}	?	NP-hard
h_{min}	low polynomial	NP-equivalent
E_{min}	low polynomial	NP-hard
min. rev. profile	low polynomial	NP-equivalent
maximum profile	low polynomial	NP-easy

7. Summary

Several measures used for the assessment of decision trees have been reviewed. It has been shown that they are pairwise incompatible in all but a few cases. This disproves some conjectures regarding the simultaneous optimization of those measures. Promising measures have been individually examined and two new results proved concerning the behavior of one measure in classes of Boolean functions. Based on the results presented, two measures, one concerning the run-time cost and the other the retention cost of trees, appear to be the most generally applicable at this time. The complexity of decision tree optimization under these two criteria was examined in detail.

8. References

[1] Bayes, A. J. A dynamic programming algorithm to optimize decision table code. Austral. Comp. J. 5, 2 (1973), 77-79.

[2] Breitbart, Y., and Reiter, A. Algorithms for fast evaluation of Boolean expressions. Acta Inf. 4 (1975), 107-116.

[3] Breitbart, Y., and Reiter, A. A branch-and-bound algorithm to obtain an optimal evaluation tree for monotonic Boolean functions. Acta Inf. 4 (1975), 311-319.

[4] Cerny, E., Mange, D., and Sanchez, E. Synthesis of minimal binary decision trees. IEEE Trans. Comp. TC-28, 7 (1979), 472-482.

[5] Chang, H. Y., Manning, E., and Metze, G. *Fault Diagnosis of Digital Systems.* John Wiley & Sons, New York, 1970.

[6] Garey, M. R. Optimal binary identification procedures. SIAM J. Appl. Math. 23, 2 (1972), 173-186.

[7] Garey, M. R., and Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco, 1979.

[8] Green, C. A path entropy function for rooted trees. J. ACM 20, 3 (1973), 378-384.

[9] Harrison, M. A. *Introduction to Switching and Automata Theory.* McGraw-Hill, New York, 1965.

[10] Hyafil, L., and Rivest, R. L. Constructing optimal binary decision trees is NP-complete. Inf. Proc. Letters 5, 1 (1976), 15-17.

[11] Jardine, N., and Sibson, R. *Mathematical Taxonomy.* John Wiley & Sons, New York, 1971.

[12] Knuth, D. E. *The Art of Computer Programming,* Volume 3: *Searching and Sorting.* Addison-Wesley, Reading, Mass., 1973.

[13] Lee, C. Y. Representation of switching circuits by binary-decision programs. Bell Syst. Tech. J. 38 (1959), 985-999.

[14] Loveland, D. W. Selecting optimal test procedures from incomplete test sets. Duke Univ. Tech. Rep. CS 1979-7.

[15] Martelli, A., and Montanari, U. G. Optimizing decision trees through heuristically guided search. Commun. ACM 21, 12 (1978), 1025-1039.

[16] Metzner, J. R., and Barnes, B. H. *Decision Table Languages and Systems.* Academic Press, New York, 1977.

[17] Miller, R. E., Pippenger, N., Rosenberg, A. L., and Snyder, L. Optimal 2,3-trees. SIAM J. Comp. 8, 1 (1979), 42-59.

[18] Moret, B. M. E. The representation of discrete functions by decision trees: aspects of complexity and problems of testing. Ph.D. thesis, Univ. of Tennessee, Knoxville, 1980.

[19] Moret, B. M. E., Thomason, M. G., and Gonzalez, R. C. The activity of a variable and its relation to decision trees. TOPLAS 2, 4 (1980), 580-595.

[20] Payne, H. J., and Meisel, W. S. An algorithm for constructing optimal binary decision trees. IEEE Trans. Comp. TC-26, 9 (1977), 905-916.

[21] Reinwald, L. T., and Soland, R. M. Conversion of limited-entry decision tables to computer programs I: minimum average processing time. J. ACM 13, 3 (1966), 339-358.

[22] Reinwald, L. T., and Soland, R. M. Conversion of limited-entry decision tables to computer programs II: minimum storage requirements. J. ACM 14, 4 (1967), 742-755.

[23] Rivest, R. L., and Vuillemin, J. On recognizing graph properties from adjacency matrices. Theor. Comp. Sc. 3 (1976), 371-384.

[24] Schumacher, H., and Sevcik, K. C. The synthetic approach to decision table conversion. Commun. ACM 19, 6 (1976), 343-351.

[25] Sethi, I. K., and Chatterjee, B. Efficient decision tree design for discrete variable pattern recognition problems. Patt. Rec. 9 (1977), 197-206.

[26] Verhelst, M. R. The conversion of limited-entry decision tables to optimal and near-optimal flowcharts: two new algorithms. Commun. ACM 15, 11 (1972), 291-313.

[27] Weide, B. A survey of analysis techniques for discrete algorithms. Comp. Surveys 9, 4 (1977), 291-313.

[28] Yasui, T. Some aspects of decision table conversion techniques. SIGPLAN Notices 6, 8 (1971), 104-111.

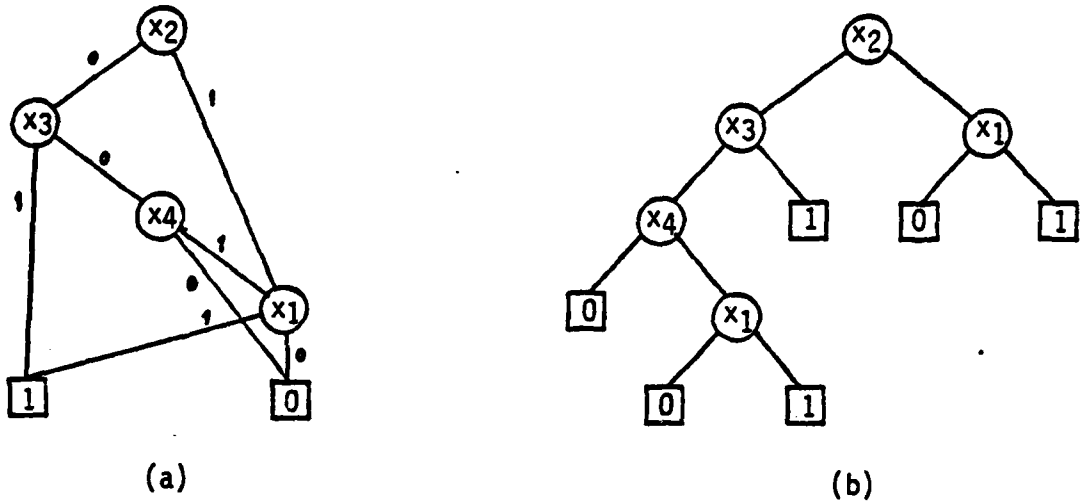


Figure 1. A decision diagram (a) and its associated decision tree (b).

	α_{min}	β_{min}	η_{min}	H_{min}	h_{min}	E_{min}	min. rev.prof.
β_{min}	=≠> <≠=						
η_{min}	<≠=	=≠> <≠=					
H_{min}	∅	∅	∅				
h_{min}	=≠> <≠=	=≠> <≠=	=≠>	=≠>			
E_{min}	=≠> <≠=	∅	∅	∅	∅		
minimum rev. profile	<≠=	=≠> <≠=		∅	==> <≠=	∅	
maximum profile	∅	∅	∅	∅	∅	∅	∅

Figure 2. Known relationships between the eight measures applicable

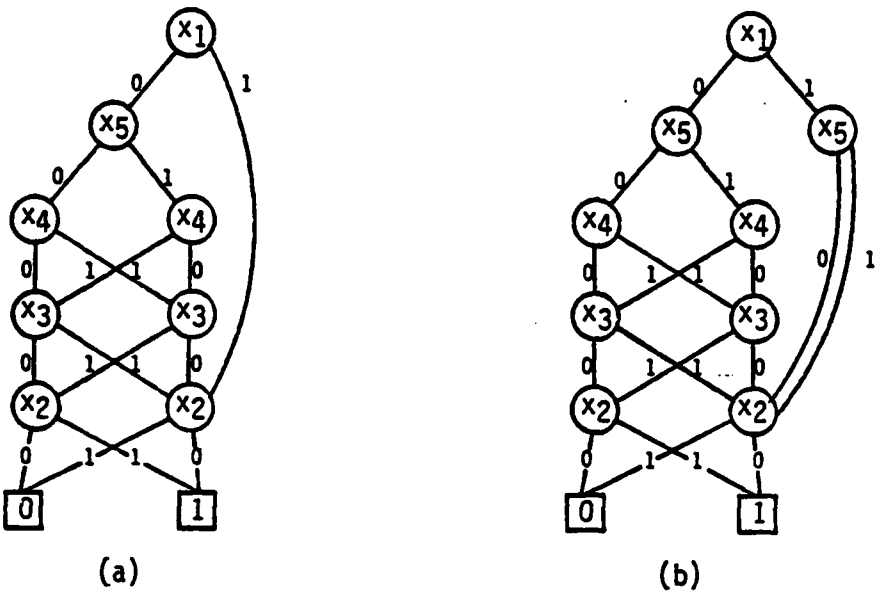


Figure 3. The two diagrams for the counterexample of Proposition 2.

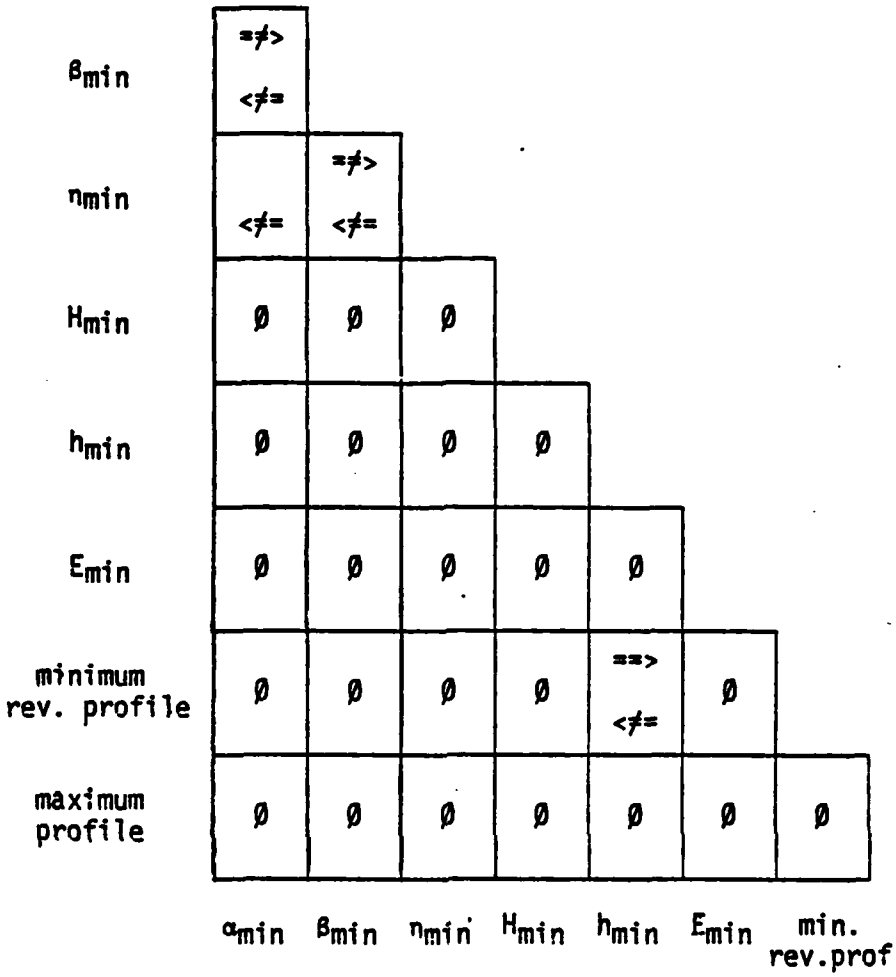


Figure 4. Known relationships between the eight measures applicable to binary decision trees for partially specified Boolean functions.

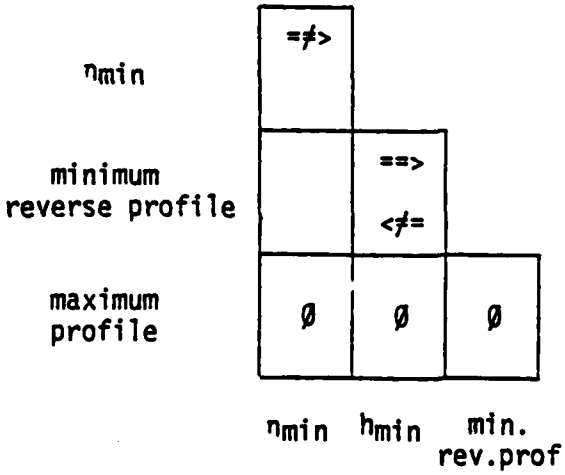


Figure 5. Known relationships between the four measures applicable to binary identification trees.

Symmetric and Threshold Boolean Functions Are Exhaustive

BERNARD M. E. MORET, MICHAEL G. THOMASON, AND RAFAEL C. GONZALEZ

Abstract—**The worst-case number of variable evaluations (testing cost) of Boolean functions is examined. Following up on a result by Rivest and Vuillemin, we show that all symmetric as well as all linearly separable Boolean functions are exhaustive, that is, have a pessimal worst-case testing cost.**

Index Terms—**Argument complexity, decision tree, multiplexer tree, threshold function, worst-case testing cost.**

I. Introduction

Rivest and Vuillemin [5] have shown that almost all (in the asymptotic sense) Boolean functions are exhaustive, that is, require in at least some cases that all of their variables be evaluated in order to find the function's value. Thus it is natural to suspect that there exist significant classes of Boolean functions in which every function is exhaustive. In this correspondence, we identify two such classes: symmetric functions and linearly separable (also known as threshold) functions.

Manuscript received July 27, 1982; revised January 3, 1983. This work was supported by the Office of Naval Research, Arlington, VA, under Contract 0014-78-C-0311.

B. M. E. Moret is with the Department of Computer Science, University of New Mexico, Albuquerque NM 87131.

M. G. Thomason is with the Department of Computer Science, University of Tennessee, Knoxville TN 37916.

R. C. Gonzalez is with the Department of Electrical Engineering, University of Tennessee, Knoxville TN 37916.

This result has implications in logic design, fault analysis, pattern recognition, and analysis of algorithms.

II. Preliminaries

Let $f(x_1, \cdots, x_n)$ be a Boolean function of n variables (arguments). A variable, x_i of f, is *redundant* if the function is independent of the value of that variable, i.e., $f|_{x_i=0} = f|_{x_i=1}$. A function without redundant variables is said to be *intrinsic*. A binary *decision tree* is a model of the sequential evaluation of a Boolean function, wherein the value of a variable is determined and the next action (to choose another variable to evaluate or to emit the value of the function) is chosen accordingly. Decision trees have found numerous applications in pattern recognition, taxonomy, logic design, decision table programming, fault detection, and analysis of algorithms (see [4] for further definitions and references).

From the definition, we see that the height of a decision tree corresponds to the maximum number of variables that had to be evaluated in order to determine the value of the function. The *argument complexity* of a function is then defined as the minimum height over all decision tree representations of that function. Thus, the argument complexity of a function is the minimum number of variables that must, in the worst case, be examined before the value of the function can be determined. A function is said to be *exhaustive* if its argument complexity is maximal (equal to the total number of variables). Rivest and Vuillemin [5] have used an elegant counting method to show that almost all Boolean functions are exhaustive. In the following we prove that all intrinsic[1] symmetric and linearly separable Boolean functions are exhaustive, using the specific properties of those classes.

III. The Main Results

A Boolean function of n variables, $f(x_1, \cdots, x_n)$, is said to be *symmetric* if and only if (*iff*), for each permutation, σ, over n letters,

$$f(x_{\sigma(1)}, \cdots, x_{\sigma(n)}) = f(x_1, \cdots, x_n).$$

Equivalently, a function is symmetric *iff* there exists a set of k numbers ($k \leq n$), $\{a_1, \cdots, a_k\}$, where $0 \leq a_1 < \cdots < a_k \leq n$, such that the function is equal to 1 exactly when a_i of its variables are equal to 1, for any i, $1 \leq i \leq k$ [3]. Such a function has a single decision tree structure; in particular, whenever $n - a_i$ variables have been found equal to 0, the remaining a_i variables must all be tested, since the function will be equal to 1 if all are equal to 1. This proves the following result.

Theorem 1: All (intrinsic) symmetric Boolean functions are exhaustive. □

Now let P be the defining property of a class of functions such that, if f possesses P, then both $f|_{x_i=0}$ and $f|_{x_i=1}$ possess P, for any choice of x_i; in other words, P is preserved by Shannon's decomposition. We then have the following characterization of exhaustiveness.

Proposition: All intrinsic functions in the class defined by P are exhaustive *iff*, in any Shannon's decomposition, at least one of their two subfunctions is intrinsic.

Proof: The *only if* part follows immediately from the definition of exhaustive: if f is a function of n variables and neither of its restrictions with respect to some variable x is intrinsic, then each of the restrictions has a decision tree of height no greater than $n - 2$, so that f has a decision tree of height $n - 1$ rooted in x and hence, is not exhaustive. For the *if* part, we use induction on the number of variables of the functions. All intrinsic functions of one variable are trivially exhaustive. Assume then that all intrinsic functions of $n - 1$ variables or less that answer the theorem's hypotheses are exhaustive. Consider a function f of n variables that answers the theorem's hypotheses. Any

[1] In the asymptotic sense again, almost all Boolean functions are intrinsic.

decision tree for f starts by testing one of the n variables. By assumption, for any variable x tested at the root, one of the restrictions $f|_{x=0}$ and $f|_{x=1}$ is intrinsic. By inductive hypothesis, that restriction is also exhaustive, since it is a function of $n-1$ variables that answers the theorem's hypotheses. Thus all decision trees for that restriction have height $n-1$; but then the decision tree for f rooted in x has height n. Since this holds for any choice of x, f is exhaustive. □

We first consider the class of *unate* functions—those functions representable by a Boolean formula where no variable appears in both complemented and uncomplemented form. Since decision trees are invariant under complementation of variables, it can be assumed without loss of generality that all variables are uncomplemented; this defines the class of *positive unate* functions, which are monotone increasing [3]. Both properties are easily seen to be preserved by Shannon's decomposition.

Let $f(x_1, \cdots, x_n)$ be an intrinsic positive unate function; then f is exhaustive *iff*, for each x_i, either $f|_{x_i=0}$ or $f|_{x_i=1}$ is intrinsic, that is, there cannot be found $x_j, x_k (j, k \neq i)$ such that $f|_{x_i=0}$ does not depend on x_j and $f|_{x_i=1}$ does not depend on x_k. Without loss of generality, let $i=1, j=2$, and $k=3$, and let $\underline{x}$ stand for $(x_4, \cdots, x_n)$. Then f is not exhaustive *iff*

$$f(0, 0, x_3, \underline{x}) = f(0, 1, x_3, \underline{x}) \text{ and } f(1, x_2, 0, \underline{x}) = f(1, x_2, 1, \underline{x}).$$

Since f is monotone increasing, it must be the case that

$$f(1, x_2, 1, \underline{x}) \geq f(0, 0, x_3, \underline{x}),$$

so that, by topological sorting, the following relations are obtained

$$f(1, 1, 1, \underline{x}) = f(1, 1, 0, \underline{x}) \geq$$
$$f(1, 0, 1, \underline{x}) = f(1, 0, 0, \underline{x}) \geq$$
$$f(0, 1, 1, \underline{x}) = f(0, 0, 1, \underline{x}) \geq$$
$$f(0, 1, 0, \underline{x}) = f(0, 0, 0, \underline{x}).$$

Let the four pairs of points above be denoted a, b, c, and d in that order. For any choice of $\underline{x}$, these four pairs can be mapped to the same value or to two distinct values (0 and 1), with the following partitions.

i) $(abcd)$ mapped to the same value; then x_1, x_2, and x_3 are redundant for that choice of $\underline{x}$.

ii) (abc) mapped to 1 and (d) to 0; then x_2 is redundant for that choice of $\underline{x}$.

iii) (ab) mapped to 1 and (cd) mapped to 0; then x_2 and x_3 are redundant for that choice of $\underline{x}$.

iv) (a) mapped to 1 and (bcd) to 0; then x_3 is redundant for that choice of $\underline{x}$.

The monotone property excludes any other choice. This shows that all unate functions of no more than three variables are exhaustive, since then there is no choice for $\underline{x}$ and one of the four partitions above must exist, contradicting the assumption of intrinsicalness. At the same time, it shows how to construct a nonexhaustive unate function of four variables; specifically, it is sufficient to find $\underline{x}' > \underline{x}''$ such that $f(x_1, x_2, x_3, \underline{x}')$ is partitioned according to ii) and $f(x_1, x_2, x_3, \underline{x}'')$ according to iv), since then x_2 is redundant in one case and x_3 in the other, but both are necessary overall. One such function is given by the formula

$$f(x_1, x_2, x_3, x_4) = x_1x_2 + x_1x_4 + x_3x_4.$$

It is easily verified that this function has decision tree representations of height 3.

Thus, not all unate functions are exhaustive; however, a subclass of these functions does possess the property.

A Boolean function, $f(x_1, \cdots, x_n)$, is *linearly separable* (is a *threshold* function) *iff* there exists a set of weights, $\{w_1, \cdots, w_n\}$, and a threshold, T, such that the function evaluates to 1 exactly when

$$\sum_{i=1}^{n} w_i x_i \geq T.$$

Again, it is easily seen that linear separability is preserved under Shannon decomposition. Since unate functions can be taken to be positive, weights and threshold are assumed positive without loss of generality. Let $\underline{w}$ stand for $(w_4, \cdots, w_n)$ and $\underline{x}'$ for the transpose of $\underline{x}$; substituting weights into the four pairs of function points yields

(a) $(w_1 + w_2 + \underline{w} \cdot \underline{x}'; w_1 + w_2 + w_3 + \underline{w} \cdot \underline{x}')$,
(b) $(w_1 + \underline{w} \cdot \underline{x}'; w_1 + w_3 + \underline{w} \cdot \underline{x}')$,
(c) $(w_3 + \underline{w} \cdot \underline{x}'; w_2 + w_3 + \underline{w} \cdot \underline{x}')$,
(d) $(\underline{w} \cdot \underline{x}'; w_2 + \underline{w} \cdot \underline{x}')$,

where any weight sum in (a) is no smaller than any weight sum in (b), and so on. As seen above, a function will not be exhaustive *iff* $\underline{x}' > \underline{x}''$ can be found such that $f(x_1, x_2, x_3, \underline{x}')$ is partitioned as (abc)(d) and $f(x_1, x_2, x_3, \underline{x}'')$ as (a)(bcd). Thus, the smallest sums of weights in any of (a), (b), and (c) must be larger than the largest sums of weights in (d) when $\underline{x}'$ is chosen (since the first are above the threshold while the second are below); similarly, the smallest sums of weights in (a) must be larger than the largest sums of weights in any of (b), (c), or (d) when $\underline{x}''$ is chosen. Using only the extremal sums—those closest to the threshold value, this implies, for the first partition, (abc)(d),

$$w_1 + w_2 + \underline{w} \cdot \underline{x}'^t > w_1 + w_3 + \underline{w} \cdot \underline{x}'^t,$$

and for the second, (a)(bcd),

$$w_3 + \underline{w} \cdot \underline{x}''^t > w_2 + \underline{w} \cdot \underline{x}''^t.$$

The first inequality yields $w_2 > w_3$ while the second implies $w_2 < w_3$, a contradiction. Hence, $\underline{x}'$ and $\underline{x}''$ cannot be found, and we have the following result.

Theorem 2: All (intrinsic) linearly separable Boolean functions are exhaustive. □

IV. Conclusion

Since the argument complexity of a function determines the worst-case performance of a sequential evaluation algorithm, our results show that no optimization is possible for symmetric and threshold Boolean functions. In particular, the total delay of a multiplexer (or sequential lookup) implementation of such functions [2] is fixed by the number of variables only. Similar remarks hold for sequential evaluations of linear decision functions in pattern recognition [6], longest paths through fault-trees (which usually describe unate—and often linearly separable—functions) [1], and software implementations of decision tables [4].

References

[1] R. E. Barlow, J. B. Fussell, and N. D. Singpurwalla Eds., *Reliability and Fault Tree Analysis*. Philadelphia, PA: SIAM Press, 1975.
[2] E. Cerny, D. Mange, and E. Sanchez, "Synthesis of minimal binary decision trees," *IEEE Trans. Comput.*, vol. C-28, pp. 472-482, July 1979.
[3] M. A. Harrison, *Introduction to Switching and Automata Theory*. New York: McGraw-Hill, 1965.
[4] B. M. E. Moret, "Decision trees and diagrams," *Comput. Surveys*, vol. 14, Dec. 1982.
[5] R. L. Rivest and J. Vuillemin, "On recognizing graph properties from adjacency matrices," *Theor. Comput. Sci.*, 371-384, 1976.
[6] J. T. Tou and R. C. Gonzalez, *Pattern Recognition Principles*. Reading, MA: Addison-Wesley, 1974.

SECTION III

FAULT TREES

Boolean difference techniques for time-sequence and common-cause analysis of fault-trees.†

B. M. E. Moret and M. G. Thomason.

ABSTRACT

Fault trees are a major model for the analysis of system reliability. In particular, Boolean difference methods applied to fault trees provide a widely used measure of subsystem criticality. This paper considers the generalization of the fault-tree model to time-varying systems and how time-dependent Boolean differences can be used for the analysis of such systems. In particular, suitable partial Boolean differences are shown to provide maximal and minimal solution sets for sensitization conditions. A method of common-cause failure analysis based on partial time-dependent Boolean differences is developed, which allows the study of failures due to repeated occurrences, at different times, of the same phenomenon. Finally, the application of those methods to systems with repair is studied; it is shown how, under certain assumptions of independence, steady-state distributions can be used for the analysis of system faults.

Authors' addresses:
B.M.E. Moret, Department of Computer Science,
The University of New Mexico, Albuquerque, N.M. 87131.
M.G. Thomason, Department of Computer Science,
The University of Tennessee, Knoxville, TN. 37916.

† This work was supported by the Office of Naval Research, Arlington, Virginia, under grant No. 0014-78-C-0311.

1. Introduction

Fault tree analysis is a method of major importance in reliability and safety studies [2,7,10,14]. A fault tree is a representation (using logic operations) of a Boolean function, the *structure function* of the system, which describes the set of elementary events (subsystem failures) necessary to make the system fail. When the structure function is monotone non-decreasing, that is, when a system is such that a failure of an additional subsystem cannot improve the system's status, the structure function is called *s-coherent* [2]. In this article, we restrict ourselves to such functions.

Of particular importance in system reliability studies is the determination of a component's criticality, that is, of a component's influence on the behavior of the system. Among several criticality measures [8], the most commonly used is Birnbaum's importance measure, which is simply the probability that the system is in a state in which the functioning of the component completely determines that of the system. (That is, the system fails if the component fails, works if the component works.) This measure can be obtained by using the Boolean difference operator [12], a powerful analytical tool for combinational logic expressions, in particular when a large number of variables is involved. Recall that the Boolean difference of a Boolean function, f, with respect to one of its variables, x, is the function

$$\frac{df}{dx} = f_{|x=0} \oplus f_{|x=1},$$

where $\oplus$ stands for exclusive-or and $f_{|x=0}$ denotes the restriction of f to that part of its domain where x takes the value 0.

There is interest in fault tree analysis also in systems in which the configuration of components required to cause failure changes at a finite number of discrete points during the interval in which the system is in operation. The term of *phased mission* [6] has been used to describe such time-varying systems. In effect, variations with time force one to consider sequential rather than combinational logic functions as the structure functions for the system's description. In such systems, the importance of a component can only be measured in each separate phase by conventional methods, since the measure is itself time-dependent. However, it is important to evaluate the effects of individual components over the system's operation time, as well to attempt to identify underlying causes (known as *common causes* [5,15]) for time-dependent failures.

In the following, we show how the concept of time-dependent Boolean differences [9] can be used to develop methods for the analysis of sequential (rather than combinational) structure functions. Methods for the determination of sensitization conditions, path dependences, and measures of importance are illustrated. We show that partial Boolean differences, taken with respect to suitable subfunctions, allow the determination of maximum and minimum sets of conditions for sensitization and criticality measurements. We then develop a new method for common cause analysis using partial Boolean differences and illustrate it on a phased mission example. We conclude by showing how the above methods can be applied to systems with repair, using steady-state distributions under mild assumptions of independence.

2. Time-dependent Boolean differences, sequential functions, and phased missions

Time-dependent Boolean differences were introduced in [9] as a tool for the analysis of sequential logic functions. A time superscript is used on switching expressions (single variables or more complex expressions) to denote their value during a specific time interval relative to some reference starting point. In effect, the superscripts create distinct variables for each time reference.

To illustrate these concepts briefly in the original context of sequential digital networks, we consider the circuit of Figure 1, composed of an AND gate, an OR gate, and two D-type (delay) flip-flops. The value of the primary output, Z, at time t, Z^t, is given by the sequential Boolean function:

$$Z^t = X_1^t + X_1^{t-2} \cdot X_2^{t-2}.$$

To determine the conditions which make Z^t dependent upon X_1^t (that is, such that X_1^t is critical), we compute the Boolean difference of Z^t with respect to X_1^t:

$$\frac{dZ^t}{dX_1^t} = \bar{X}_1^{t-2} + \bar{X}_2^{t-2}.$$

The solutions of $dZ^t/dX_1^t = 1$ give the necessary and sufficient conditions, both in logical value and timing, for X_1^t to be critical for Z^t, namely X_1=0 or X_2=0 at time t-2. For dependence of Z^t on X_1^{t-2}, we compute

$$\frac{dZ^t}{dX_1^{t-2}} = \bar{X}_1^t \cdot X_2^{t-2},$$

thus establishing the conditions X_1=0 at t and X_2=1 at t-2. Note that dZ^t/dX_1^t=0 for

$\tau \neq t$ and $\tau \neq t-2$, reflecting the fact that X_1 can only influence Z^t at times $t-2$ and t.

Specific path dependencies can be isolated by partial Boolean differences. For example, if the dependence of Z on X_1 via the path $X_1 \rightarrow A \rightarrow B \rightarrow C \rightarrow Z$ is desired, we compute the chain of derivatives:

$$\frac{dA^{\tau_3}}{dX_1^{\tau_1}} \cdot \frac{dB^{\tau_3}}{dA^{\tau_2}} \cdot \frac{dC^{\tau_4}}{dB^{\tau_3}} \cdot \frac{dZ^{\tau_5}}{dC^{\tau_4}},$$

wherein the linking of the time sequence requirements is reflected in the time superscripts. Computing these derivatives at actual time intervals yields:

$$\frac{dA^t}{dX_1^t} = X_2^t \qquad \frac{dB^{t+1}}{dA^t} = 1$$

$$\frac{dC^{t+1}}{dB^{t+1}} = 1 \qquad \frac{dZ^{t+2}}{dC^{t+2}} = \overline{X}^{t+2}.$$

Thus the chain yields:

$$\frac{dZ^{t+2}}{dX_1^{t+2}} = X_2^t \cdot 1 \cdot 1 \cdot \overline{X}_1^{t+2},$$

in accordance with our earlier computation of dZ^t/dX_1^{t-2}.

As noted above, the concept of a system's requirements changing at a finite number of points over an interval of interest essentially converts its structure function into a sequential logic function, so that the appropriate analytical method becomes the time-dependent Boolean difference. In each time interval, the structure function is s-coherent. We illustrate these concepts by applying them to a simplified example of phased mission, due to Esary and Ziehms [6].

The example considers the interaction of a fire department, which operates three vehicles, a large fire engine (M), a tanker (T), and a light truck (L), with a chemical plant, the safety equipment of which consists of a sprinkler system (S), a hydrant (H), and a special chemical fire extinguisher system (F). A fire at the plant can be decomposed in three phases. In the initial stage, the large engine or the light truck combined with the sprinkler system will allow time for evacuation. In the second stage, the special chemical extinguisher system is needed to contain the fire, together with either the large engine or the light truck; the needed water can be supplied by the hydrant or, if necessary, by the tanker through the large engine's pumps. In the last phase, the fire is

brought under control by the special system or by the large engine; again, the needed water can come from the hydrant or the tanker.

Thus we have a six component, three phase system, described by the block diagram of Figure 2. The whole system works iff each succeeding phase works in turn. Thus the system's success function is the product of the three phases' success functions. The three phases are described by the functions

$$P_1 = S^t L^t + M^t,$$

$$P_2 = F^t \cdot \left(T^t M^t + H^t \cdot (M^t + L^t) \right)$$

$$P_3 = F^t H^t + M^t \cdot (T^t + H^t)$$

Hence the system's success function is, at time t:

$$Succ^t = P_1^{t-2} \cdot P_2^{t-1} \cdot P_3^t.$$

This is a sequential logic function. (Note that it is a particularly simple type of phased mission, since the several phases do not mix or interact.) We can analyze each phase separately, such as by finding under which conditions the availability of the large fire engine, M, is critical in phase 1:

$$\frac{dP_1}{dM} = 1 \iff \frac{d(SL+M)}{dM} = 1 \iff \bar{S}+\bar{L} = 1 \iff S \text{ or } L \text{ fails.}$$

However, we can use time differences to find under which conditions the availability of the same component in phase 1 is critical to the success of the mission:

$$\frac{dSucc^t}{dM^{t-2}} = 1 \iff$$

$$(\bar{S}^{t-2} + \bar{L}^{t-2}) \cdot F^{t-1} \cdot \left(T^{t-1} M^{t-1} + H^{t-1} \cdot (M^{t-1} + L^{t-1}) \right) \cdot \left(F^t H^t + M^t \cdot (T^t + H^t) \right) = 1,$$

which says, of course, that those conditions are that either S or L fails in phase 1 while phases 2 and 3 are successful.

Yet of more interest is the criticality of the same component throughout the mission, under the assumption that a failure at time τ implies that the component stays faulty for $t > \tau$ (no repair).

$$\frac{d^3 Succ^t}{dM^{t-2} dM^{t-1} dM^t} = 1 \iff \bar{S}^{t-2} F^{t-1} \bar{L}^{t-1} \cdot (\bar{F}^t H^t + \bar{F}^t T^t + \bar{H}^t T^t) = 1.$$

This triple Boolean difference gives the conditions under which the functioning of

component M is critical in every phase of the mission; that is, under these conditions, each phase of the system reduces to component M. Other conditions of interest include those under which the status of a component in some phase is critical, or those under which specific combinations of components become critical; the next section develops a general approach to the derivation of such conditions with Boolean differences.

3. Some properties of Boolean differences

When analyzing a system, it is often desired to determine its sensitivity to various modes of failure of its components. While standard (multiple) Boolean differences allow the determination of a system's sensitivity to a particular sequence of component failures, they cannot provide the answer to such questions as: "If one of two components fails, under which conditions will the system certainly fail? possibly fail?"

In order to answer such questions, we turn to Boolean differences with respect to subfunctions, a generalization of the conventional Boolean differences with respect to variables. The Boolean difference of the function, $f(X_1, \ldots, X_n)$ with respect to the subfunction $g(X_{i_1}, \ldots, X_{i_k})$, where $k \leq n$ and $1 \leq i_j \leq n$, is defined as

$$\frac{df}{dg} = f_{|g=1} \oplus f_{|g=0},$$

which definition exactly parallels that of standard Boolean differences (indeed, a single variable is a--very simple--subfunction). In the definition, we have

$$f_{|g=1} = \bigcup f_{|(X_{i_1}, \ldots, X_{i_k})},$$

where the union (inclusive-or) is taken over all k-tuples which are minterms of g (i.e., which make g take the value of 1). For instance, with $f(A,B,C)$, $g(A,B)=A+B$, and $h(A,B)=A \cdot B$, we have

$$\frac{df}{dg} = f_{|A+B=1} \oplus f_{|A+B=0}$$

$$= \left(f_{|A=B=1} + f_{|A=0,B=1} + f_{|A=1,B=0}\right) \oplus f_{|A=B=0},$$

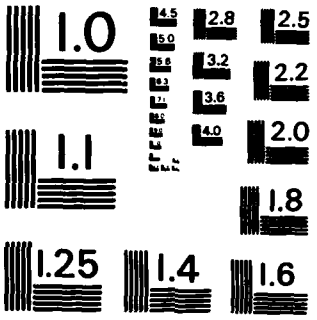
$$\frac{df}{dh} = f_{|A \cdot B=1} \oplus f_{|A \cdot B=0}$$

$$= f_{|A=B=1} \oplus \left(f_{|A=B=0} + f_{|A=0,B=1} + f_{|A=1,B=0}\right).$$

Because of the symmetric definition of Boolean differences and since $X \oplus Y = \bar{X} \oplus \bar{Y}$, we

AD-A148 465 COMPLEXITY TESTABILITY AND FAULT ANALYSIS OF DIGITAL ANALOG AND HYBRID SY..(U) TENNESSEE UNIV KNOXVILLE DEPT OF ELECTRICAL ENGINEERING R C GONZALEZ ET AL. 2/2

UNCLASSIFIED 30 SEP 84 TR-EE-84-50 N00014-78-C-0311 F/G 12/1 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

clearly have

$$\frac{df}{dg} = \frac{df}{d\bar{g}} = \frac{d\bar{f}}{dg} = \frac{d\bar{f}}{d\bar{g}}.$$

In particular, let S be the success function of a system, $F=\bar{S}$ its failure function, and S_1, F_1 the success and failure functions of a subsystem; we then have

$$\frac{dS}{dS_1} = \frac{d\bar{S}}{d\bar{S}_1} = \frac{dF}{dF_1}.$$

It follows that whatever results are developed for success functions remain unchanged for failure functions; thus, without loss of generality, we shall from now on use only success functions.

Given a system, S, and two components, X and Y, we wish to know how the failure of one or both of the components will affect the system's behavior. There are three failure modes to consider:

i. both X and Y originally work and they fail simultaneously;

ii. at least one of X and Y originally works and both eventually fail;

iii. both X and Y originally work and at least one of them fails.

The first mode corresponds to a change from $(X,Y)=(1,1)$ to $(X,Y)=(0,0)$; the second corresponds to a change in the value of the function $g=X+Y$; and the third corresponds to a change in the value of the function $h=X\cdot Y$. This leads us to consider the functions:

i. $S_{|X=Y=0} \oplus S_{|X=Y=1}$;

ii. $\frac{dS}{dg}$;

iii. $\frac{dS}{dh}$.

The following result defines the relationship (illustrated in Figure 3) between these and other Boolean differences.

Theorem 1: Let S be the success function of a system, X, Y the success functions of two of its components. Then

$$\left\{\frac{dS}{d(X\cdot Y)}\right\} = \left\{\frac{dS}{d(X\oplus Y)}\right\} \subseteq \left\{\frac{d^2S}{dXdY}\right\} \subseteq \left\{\frac{dS}{d(X+Y)}\right\} = \left\{S_{|X=Y=0} \oplus S_{|X=Y=1}\right\},$$

where $\{f\}$ denotes the set of minterms of f. □

Notice that the conditions expressed by $\left\{\frac{dS}{d(X\cdot Y)}\right\}$ are such that the failure of either X or Y will precipitate that of S, while those expressed by $\left\{\frac{dS}{d(X+Y)}\right\}$ are such that the failure of both X and Y may be necessary to cause that of S. This is formalized in the following result.

Corollary 1: $\left\{\frac{dS}{d(X\cdot Y)}\right\}$ and $\left\{\frac{dS}{d(X+Y)}\right\}$ are the minimum, respectively maximum, solution sets for the sensitization of S to the subsystems X and Y. □

The inclusions stated in the theorem are in general proper; however, one or both may degenerate into an equality. In this case, we have the following results.

Corollary 2:

i. $$\frac{dS}{d(X\cdot Y)} = \frac{d^2S}{dXdY} \Rightarrow \left\{S_{|X=Y=0}\right\} = \left\{S_{|X=1,Y=0}\right\} \cap \left\{S_{|X=0,Y=1}\right\}$$

that is, if the system works equally with only one subsystem or the other functioning, it will work without either subsystem.

ii. $$\frac{d^2S}{dXdY} = \frac{dS}{d(X+Y)} \Rightarrow \left\{S_{|X=1,Y=0}\right\} = \left\{S_{|X=0,Y=1}\right\}$$

that is, the system's success function is symmetric in X and Y.

iii. $$\frac{dS}{d(X\cdot Y)} = \frac{d^2S}{dXdY} = \frac{dS}{d(X+Y)} \Rightarrow$$
$$\left\{S_{|X=Y=0}\right\} = \left\{S_{|X=1,Y=0}\right\} = \left\{S_{|X=0,Y=1}\right\}$$

that is, the system only depends on whether or not X and Y both work (only the value of the function $X\cdot Y$ need be known, rather than the individual status of X

and Y). □

As an example, consider again the phased mission example presented above. If we need to find under which circumstances an initial failure of either the light truck or the tanker will precipitate the failure of the mission, we need to consider the partial Boolean difference

$$\frac{d^3 Succ^t}{(dL^{t-2}dL^{t-1}dL^t)\cdot(dT^{t-2}dT^{t-1}dT^t)},$$

whereas, if we are concerned with the influence of the simultaneous failure of both subsystems in the last phase, then we must consider the partial Boolean difference

$$\frac{dSucc^t}{d(L^t+T^t)}.$$

4. Common cause failure analysis

A common cause may be defined as an event which precipitates the failure of one or more components of a system, yet is not explicitly described in the system [5,14,15]. For instance, in a fault-tree describing how an integrated circuit could fail, primary events may include cracked die, loose bondings, input and output short-circuits, all events which could be have been caused by excessive mechanical or thermal stresses. Thus vibrations and temperature, although not explicitly mentioned in the fault tree, could be a major factor in that circuit's reliability analysis.

Several approaches have been proposed for common-cause analysis (see [14] for a brief survey), using probabilistic or logic methods, and trying to identify common causes or to assess their consequences. We outline below a method based on partial Boolean differences, which is more flexible than most methods proposed to date and lends itself to both qualitative and quantitative analysis.

A common cause may have more complex direct consequences than the simple failure of a number of components; in particular, the failure of a component may protect another from the common event's effects. Thus, common cause analysis cannot proceed in a general manner by substituting specific component failures for the common event. Rather, the common cause must be represented as a (not necessarily s-coherent) Boolean function, call it C, and the effects of $\{C\}$ must be investigated. This can be easily done with partial Boolean differences. Specifically, the common cause event, C,

will precipitate the failure of the system, S, exactly when

$$\frac{dS}{dC} = 1.$$

Examining the cut sets for dS/dC allows a qualitative analysis of the common event's effects, while the probability of its being fatal to the system is directly obtained by computing the probability of the set $\{dS/dC\}$. Furthermore, the use of time-dependent Boolean differences allows us to consider the time-sequence effects of common causes.

Thus a complete common cause analysis would proceed by first determining common causes of interest and expressing their effects on the system's components as a (time-dependent) Boolean function, then computing the Boolean differences, and finally extracting minterms, cut sets, etc., as needed for the analysis. It is noted that all of the Boolean operations involved in computing Boolean differences are elementary and can be easily carried out by an automatic system (such as the SETS program [13]).

Returning to our phased-mission example, consider the influence of the common cause event that results in cutting the water supply at the site. As a result, both the hydrant and the sprinkler system will fail, so that the common event can be written as

$$C = \overline{H} \cdot \overline{S}.$$

If that event occurs during the second phase of the mission, then the continuing success of the mission will depend on the event if the following Boolean difference evaluates to 1:

$$\frac{d^2 Succ^t}{dC^{t-1} dC^t} = P_1^{t-2} \cdot F^{t-1} \cdot \left(M^{t-1} \overline{T}^{t-1} + L^{t-1} \cdot (\overline{T}^{t-1} + \overline{M}^{t-1}) \right) \cdot \left(F^t \cdot (\overline{T}^t + \overline{M}^t) + M^t \overline{T}^t \right).$$

5. Simple steady-state systems with repair

Allowing for the possibility of repair of faulty subsystems results in a much more complex system. A steady-state condition can be established within each phase, however, and analyzed with standard modelling techniques, such as queueing theory (see, e.g. [11]).

A common assumption in such analyses is that of time-independence; both the failure and the repair processes are treated as Poisson processes, so that the behavior of the system in a phase can be derived from the knowledge of just the failure and repair

rates. In a phased mission, the time analysis is complicated by the presence of phase transitions, which may result in a phase being initiated with previous failures still present or already repaired, depending on the interaction of the phase transitions and the failure and repair processes. However, the assumption of time-independence allows us to complete the analysis on a phase-by-phase basis, which also allows for the possibility of phase-dependent failure and repair rates. Moreover, and more importantly, the Boolean difference methods developed above are still applicable. (Since the success functions of the various subsystems are unchanged, the difference is just in the probability computations.) Finally, queueing methods permit the analysis of phased missions where the change of phase is itself a random process (e.g., because it is triggered by external events). In fact, if the phase transitions are themselves time-independent processes, the analysis can be done by superposing two finite-state models, with resulting states describing the functioning of all subsystems as well the present phase.

Since the number of states in the final model grows exponentially as a function of the number of system components, we present a very simple example. Our system has three phases and two distinct components, as shown in Figure 4. We let λ_A,λ_B be the failure rates of components A and B, respectively, and μ_A,μ_B their repair rates; the phase dependence of the the rates is indicated by a superscript, according to our time notation; finally, the rate of a transition from phase i to phase j is indicated by δ_{ij}. The resulting model has $2^2 \cdot 3 = 12$ states, as shown in Figure 5, where each state is labelled by three digits, denoting, in that order, the functioning of component A (1 for working, 0 for failing), that of component B, and the phase number. (Note that transitions represent only a single change in the system, since simultaneous changes have zero probability.)

The steady-state equations (which we can write a a homogeneous linear system by using the Markov transition rate matrix) describe a balanced flow in and out of each state; together with the binding equation stating that the system must be in one of the 12 states, they allow us to solve for the probability of occupation of any state. Since each state is also a unique point in the space of the time-dependent structure functions, we can find the probability of any set of minterms by summing the probabilities of occupation of the corresponding states (again, all of this is easily expressed in matrix form).

6. Conclusion

We have investigated the use of Boolean difference methods for time-sequence and common cause analysis of coherent systems, as represented by fault-trees. In particular, we have shown how specific partial, time-dependent, Boolean differences can be used for the derivation of minimum and maximum sensitization conditions and for the analysis of complex common causes. We have also shown that such methods generalize without changes to systems with repairs, as long as events are assumed to be time-independent. We conclude that Boolean difference methods, which have been and still are widely used for fault detection, have considerable potential in reliability and sensitivity analysis applications.

References

[1] Akers, S. B. Probabilistic techniques for test generation from functional descriptions. Proc. IEEE Symp. on Fault-Tolerant Comp., 1979, 113-116.

[2] Barlow, R. E., *et al.*, eds. *Reliability and Fault-Tree Analysis: Theoretical and Applied Aspects of System Reliability and Safety Assessment.* SIAM, Philadelphia, 1975.

[3] Bell, N. J., Page, E. W., and Thomason, M. G. Extension of the Boolean difference concept to multi-valued logic systems. Proc. Symp. on Theory and Appl. of Multiple-Valued Logic Design, Buffalo, 1972.

[4] Chang, H. Y., Manning, E., and Metze, G. *Fault Diagnosis of Digital Systems.* John Wiley & Sons, New York, 1970 (Chap. 3).

[5] Easterling, R. G. Probabilistic analysis of common mode failures. Proc. ANS Conf. on Prob. Analysis of Nuclear Reactor Safety, Newport Beach, 1978.

[6] Esary, J. D., and Ziehms, H. Reliability analysis of phased missions. In: *Reliability and Fault-Tree Analysis: Theoretical and Applied Aspects of System Reliability and Safety Assessment.* SIAM, Philadelphia, 1975.

[7] Fussell, J. B., Powers, G. J., and Bennetts, R. G. Fault-trees–a state of the art discussion. IEEE Trans. Rel. R-23, 1 (1974), 51-55.

[8] Lambert, H. E. Measures of importance of events and cut sets in fault trees. In: *Reliability and Fault-Tree Analysis: Theoretical and Applied Aspects of System*

Reliability and Safety Assessment. SIAM, Philadelphia, 1975.

[9] Marinos, P. N., Page, E. W., and Thomason, M. G. Fault-detection in sequential networks using time-dependent Boolean differences. Proc. 5th IEEE Comp. Soc. Conf., Boston, 1971.

[10] *Reactor Safety Study* - An Assessment of Accident Risks in US Commercial Nuclear Power Plants. WASH-1400. *NUREG-75/014. US Nuclear Regulatory Commission, Washington, D.C., 1975.

[11] Siewiorek, D.P., and Swarz, R.S. *The Theory and Practice of Reliable System Design.* Digital Press, Mass., 1982 (Chap. 5).

[12] Thomason, M. G., and Page, E. W. Boolean difference techniques in fault tree analysis. Int'l J. Comp. & Inf. Sc. 5, 1 (1976), 81-88.

[13] Worrell, R. B. Set equation transformation system (SETS). SLA-73-0028A, Sandia Laboratories, Albuquerque, 1974.

[14] Worrell, R. B., and Burdick, G. R. Quantitative analysis in reliability and safety studies. IEEE Trans. Rel. R-25, 3 (1976), 164-170.

[15] Worrell, R. B., and Stack, D. W. A Boolean approach to common cause analysis. Proc. IEEE Annual Rel. & Maintainability Symp., 1980, 363-366.

Appendix

Proof of Theorem 1: By definition, we have

$$\frac{dS}{d(X\cdot Y)} = S_{\,|\,X=Y=1} \oplus \left(S_{\,|\,X=1,Y=0} + S_{\,|\,X=0,Y=1} + S_{\,|\,X=Y=0}\right).$$

Since S is s-coherent and (0,0) < (0,1), $S_{\,|\,X=Y=0}$ is absorbed in $S_{\,|\,X=0,Y=1}$ (since, whenever the first term has value 1, the second must have value 1 to maintain s-coherence); thus we get

$$\frac{dS}{d(X\cdot Y)} = S_{\,|\,X=Y=1} \oplus \left(S_{\,|\,X=0,Y=1} + S_{\,|\,X=1,Y=0}\right).$$

Similarly,

$$\frac{dS}{d(X\oplus Y)} = \left(S_{\,|\,X=Y=0} + S_{\,|\,X=Y=1}\right) \oplus \left(S_{\,|\,X=0,Y=1} + S_{\,|\,X=1,Y=0}\right),$$

but $S_{\,|\,X=Y=0}$ gets absorbed in $S_{\,|\,X=Y=1}$, so that we get

$$\frac{dS}{d(X\cdot Y)} = \frac{dS}{d(X\oplus Y)},$$

whence our first equality. Again, by definition,

$$\frac{dS}{d(X+Y)} = S_{\,|\,X=Y=0} \oplus \left(S_{\,|\,X=Y=1} + S_{\,|\,X=0,Y=1} + S_{\,|\,X=1,Y=0}\right),$$

but both $S_{\,|\,X=0,Y=1}$ and $S_{\,|\,X=1,Y=0}$ get absorbed in $S_{\,|\,X=Y=1}$, so that we get

$$\frac{dS}{d(X+Y)} = S_{\,|\,X=Y=0} \oplus S_{\,|\,X=Y=1},$$

whence our second equality.

Finally, we have

$$\begin{aligned}\frac{d^2S}{dXdY} &= \frac{d}{dY}\left(S_{\,|\,X=0} \oplus S_{\,|\,X=1}\right)\\ &= \left(S_{\,|\,X=0} \oplus S_{\,|\,X=1}\right)_{\,|\,Y=0} \oplus \left(S_{\,|\,X=0} \oplus S_{\,|\,X=1}\right)_{\,|\,Y=1}\\ &= S_{\,|\,X=Y=0} \oplus S_{\,|\,X=0,Y=1} \oplus S_{\,|\,X=1,Y=0} \oplus S_{\,|\,X=Y=1}.\end{aligned}$$

Now we establish the inequalities simply by remarking that

$$\{f\} \subseteq \{g\} \subseteq \{h\} \Rightarrow \{g\oplus h\} \subseteq \{f\oplus h\} \text{ and } \{g\oplus f\} \subseteq \{h\oplus f\}.$$

Thus, since

$$\left\{S_{|X=0,Y=1} \oplus S_{|X=1,Y=0} \oplus S_{|X=Y=0}\right\}$$

$$\subseteq \left\{S_{|X=0,Y=1} + S_{|X=1,Y=0} + S_{|X=Y=0}\right\} \subseteq \left\{S_{|X=Y=1}\right\}$$

(the latter because of s-coherence), we get our first inequality by composition with $S_{|X=Y=1}$. Similarly, since

$$\left\{S_{|X=Y=0}\right\} \subseteq \left\{S_{|X=0,Y=1} \oplus S_{|X=1,Y=0} \oplus S_{|X=Y=1}\right\}$$

$$\subseteq \left\{S_{|X=0,Y=1} + S_{|X=1,Y=0} + S_{|X=Y=1}\right\}$$

(the former because of s-coherence), we get our second inequality by composition with $S_{|X=Y=0}$. □

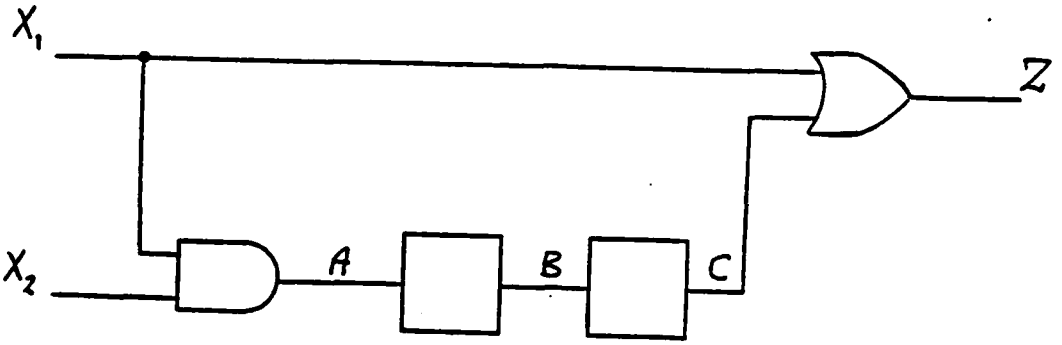


Figure 1. A simple sequential circuit.

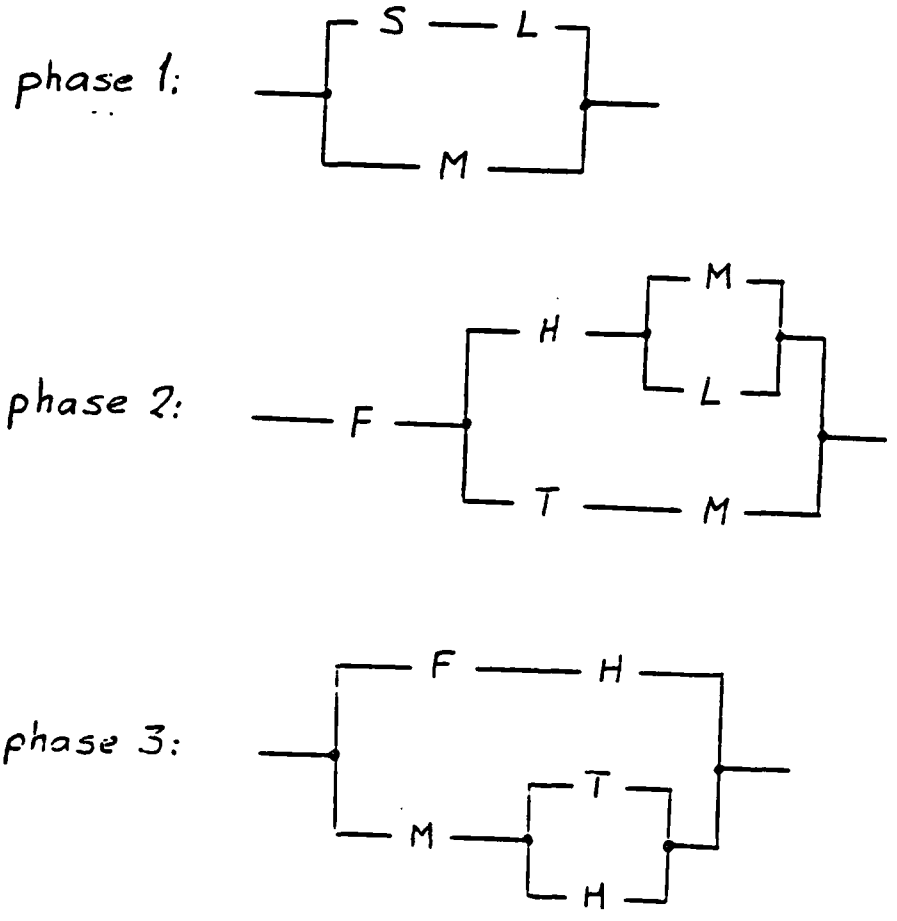


Figure 2. A phased firefighting mission (from Esary & Ziehms [7]).

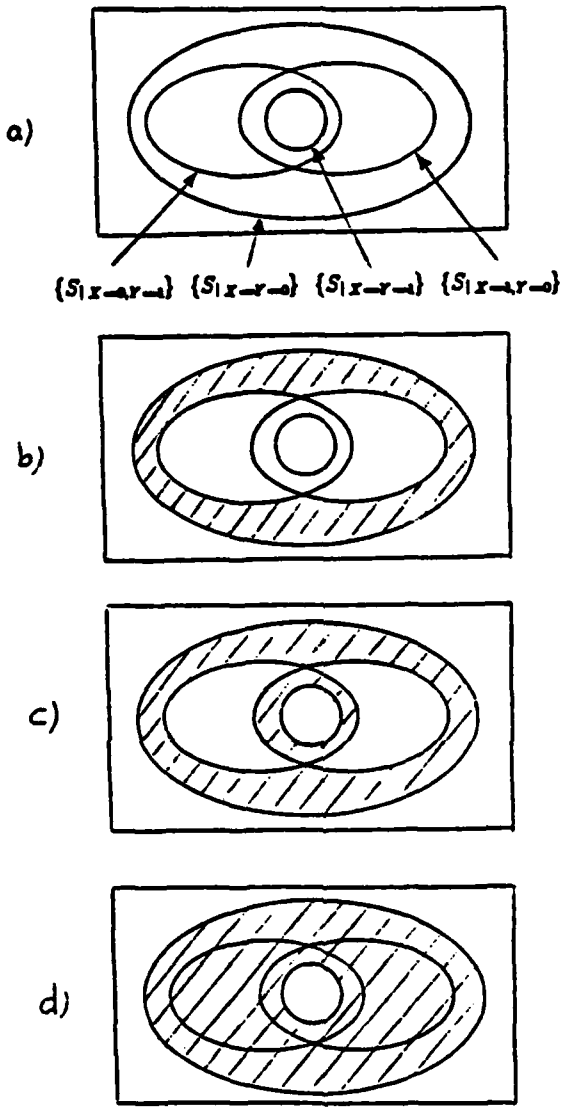


Figure 3. Venn diagrams for the minterms sets of the 3 Boolean differences

a. the 4 subfunctions

b. $\dfrac{dS}{d(X \cdot Y)} = \dfrac{dS}{d(X \oplus Y)}$

c. $\dfrac{d^2S}{dXdY}$

d. $\dfrac{dS}{d(X+Y)} = S_{|X=Y=0} \oplus S_{|X=Y=1}$

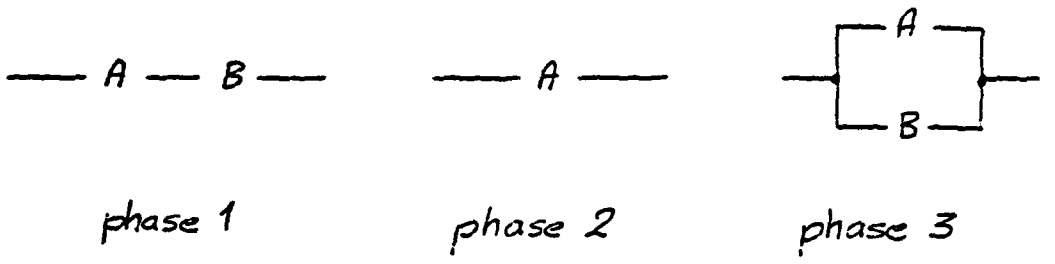


Figure 4. A very simple phased system.

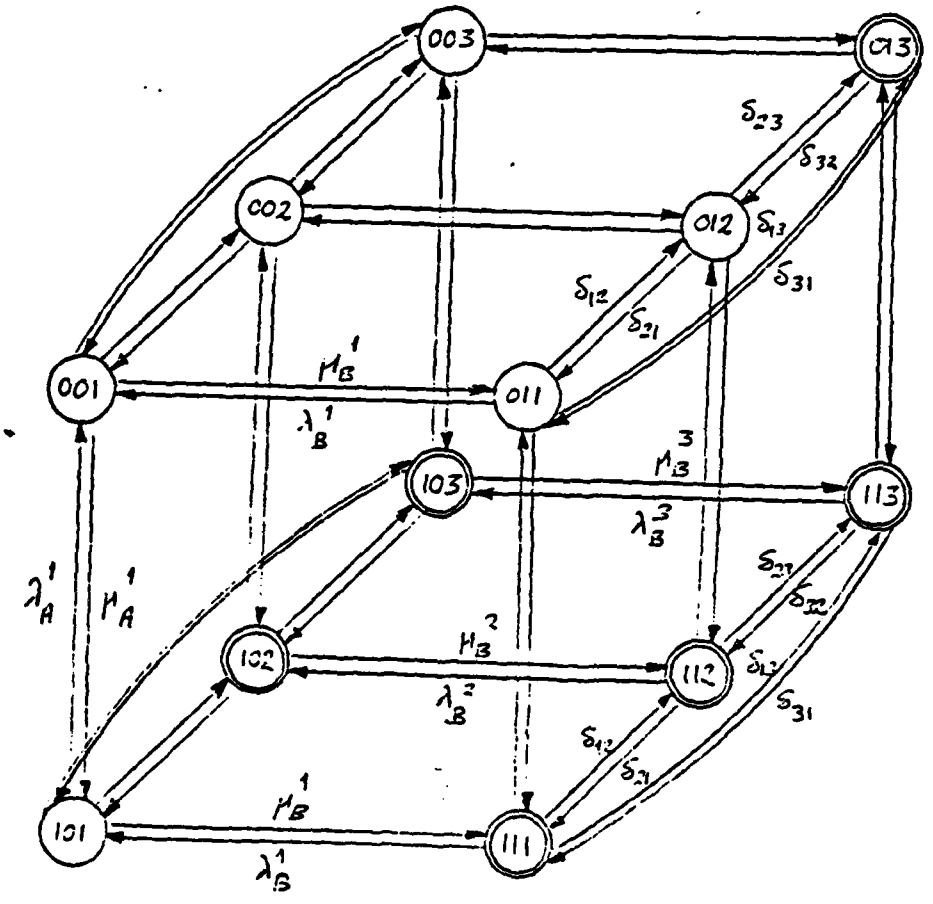


Figure 5. The state diagram for the system of Figure 4.

SECTION IV

TESTING COMPLEXITY

On minimizing a set of tests

Bernard M.E. Moret† and Henry D. Shapiro

Department of Computer Science
The University of New Mexico, Albuquerque, NM 87131

ABSTRACT

Minimizing the size or cost of a set of tests without losing any discrimination power is a common problem in fault testing and diagnosis, pattern recognition, and biological identification. This problem, known as the minimum test set problem, is known to be NP-hard, so that determining an optimal solution is not always computationally feasible. Accordingly, researchers have proposed a number of heuristics for building approximate solutions, without, however, providing an analysis of their performance. In this paper, we take an in-depth look at the main heuristics and at the optimal solution methods, both from a theoretical and an experimental standpoint. We conjecture that the heuristics will yield solutions that stay within of factor of two of the optimal cost and present generic examples where this factor is reached by any greedy heuristic. We then present the results of extensive experimentation with randomly generated problems. While the exponential explosion suggested by the problem's NP-hardness is apparent, our results suggest that real world testing problems of large sizes can be solved quickly at the expense of large storage requirements.

† The work of the first author was supported by the Office of Naval Research, Arlington, Virginia, under grant No. 0014-78-C-0311.

1. Introduction

Identification problems arise in almost all fields of scientific research. We are concerned here with a special type of deterministic identification, where an unknown (system state, animal species, location of a fault) must be classified in one of a given set of categories, based on the outcome of a set of tests. Each category is characterized by a vector of test outcomes, and an unknown object is classified in that category if its vector of test outcomes matches the category's characteristic vector. The collection of all categories together with their characteristic vectors is known as a **diagnostic table**. A diagnostic table with m categories and n tests can be represented as an $m \times n$ matrix, where the (i,j) entry is the result of test T_j applied to the unknown object O_i. Such formulation is common in testing and fault analysis [2,4,11], biology [15,22,23] and pattern recognition [5,13].

Given a diagnostic table, it is often the case that some tests are redundant. In such a case, it is of interest to find the smallest suitable subset in order to minimize the cost of identification. The **minimum test set** (also known as the test of minimum length) is the smallest subset of tests which discriminates between all categories distinguished by the full set of tests. Knowledge of the minimal test set can reduce costs in applications where a rapid identification is needed, that is, in situations where all the tests will be applied in parallel. Cost reduction will also occur in applications where the capital costs (procurement of the test equipment) far exceed the running costs, regardless of whether the actual testing is done in a parallel or sequential manner. (This is the minimization of the **acquisition cost** in decision trees [12].) Applications of the second type are to be found in most fields of human endeavor, including some that do not explicitly include testing: servicing equipment under poor access conditions (military equipment in the field, oil rigs at sea), where the cost of delivering service personnel and apparatus must be minimized; remote sensing missions, where the cost of the apparatus

must be minimized subject to performing all the required tasks; and fault diagnosis and design for testability, where, for instance, the number of checkpoints added to a circuit must be minimized subject to retaining a prescribed level of testability.

Unfortunately, the minimization problem is known to be NP-hard [6]. Accordingly, researchers have developed a number of heuristics for building suboptimal test collections by using variants of a greedy algorithm where, at each step, the locally optimal test is added to the partial solution. However, no analysis of those methods is offered in the literature.

In this paper, we take an in-depth look at existing heuristics and how they can be applied to develop optimal solutions. We conjecture that existing selection heuristics will not exceed the optimal by more than a factor of 2 and provide generic examples where this factor is asymptotically reached for all existing heuristics. We then present and discuss the results of extensive experiments with both artificial (randomly generated) and real-world problems. While the exponential explosion suggested by the problem's NP-hardness is quite apparent in the artificial examples, our results suggest that real-world problems of large sizes can be solved in reasonable time.

2. An Analysis of Proposed Heuristics

Almost all proposed heuristics belong to the class of **greedy** algorithms, in that they perform local, step-by-step optimization, using a suitable selection criterion. Very few analytical results are available about the minimum test set problem in general and the behavior of the proposed heuristics in particular. A number of Russian researchers [8,9,21] have studied the expected size of the minimum test set for randomly constructed tables; the analyses of the main two heuristics discussed in [12] in the context of identification trees do not extend to the minimum test set problem. In the following, we

briefly define the four main heuristics proposed in the literature and offer a partial analysis of their worst-case behavior.

2.1 Definitions

When a pair of categories is distinguished by only one test (that is, the categories' characteristic vectors differ in exactly one component), that test is called **essential** and must be included in any complete set of tests. Thus, in a step-by-step method, pre-including all essential tests is an optimal policy; all proposed methods [2,3,18,20] make use of this policy.

When all essential tests have been included, one can either attempt to extend the notion of essentiality or resort to a measure of a test's local optimality. The first approach has been used by researchers in microbiology [16,18,20]: since a test is essential when it is the only test to separate a pair of categories, a test is "nearly essential" if it is one of only two (or a few) tests to separate a pair of categories. This extension restricts the choice of the next test to one of those that separate that pair of categories which is separated by the least number of of tests — the least-separated pair. An algorithm using this criterion will thus focus on category pairs; ties between tests and/or between equally poorly separated pairs are broken by the use of a "second-level" heuristic — one of the measures of local optimality described below. We shall call this the **least-separated pair** criterion.

The second approach attempts to measure how well a new test will complement those already chosen; in such an approach, all as yet unincluded tests are considered for inclusion. An obvious choice is to count how many as yet unseparated pairs the new test will distinguish and choose a test which maximizes this count: we shall call this the **separation** criterion. This heuristic has been extensively used in fault analysis [3,4] and microbiology [16,20]. The contribution of a test can also be measured in terms of

entropy (or, equivalently, of information), in which case the initial state — a single homogeneous group — corresponds to an entropy of 0 and the final state — m distinct groups of one category each — to an entropy of $\log_2 m$; it can also be measured in terms of permutations, where the initial state corresponds to a value of 1 — for there is only one way to assign a label to the single initial set — and the final state corresponds to a value of $m!$, the number of ways in which m distinct items can be labelled. The information theory approach is found early in the literature and used extensively for both the minimum test set and the related minimum identification tree problems [4,12,16]. Formally, the entropy of collection C of k clusters, of sizes $s_1, \ldots, s_k$, comprising m elements in all, is defined as

$$H(C) = \log_2 m - \sum_{i=1}^{k} s_i \cdot \log_2 s_i .$$

Applying a test to a collection of clusters yields a new collection, with larger (or equal) entropy; the difference is the amount of information contributed by that test. The test which brings about the largest increase will be selected. We shall call this approach the **information** criterion. The combinatorial approach, described in [17] and used in [14], considers how many possible distinct partitions of the size used could exist; the logarithm of this quantity is used as a measure, called repartment [17]. However, the repartment of a partition of m objects is equal to m times the entropy of the partition (within an additive factor of $\log_2 m$), so that the two approaches are essentially equivalent.

Thus, we have four main heuristics: least separated pairs with separation used to choose among the candidate tests (at the "second level"); least separated pairs with information used at the second level; separation alone; and information alone. The first two restrict the choice of tests before applying a local measure of optimality, while the last two apply such a measure to all remaining tests.

2.2 Analysis

Examples are easily constructed that show that no heuristic is uniformly better than the other three. The four heuristics are rather similar: the effect of restricting the choice to those tests separating the least separated pair does affect the order in which the tests are selected — which is of no consequence with regard to the final subset selected; it also affects the composition of the final subset, since a different order or selection may modify the local measures of optimality of the remaining tests. Typically, we found that these indirect effects are minor. Moreover, the measures of local optimality are all convex functions, the minima and maxima of which all occur at the same points.

It is easily shown that, in the worst-case, none of these heuristics will yield a solution with a cost that is at most a constant away from the the optimal. Indeed, staying within a *fixed constant* around the optimal is itself an NP-hard problem (our proof uses a technique that has become standard in the field: see [6, pp. 138-139]). We show that this problem is NP-hard by showing that, if a heuristic existed that produced a solution that had at most k more tests than the optimal, then we could use it to construct the optimal solution (and thus solve an NP-hard problem). The idea is to scale our problem up, solve it with the heuristic, and then scale it down, so that the error margin will shrink to zero by rounding. Specifically, given a problem, we "multiply" it by $k+1$ by making $k+1$ copies of each object (regard each copy as a coordinate in a $k+1$-tuple) and thus $k+1$ copies of each test (one copy for each coordinate). Notice that the optimal solution for this problem has exactly $k+1$ times the number of tests of the optimal solution for the original problem. The heuristic solution will fall within k of the optimal for the larger problem; now pick as solution for the original problem the smallest of the test sets obtained by retaining from the heuristic solution only those tests that apply to the same coordinate. That set has no more than $1/(k+1)$ the number of tests of the heuristic solution; thus it has at most $k/(k+1)$ more tests than

the optimal solution; but all quantities must be integer, so that we have in fact obtained the optimal solution.

Furthermore, another standard technique can be used to show that staying within an *arbitrarily small ratio* of the optimal is also NP-hard (i.e., no fully polynomial time approximation scheme [6] exists for the problem): it is an immediate consequence of the corollary on page 141 of [6] and of the fact that our problem is *strongly* NP-hard. Thus the best possible algorithm is one that produces solutions, the size of which stays within a *fixed* ratio of the optimal.

We conjecture that all four heuristics discussed above exhibit the same worst-case behavior, yielding a test set that is, for binary tests, at most twice larger than the optimal — which, in view of the preceding proof, is about as good as can be expected. The main rationale behind our conjecture can be stated as follows. For the ratio to grow large, the optimal solution must remain small, thereby requiring tests with good discriminating power; however, those tests that the heuristic erroneously selects must be even better locally. Now, the discriminating power of a test is always measured on the whole partition, so that a test cannot be good locally without being fairly good overall. Therefore, the worst-case behavior should occur when the tests comprising the optimal solution are quite good for the most part, although marginally less good at each step than those selected by the heuristics. As to the heuristics, after selecting those locally good tests, they must complete their solution set with locally poor tests, so that they yield a large solution set.

Pushing this to the extreme, we present below a generic example where any of the proposed step-by-step heuristics will select what appear to be "perfect"† tests (in the

† i.e., even-splitting and such that each successive test divides exactly in two each of the clusters determined by the previous tests.

sense that they produce successive partitions where all subsets are of equal size), only to be forced to include tests which, although initially good, are poor at this point, each effecting only a few discriminations. All apparently perfect tests are in fact redundant in the final solution, because the heuristics had to complete their partial set with those tests which alone would comprise the optimal solution. Since the example yields an asymptotic ratio of 2, and in view of the above arguments, we conjecture that the worst-case performance ratio of any of the proposed heuristics never exceeds 2.

We first construct the example for heuristics based on local optimality, then show how to modify it by dropping a few tests to make it applicable to least separated pairs heuristics as well. The example has 2^n objects and 2^n+2n-1 tests (where $n>3$), in three groups. The collection comprises all 2^n **simple** tests, $\{s_1, \ldots, s_{2^n}\}$, where test s_i asks "does this object belong to category i?". Then there are $n-1$ "perfect" tests, $\{p_1, \ldots, p_{n-1}\}$, which by themselves determine a partition of 2^{n-1} subsets of 2 objects each; they are most easily constructed by filling the table with, for each object, the most significant $(n-1)$ bits of its index in the table. Finally, there are n "almost perfect" tests, $\{t_1, \ldots, t_n\}$, each of which splits the categories $2^{n-1}+1$ *vs.* $2^{n-1}-1$; we construct them by filling the table, for each object, with its binary Gray code modified only by repeating the first code (all 0s) and omitting the code consisting of all 1s — this guarantees the appropriate split by tilting the balance of 1s and 0s for each of those tests. Following is the diagnostic table for $n=4$.

Now, *any* heuristic based on local optimality will first select all p tests, since they produce a perfectly balanced partition at each step. Indeed, any k-step heuristic (that is, one which selects tests k at a time, for some fixed k, rather than one at a time) will also select the p tests, whenever k divides $n-1$. After that, the heuristic will select all of the t_i tests, which are always preferable to the simple tests, and complete the test set by picking either s_1 or s_2, to yield a solution of $2n$ tests. The optimal test set,

objects	tests		
	p	t	s
0	000	0000	1000000000000000
1	000	0000	0100000000000000
2	001	0001	0010000000000000
3	001	0011	0001000000000000
4	010	0010	0000100000000000
5	010	0110	0000010000000000
6	011	0111	0000001000000000
7	011	0101	0000000100000000
8	100	0100	0000000010000000
9	100	1100	0000000001000000
10	101	1101	0000000000100000
11	101	1110	0000000000010000
12	110	1010	0000000000001000
13	110	1011	0000000000000100
14	111	1001	0000000000000010
15	111	1000	0000000000000001

however, comprises all t_i tests and either the s_1 or the s_2 simple test, for a total of $n+1$ tests. Thus the ratio is $2n/(n+1)$ for any heuristic that uses step-by-step optimization with a local measure of discriminating power.

To make this example work for the least separated pair heuristics with a second-level selection criterion of the type described above, we need to modify it so that one of the least separated pairs always includes the next "perfect" test, thereby ensuring that test's selection. In the example as built, the least separated pairs are separated by two tests. Therefore, for each p test, we shall remove selected s tests to give least separated "status" to a pair separated by that p test. In doing that, we cannot remove both s_{2i-1} and s_{2i}, since then the $(2i-1,2i)$ pair would be separated only by the remaining t test, making it essential; nor can we remove either s_1 or s_2, since removing one makes the other essential. Hence we pick suitable pairs separated by four tests (two s, one t, and one p), and remove the two s tests in order to guarantee the selection of the p test.

Specifically, we remove the following two s tests for each p test. For p_1, we remove tests $s_{2^n-2^{n-1}+1}$ and $s_{2^n-2^{n-1}+2^{n-2}+1}$; for p_{n-1}, we remove tests s_{2^n-3} and s_{2^n}; finally, for p_i, $2 \leq i < n-1$, we remove tests $s_{2^n-2^{n-i}+1}$ and $s_{2^n-2^{n-i}+2^{n-i-1}+2}$. It is easily verified that no two of those removed tests are of the form $2i-1,2i$ and that the pairs chosen are separated only by the appropriate p test. The end result is a problem on which *any* of the proposed step-by-step heuristics — including those that select more than one test at a time — will produce a set of $2n$ tests as opposed to the optimal's $n+1$.†

A different approach yields another example for which the separation-based heuristics will exhibit an asymptotic ratio of 2. The idea is to transform known worst-case examples for the related **set covering** problem [1,6,10] into minimum test set problems. Recall that a set covering problem is given by a family of sets, the goal being to find the smallest number of sets in that family that cover the family, i.e., such that their union is equal to the union of all sets in the family. Let m be the number of elements in that union. The transformation creates a pair of categories for each of the m distinct set elements; each set in the family gives rise to a test, which takes values of 1 for the first of the pair of categories for each element in that set and values of 0 everywhere else. The problem is completed by adding $\lceil \log_2 m \rceil$ tests to distinguish the m pairs of categories. After selecting all these tests, the separation heuristics will have to select those tests that correspond to the sets selected in the set covering problem by the standard greedy method. Johnson [7] has shown that the latter selection can be arbitrarily far from optimal, by a factor of $\log_2 m$; specifically, he provides a generic example with $m = 3 \cdot 2^k$ where the optimal cover uses 3 sets and the greedy solution requires $k+1$.

† One could apply a backwards elimination procedure on the result of the forward selection procedure (see [5]) in order to eliminate some of the redundant tests. However, the elimination of redundant tests is itself a minimum test set problem and thus not amenable to optimal solution. Although a greedy-based backwards elimination procedure would indeed improve the greedy solution in this example, cases remain where the ratio of 2 would still be reached, as our next example shows.

This translates in our problem to an optimal solution of $\lceil \log_2 m \rceil + 3$ and a greedy solution of $\lceil \log_2 m \rceil + k + 1$. Since $k = \log_2 m - \log_2 3$, our worst-case ratio is

$$\frac{\lceil \log_2 m \rceil + \log_2 m - (\log_2 3 - 1)}{\lceil \log_2 m \rceil + 3}$$

which is always smaller than 2 and reaches 2 asymptotically. (In this example, the greedy solution has no redundant tests, so that it cannot be improved through the application of a backwards elimination procedure.) This example can also be transformed to make it work for the information-based heuristics.

3. Bounding Methods

Non-exhaustive search algorithms that find the optimal solution — such as branch-and-bound or depth-first search — require both upper and lower bounds on the size of the optimal solution. The bounds are used in pruning, i.e., in eliminating fruitless directions of search (pruning occurs whenever the local lower bound reaches or exceeds the global upper bound); they can also be used in guiding the selection. A global upper bound is trivially provided by the size of the best solution found so far; our conjecture of the previous section, if true, would imply that this bound is fairly tight. (Indeed, a proof of our conjecture would also provide a lower bound: the optimal solution must be at least half as large as the easily computed greedy solution.)

Any measure of a test's discriminating power that gauges distances on the way from the initial to the final partition can also be used to derive lower bounds. At any step, we compute the distance from the partition determined by our partial solution to the final partition, as well as the local contribution of each available — i.e., not yet chosen nor eliminated — test. We then sort the available tests' contributions in decreasing order and, assuming no interaction between tests, find by repeated summing how many tests are needed to complete the partial solution. Since the contribution of a

test does not increase as the partial solution is expanded, this gives us a safe lower bound.

The separation-derived function gives us a lower bound on the number of additional tests needed to distinguish the remaining unseparated pairs, assuming that no two tests separate the same pair — a very unlikely event. The information-derived function finds a lower bound on the number of additional tests needed to increase the entropy to the final partition's value, assuming no cross-information† between tests — an unlikely assumption, but one that can be closely approximated. We note that, whenever there is cross-information between two tests, there will be some pairs that they both distinguish, but that the converse is false. For instance, if m is a power of 2 and $\log_2 m$ of the tests perfectly complement each other, then those tests have no cross-information, yet any two of them distinguish $m^2/8$ common pairs. Thus we expect the separation bound to be much looser than the information bound.

A simple example will illustrate our point. Consider a problem with m (an even number) categories, where all tests effect an even split, $m/2$ *vs.* $m/2$. The tightest possible lower bound on the size of the optimal solution is $\lceil \log_2 m \rceil$, an achievable size. Now, each test separates $(m/2)\cdot(m/2) = m^2/4$ pairs and brings an increase in entropy of 1 bit, so that the separation-derived bound is

$$\left\lceil \frac{m\cdot(m-1)/2}{m^2/4} \right\rceil = \left\lceil \frac{2\cdot(m-1)}{m} \right\rceil = 2,$$

while the information-derived bound is

$$\left\lceil \frac{\log_2 m}{1} \right\rceil = \lceil \log_2 m \rceil .$$

† There is cross-information between two tests if the amount of information which they contribute together is less than the sum of the amounts which they contribute individually.

The information-derived bound is as tight as possible; the separation-derived bound is off by an unbounded factor.

The better the tests are, the poorer the separation-derived lower bounds become; in fact, the case illustrated above is essentially the average case in random tables with large numbers of tests, since well-splitting tests are far more likely than others. Thus the separation-derived bounds will be practically useless in problems that have a large number of tests relative to their number of categories (a "wide" diagnostic table). Even for "square" or "tall" diagnostic tables, those bounds will be effective only if the tests are rather poor.

In fact, the information-derived bound can also be arbitrarily smaller than the size of the optimal solution, although the factor cannot grow as large as for the separation-derived bound. Clearly, the worst possible behavior for the information-derived bound is to indicate the need for a logarithmic number of tests (close to the theoretical minimum) while the problem in fact requires a linear number of tests (close to the theoretical maximum). Such a behavior can be observed in a square diagnostic table where the 1 entries are disposed so as to make the table into a lower triangular matrix. Thus, for m object categories, the information-derived bound can be off by at most a factor of $O(m/\log_2 m)$, while the separation-derived bound can be off by a factor of $O(m)$.

4. Experimental Results and Discussion

4.1 Goals and methodology

The goals of experimentation were three-fold: to verify our deductions about the selection criteria and bounding functions; to determine how much work was expended on finding the optimal solution (as opposed to verifying its optimality); and to obtain an

estimate of the size of the largest problems that could be solved in a reasonable amount of time by these techniques.

Faced with a problem of subset search, the algorithm designer usually has a choice of four techniques: dynamic programming, cutting plane techniques, branch-and-bound, and depth-first search. The minimum test set problem has no apparent formulation in the framework of dynamic programming. Integer programming, through the solution of its linear programming subproblem and the use of cutting planes, has proved very effective with the related (also NP-hard) problem of set covering [1]. Unfortunately, the linear programming formulation of the minimum test set problem requires a number of equations that grows as a quadratic function of the problem's size (as opposed to a linear function for the set covering problem), thereby producing an excessively large system. The last two methods are more attractive for our purposes since they both perform an explicit search of the state space as guided by selection criteria and bounding functions. An estimate of the amount of storage needed for the intermediate solutions in a straightforward implementation of branch-and-bound techniques shows that memory requirements are too large to allow the solution of problems of useful size.

Therefore we chose the depth-first search technique (also known as single branch enumeration). It has the advantage of requiring only the storage of a single path in the search space (whereas branch-and-bound techniques may require an exponential number). Also, since the first solution produced by the depth-first search algorithm is the greedy solution, the chosen algorithm has the added advantage of allowing immediate comparisons between selection criteria. Finally, the two methods based on the least-separated pair should work very well in most cases, as the restriction on the choice of candidate tests should drastically diminish the size of the search space. We wrote four PASCAL programs, each implementing one of the four selection heuristics with its

matching bounding function. The global upper bound is provided by the size of the best solution found so far; our conjecture implies that this bound is fairly tight. The lower bounds can be derived from the local contributions of each remaining test and the distance to the solution: we assume that the tests do not interact and find by repeated summing of the tests' contributions (sorted in decreasing order) how many tests are needed to complete the partial solution.

All four programs pre-include essential tests whenever such are to be found. It is noted that, although only those tests that are essential with the initial partition must appear in any solution, the backtracking process may give rise to "locally essential" tests, since the process of removing a test from consideration in the subtree may make other tests essential in that region of the state space. As a result, an efficient implementation requires a pair-oriented data structure, keeping track of which available tests distinguish each unseparated pair, so that the search for essential tests can proceed in nearly constant time. This data structure can grow very large and its memory requirements turned out to be the main limiting factor in real world examples.

All four programs were run on randomly generated problems and those two programs based on the separation measure were also run on real world examples excerpted from the microbiology literature (in which test sets are regularly published). Nearly all real world examples included variable outcomes (i.e., undefined test values) and a few had multiple-valued (as opposed to binary) tests. All artificial examples had binary tests only and the random generator was set so that the two possible outcomes would be exactly balanced over the whole table. (Such problems are harder than those where one outcome is favored, because the even balance introduces a bias in favor of well-splitting tests. We also ran a number of experiments with various skews; all proved noticeably easier to solve than their evenly balanced counterparts.) When the number of tests was small for the number of objects, the randomly generated table often did not distinguish

between all objects; in such a case, a solution is a subset of tests that effects as much discrimination as the full set of tests. The data collected included the size of the greedy solution and of the optimal solution, the initial lower bounds, the number of backtracks needed to reach the solution, and the total number of backtracks used.

4.2 Artificial examples

In order to study the influence of the number of categories and that of the number of tests on the behavior of the algorithms, four series of experiments were run. In two of the series, the number of categories was kept constant while the number of tests was varied; in one series, the process was reversed; and in the last series, all problems were square with increasing sizes. The sizes varied from 6 to 64, with varying resolution. Twenty-five examples were generated for each size in each series and their results averaged; in all, nearly 2500 examples were run.

The results are presented in graphical form in Figures 1-6. Each of the first four figures displays the data collected in one series of experiments in the form of four graphs: the top two show the average total number of backtracks required (the most accurate measure of work) as well as the average size of the solution, while the bottom two show the percentage of work that was devoted to finding the optimal solution (as opposed to verifying it). The two graphs on the left display the data obtained with the least separated heuristics and those on the right are concerned with the other two heuristics. In all graphs, data points marked with a triangle correspond to results obtained with the information-derived bounding and selection functions; those marked with a square correspond to results obtained with the separation-derived bounding and selection functions; and those marked with a circle indicate the average size of the solution. Figure 5 illustrates the performance of the greedy methods (it displays the average and largest values of the ratio of the size of the greedy solution to that of the optimal

solution) while Figure 6 illustrates the performance of the bounding methods (it shows the average and largest values of the ratio of the initial lower bound to the size of the optimal solution). Finally, curves were passed through the points in order to make the graphs more legible. (Those curves should not be taken as an accurate depiction of the heuristics' behavior: the data are intrinsically discrete.)

In the first series, all problems had 16 categories, while the number of tests varied from 8 to 64 in steps of 2. Since 16 is a power of 2, it is a transition point for the size of the best possible test set: although problems of that size could admit a solution of 4 tests, such a solution would be hard to find, since it must be composed of 4 perfect tests with no cross-information. For such a solution to exist, a rather large choice of tests must be provided; therefore we expect the average size of a solution to stay above 5 until the number of tests is large, then to decrease slowly. Since a solution of 5 tests is optimal in most cases, we expect that a good bounding function will stop the search algorithm shortly after the solution is found. Moreover, until a solution of 4 tests is feasible, there will be a number of optimal solutions of size 5, so that one such solution will be found almost immediately by all heuristics. Figure 1 shows the experimental results, which confirm our expectations. While the information-based bounding did very well, the separation-based one did very poorly — because many of the tests are well-splitting. Since the information bounds were tight, the reduced branching factor associated with the least separated pair heuristics did not play a significant role, while that role is clearly exemplified by the two programs using the separation bounds.

In the second series, all problems had 22 categories; other parameters were as in the first series. With 22 categories, the best possible test set has size 5. Such a solution is not as difficult to realize as in the first series, since it is well above $\log_2 22 \cong 4.46$. On the other hand, the bounding can be decisive only if a solution of 5 tests is reached; the test interactions that make a 6-test set optimal will not be reflected strongly enough in

the bounds. As a result, we expect the programs to perform a nearly exhaustive search of the first few levels of the search tree when the optimal solution has 6 tests. Of course, such solutions abound, so that all heuristics will find one almost instantly, while solutions of 5 tests will be considerably more difficult to find until the choice of tests becomes sufficiently large. As that choice grows, all four heuristics will find a solution of 5 tests very early; the information-bounded programs will then stop shortly, while the separation-bounded ones will go on and explore nearly the full tree. The role of the reduced branching factor of the least separated pair heuristics will be as in the first series, minor for the information-bounded programs and major for the other two. Experimental results are shown in Figure 2.

In the third series, the number of tests was kept at 16, while the number of categories was varied from 8 to 64 in steps of 2. As the number of categories increases, so does the size of the theoretically minimal solution; the size of the best realizable solution increases even faster, since the choice of tests becomes relatively small. With a large number of objects, the tests will interact significantly, so that we expect bounds to be rather loose and play only a minor role. On the other hand, the probability that a pair is separated by only one or two tests significantly increases, so that we expect the least separated pair heuristics to perform considerably better than the other two. Finally, the selection heuristics will do rather well because the tests, with so many entries in each column, are well differentiated. For small to medium numbers of categories, the situation is more complex. The selection heuristics will perform rather poorly for a number of categories just above the number of tests, because the optimal solution is likely to be nearly unique. At the same time, the number of categories is not large enough that good bounding can take place; thus the overall work should increase dramatically. However, the bounding and selection will improve as the number of categories and of optimal solutions increases, so that the work done will level out. As

the number of categories further increases, it becomes more difficult again to find the optimal solution and we expect the total amount of work to increase once more. The results are shown in Figure 3. Notice how closely the curves for the information-bounded programs follow those for the separation-bounded ones, demonstrating the relative lack of success of the bounding functions.

The fourth series had square problems, with a size increasing from 6 to 60. (Only the best of the four heuristics was used for the larger sizes; indeed, 40x40 appeared to be the practical limit for the separation-bounded heuristics.) Since, in a square problem, the choice of tests is relatively restricted, we can expect a behavior similar to that exhibited in the first half of Figure 3. Least separated pair heuristics will hold a slight edge over the other two and the information-guided heuristics will vary in performance between much better than the separation-guided — when the solutions become harder to find and thus provide for better bounding — and almost as poor — when the solutions become easier to find and thus cause much looser bounds. Experimental results are shown in Figure 4. They dramatically illustrate the trade-off between tight bounding and ease in finding solutions: the harder a solution is to find, the easier it will be to prune the remaining branches, yet, if the solution is too hard to find, most of the tree will be explored just looking for it.

The data collected about the size of the greedy solutions confirmed that all four heuristics are good selection criteria. No greedy solution ever exceeded the optimal by more than 50%; on the average, greedy solutions were only 6% to 7% larger than optimal. As expected from our discussion of the separation and information measures, the two performed equally well (the average ratios were always within one percent of each other, which is not a statistically significant difference over 25 experiments); the two methods relying on the least-separated pair heuristic showed a slight advantage, presumably due to the narrower focus they impart on selection. Figure 5 presents the

results (the average ratios of the size of the greedy solution to the size of the optimal solution) in the form of four graphs (one for each series of experiments); in each graph, data points marked with a cross (×) correspond to the least-separated pair heuristics while those marked with a plus (+) correspond to the first-level heuristics.

We chose to illustrate the behavior of the bounding methods by collecting statistics on the ratio of the size of the optimal solution to the initial lower bound (as derived by using separation or information measures). The average and worst-case values of this ratio are plotted in Figure 6 in four graphs (one for each series of experiments); in each graph, data points marked with a triangle correspond to information-based bounds, while those marked with a square correspond to separation-based bounds. As expected from our discussion, the lower bounds based on information are consistently better that those base on separation. In particular, while the separation-derived bounds worsen with increasing number of objects, no such trend is apparent for the information-derive bounds.

Overall, the experimental results confirmed our evaluation of the selection criteria and the bounding functions. All four selection criteria appear equal. Least separated heuristics are vastly superior to the other two when efficient bounding is not possible (as when the number of categories grows large with respect to the number of tests), due to their small branching factor. Information-bounded heuristics are much better than the other two when the optimal solution is found early and efficient bounding can be done (as when the number of tests grows large with respect to the number of categories). In all cases, the most efficient program used the least separated pair heuristic with information-based bounding. The largest solvable problems had sizes of around 40 by 40, although a single parameter could be increased well beyond that.

4.3 Real world examples

In view of the results obtained with artificial examples, a certain optimism is justified as regards real world problems. Such problems tend to have many essential tests; moreover, they are often composed of a small number of well-splitting tests and a large number of rather poor (possibly simple) tests. With such a structure, selection criteria should perform well, as several microbiology researchers have found [16,19]. Moreover, many pairs will be be separated by only a few tests, so that the least separated pair heuristics should keep the branching factor quite low.

We used the separation criterion only, as the information criterion is not easily adapted to problems with variable test outcomes. (Being based on clusters, it requires that the size of all clusters established by the inclusion of tests be recorded. In turn, this requires that all clusters be kept track of explicitly, since common sets and subsets must be eliminated. All of this adds up to excessive bookkeeping and enormous storage requirements.) The table below presents a synopsis of the results on eight examples from the microbiology literature; in that table, LSP stands for the least separated pair heuristic with separation bounding while SEP stands for separation as the "first-level" heuristic. The data presented include the size of the problem, the size of the optimal solution and that of the greedy solutions found by each heuristic, the number of essential tests, the total number of backtracks used by each heuristic in obtaining the optimal solution, and the percentage of work used to discover (as opposed to verify) the optimal solution. Several remarks are in order. First, many of the tests incorporated in an optimal solution were essential, showing how important it is for an algorithm to include essential tests whenever possible. Secondly, the optimal solution was almost always found immediately, confirming the power of the greedy heuristics in real world examples. Third, some of the problems run were four times larger than the largest artificial examples attempted, yet ran almost a hundred times faster. Finally, the

Isolates	Problem Size Categories	Tests	Solution Opt.	LSP	SEP	Number Ess.Tests	Backtracks LSP	SEP	% Work to Sol. LSP	SEP
Actinomadura[1]	11	32	5	5	7	1	7	20	0.0	67.2
Cyanobacteria[2]	106	19	16	16	16	15	16	16	0.0	0.0
Enterobacteria[3]	7	20	7	7	7	4	7	7	0.0	0.0
Pseudomonas [14]	27	21	8	9	8	2	16	21	21.7	0.0
SMA12 kit[3]	142	12	12	12	12	12	12	10	0.0	0.0
Streptococci[3]	36	32	25	25	25	25	25	25	0.0	0.0
Streptococci [18]	50	122	36	36	36	31	43	43	0.0	0.0
Yeasts[4]	98	56	16	16	17	5	144	1556	0.0	0.2

[1] Goodfellow, M., *et al. Numerical taxonomy of Actinomadura and related Actinomycetes.* J. Gen. Microbiol. **112** (1979). pp 95-111.

[2] Rippka, R., *et al. Generic assignments, strain histories and properties of pure cultures of cyanobacteria.* J. Gen. Microbiol. **111** (1979). pp. 1-61.

[3] Rypka, E.W. Private communication. Lovelace Medical Center, Albuquerque, 1981.

[4] Belin, J.M. *Identification of yeasts and yeast-like fungi I: taxonomy and characteristics of new species described since 1979.* Can. J. Microbiol. **27** (1981)). pp. 1235-1251.

program using the least separated pair heuristic with separation bounding never used more than a minute of CPU time running on a VAX11/780 computer. (For comparison, the greedy heuristic used in [14] on the Pseudomonas example took 2.8 minutes on an IBM360/50, while our program took 1.9 seconds to guarantee an optimal solution for the same problem — an enormous difference that we attribute mostly to our careful choice of data structures.)

Overall our results confirmed our optimism about real-world problems and justify an even more positive attitude: with a judicious trade-off between time and space, it will be possible to solve even larger examples without major modifications. If some better bounding method can be developed — and preliminary research indicates that this is within reach, using linear programming with merged constraints, then the time traded off will easily be regained. In fact, this indicates that a hybrid algorithm, partaking of both depth-first search and branch-and-bound techniques, may be best.

5. Conclusion

We have reviewed the methods proposed in the literature for dealing with the minimum test set problem. We have evaluated the proposed selection heuristics and conjectured that their worst-case behavior never produces solutions larger than twice the optimal, providing two examples where this ratio is asymptotically reached. We have presented the results of extensive experimentation with four backtracking algorithms. Our results confirm that existing selection heuristics are quite satisfactory; they also indicate that the best backtracking method involves a heuristic which uses the information criterion for selecting tests and deriving bounds and relies on the least separated pair heuristic to keep branching factors low. Experimentation with real world problems showed the importance of pre-inclusion of essential tests; it also gave grounds for optimism since, despite the known NP-hardness of the general problem, an inferior version of our programs solved large problems in a very short time.

Much work remains to be done. Better bounding methods must be sought, which apply even when variable outcomes are present. An extension of the information criterion is the obvious first step. Beyond that, the use of the linear programming subproblem for deriving bounds (as used for the set covering problem in [10]) appears promising; although the size of the linear programming problem grows faster than that of the original problem, one can diminish it by merging some conditions, thereby relaxing the constraints (indeed, merging all conditions into one gives us the separation criterion). In addition, the integer programming approach with cutting plane methods is worth investigating, despite the size of its formulation. Most importantly, ways of incorporating measured amounts of redundancy into the solution must be sought: the minimum test set is, by definition, a fragile entity. While redundancy can easily be incorporated through simple methods (such as insisting that each pair be separated by at least two tests, whenever possible), the more complete risk model of pattern

recognition [5] provides a suitable framework for more sophisticated methods.

6. Acknowledgements

The authors wish to thank the referees for many helpful comments; in particular, we are indebted to the second referee for suggesting our second worst-case example for the greedy heuristics.

References

[1] E. Balas and A. Ho. *Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study.* Mathematical Programming **12** (1980). pp. 37-60.

[2] L.W. Bearnson and C.C. Carroll. *On the design of minimum length fault tests for combinational circuits.* IEEE Trans. Comp. **TC-20**, 11 (1971). pp. 1353-1356.

[3] H.Y. Chang. *An algorithm for selecting an optimum set of diagnostic tests.* IEEE Trans. Electr. Comp. **EC-14**, 5 (1965). pp. 706-711.

[4] H.Y. Chang, E. Manning, and G. Metze. *Fault Diagnosis of Digital Systems.* John Wiley & Sons, New York, 1970.

[5] P.A. Devijver and J. Kittler. *Pattern Recognition: A Statistical Approach.* Prentice-Hall International, Englewood Cliffs, N.J., 1982.

[6] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, San Francisco, 1979.

[7] D.S. Johnson. *Approximation algorithms for combinatorial problems.* J. Comp. & Syst. Sc. **9** (1974). pp. 256-278.

[8] A.D. Korshunov. *The length of minimum tests for rectangular tables I.* Cybernetics **6** (1970). pp. 723-733.

[9] A.D. Korshunov. *The length of minimum tests for rectangular tables II.* Cybernetics **7** (1971). pp. 1-14.

[10] C.E. Lemke, H.M. Salkin, and K. Spielberg. *Set covering by single-branch enumeration with linear programming subproblems.* Oper. Res. **19** (1971). pp. 998-1022.

[11] P.N. Marinos. *Derivation of minimal complete sets of test-input sequences using Boolean differences.* IEEE Trans. Comp. **TC-20**, 1 (1971). pp. 25-32.

[12] B.M.E. Moret. *Decision trees and diagrams.* Comp. Surveys **14**, 4 (1982). pp. 593-623.

[13] P.M. Narendra and K. Fukunaga. *A branch-and-bound algorithm for feature subset selection.* IEEE Trans. Comp. **TC-26**, 9 (1977). pp. 917-922.

[14] S.I. Niemela, J.W. Hopkins, and C. Quadling. *Selecting an economical binary test battery for a set of microbial cultures.* Can. J. Microbiol. **14** (1968). pp. 271-279.

[15] R.W. Payne and D.A. Preece. *Identification keys and diagnostic tables: a review.* J. Royal Statist. Soc. A **143**, 3 (1980). pp. 253-292.

[16] R.W. Payne. *Selection criteria for the construction of efficient diagnostic keys.* J. Statist. Planning & Inf. **5** (1981). pp. 27-36.

[17] A. Rescigno and G. A. Maccacaro. *The information content of biological classifications.* In *Information Theory: Fourth London Symposium,* C. Cherry, Ed., Butterworths, London (1961). pp. 437-445.

[18] E.W. Rypka, W.E. Clapper, I.G. Brown, and R. Babb. *A model for the identification of bacteria.* J. Gen. Microbiol. **46** (1967). pp. 407-424.

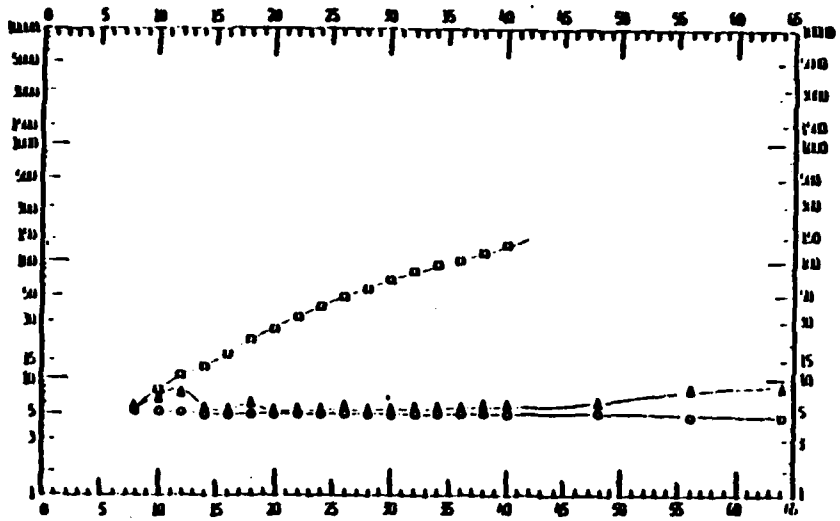
[19] E.W. Rypka. *Truth table classification and identification.* Space Life Sciences **3** (1971). pp. 135-156.

[20] E.W. Rypka, L. Volkman, and E. Kinter. *Construction and use of an optimized identification scheme.* Laboratory Medicine **9**, 11 (1978). pp. 32-41.

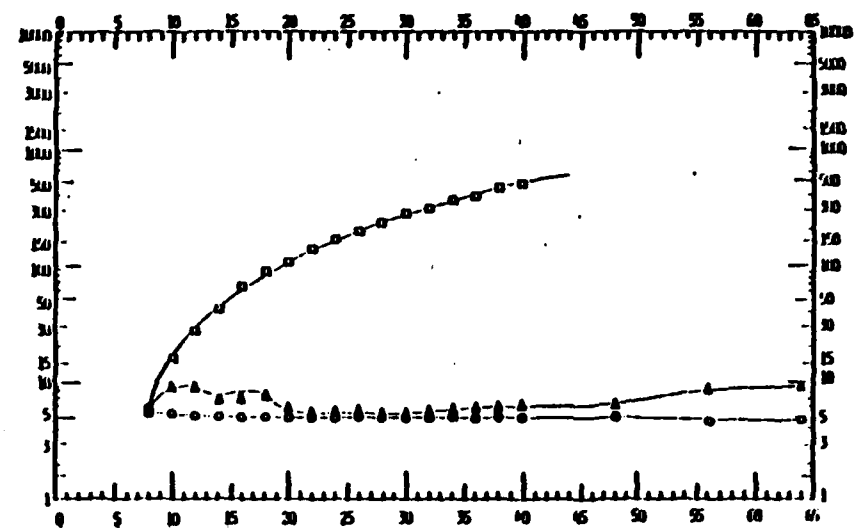
[21] V.A. Slepyan. *Minimal test length for a certain class of tables.* Discretnyi Analiz **23** (1973). pp. 59-71 (in Russian).

[22] W.R. Willcox and S.P. Lapage. *Automatic construction of diagnostic tables.* Computer J. **15** (1972). pp. 263-267.

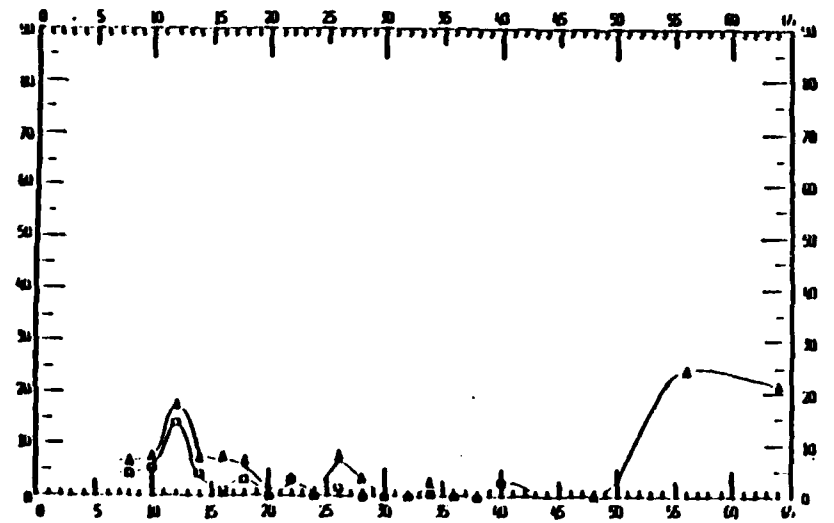
[23] W.R. Willcox, S.P. Lapage, and B. Holmes. *A review of numerical methods in bacterial identification.* Antonie van Leeuwenhoek **46**, 3 (1980). pp. 233-299.



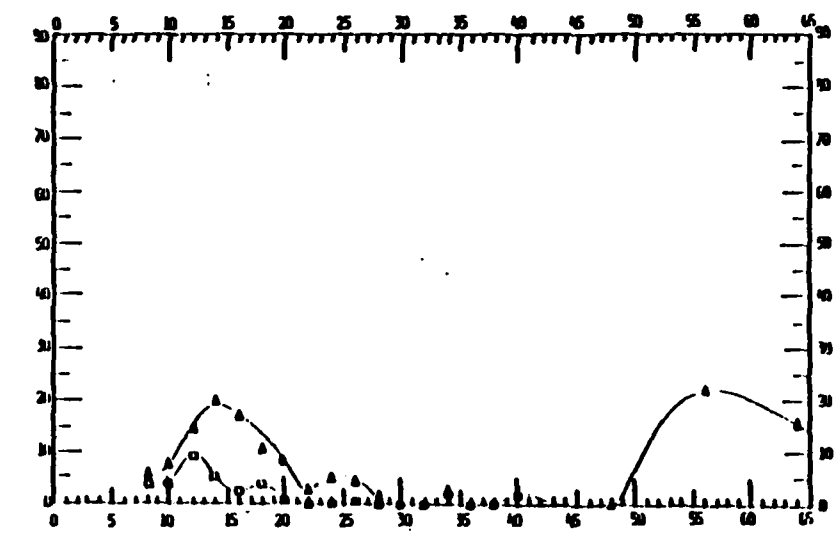
a) Least separated pair heuristics: total work



b) First level heuristics: total work

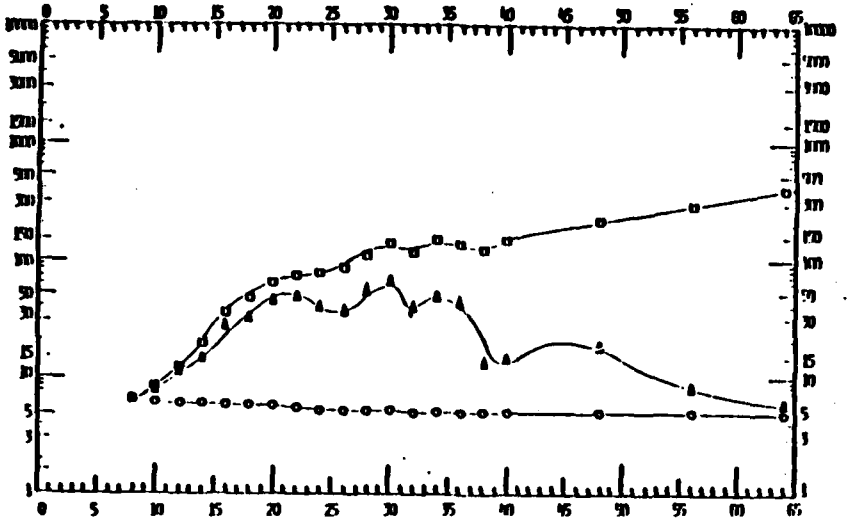


c) Least separated pair heuristics: percent work to solution

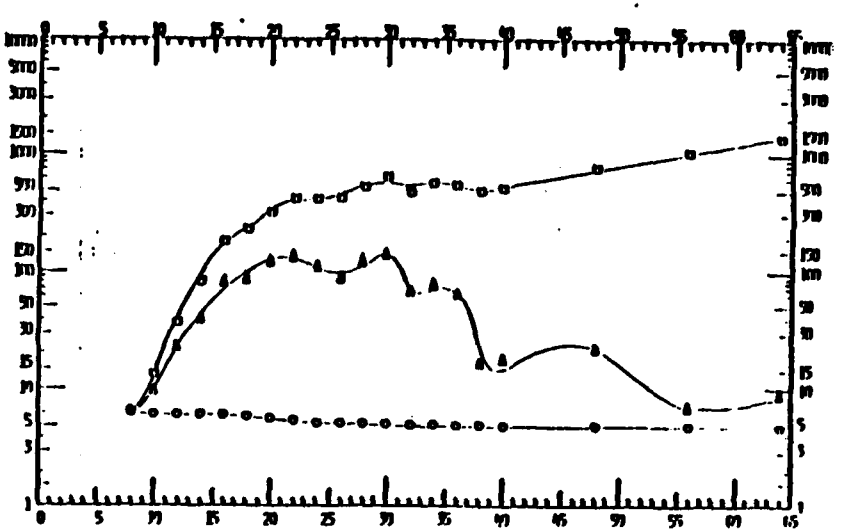


d) First level heurfstics: percent work to solution

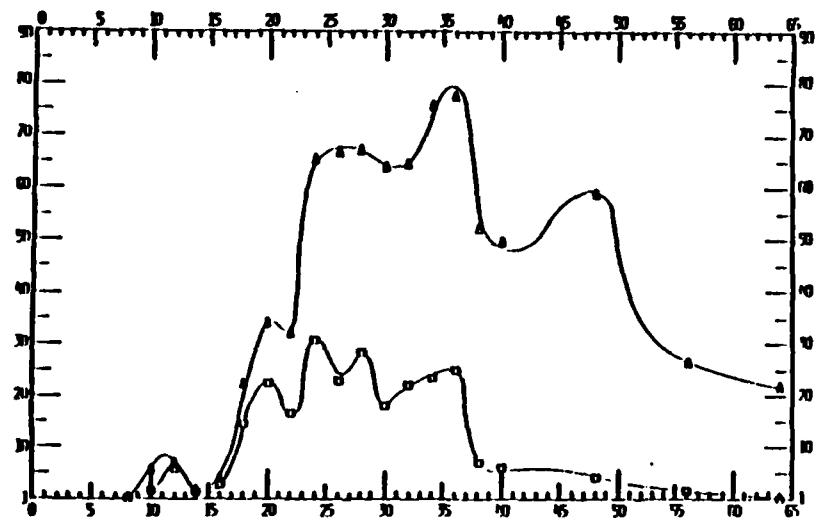
Figure 1. Results for the problems with 16 objects.



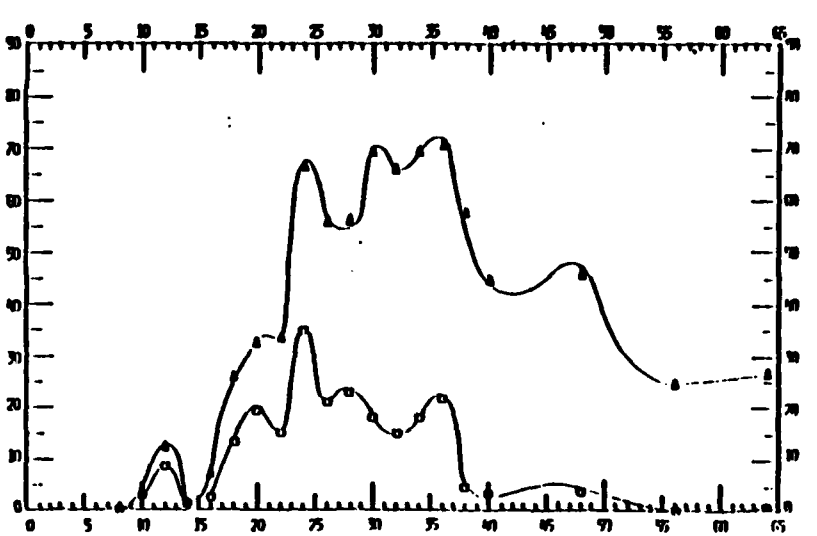
a) Least separated pair heuristics: total work



b) First level heuristics: total work

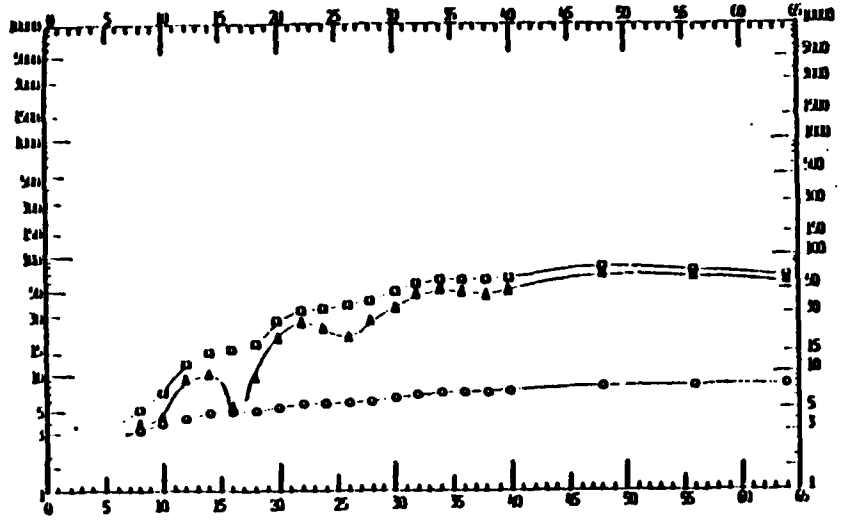


c) Least separated pair heuristics: percent work to *so*lution

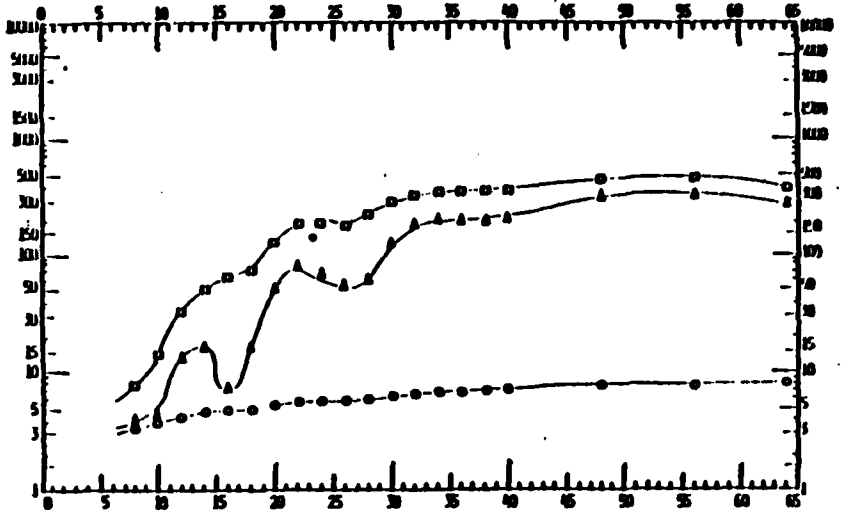


d) First level heuristics: percent work to solution

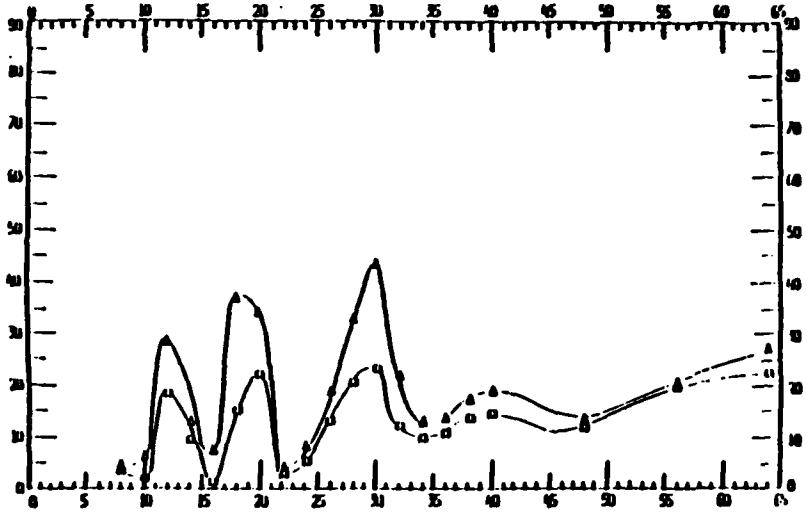
Figure 2. Results for the problems with 22 objects.



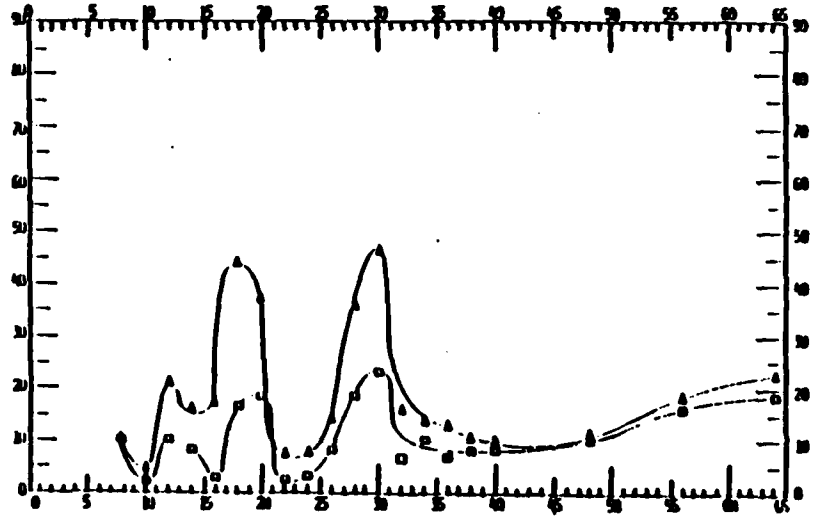
a) Least separated pair heuristics: total work



b) First level heuristics: total work

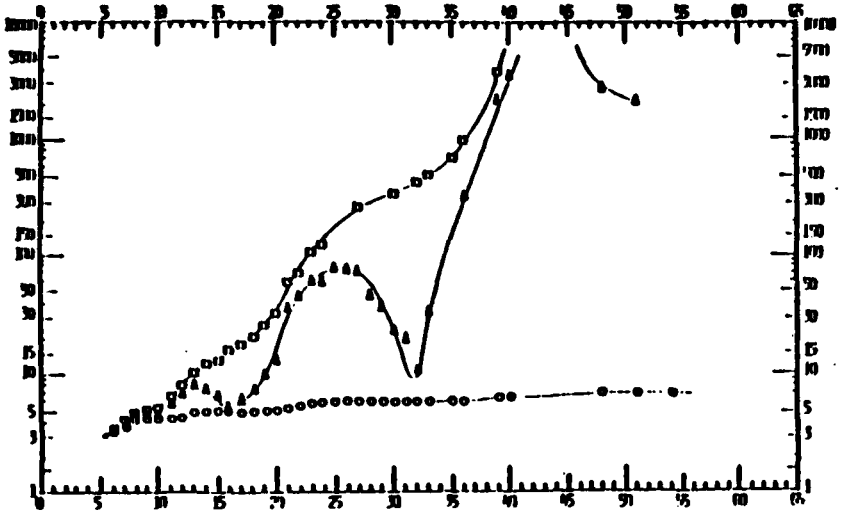


c) Least separated pair heuristics: percent work to solution

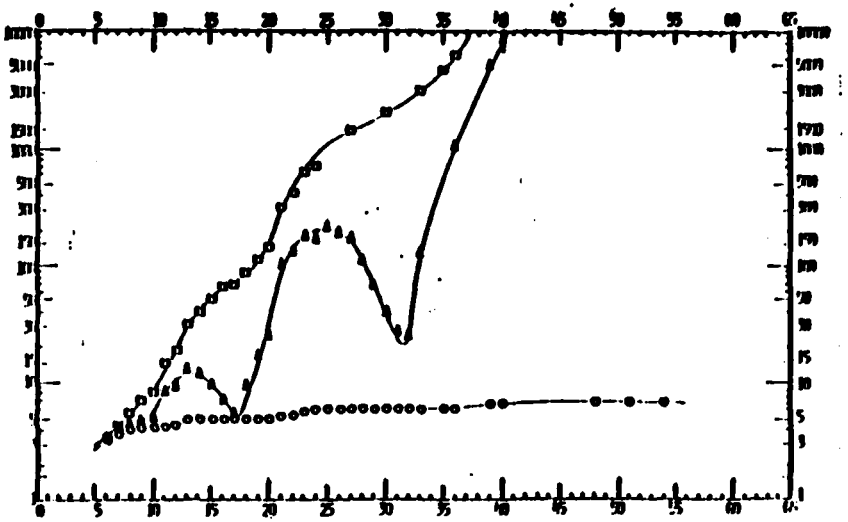


d) First level heuristics: percent work to solution

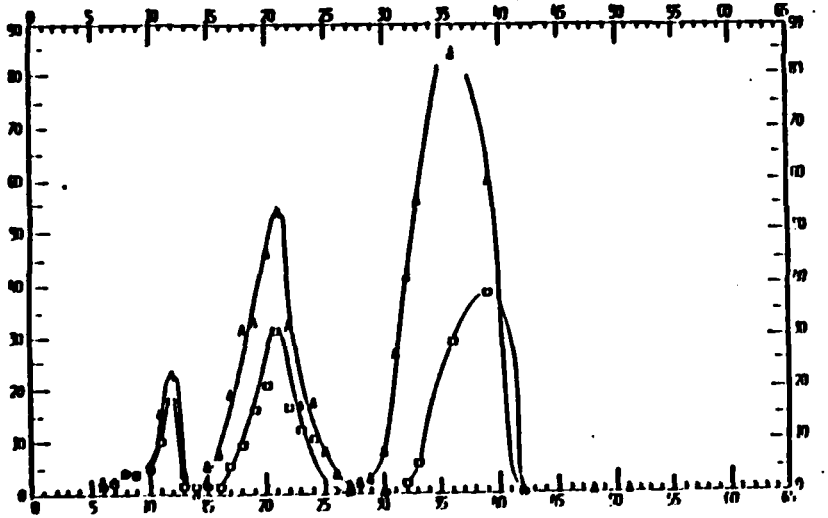
Figure 3. Results for the problems with 16 tests.



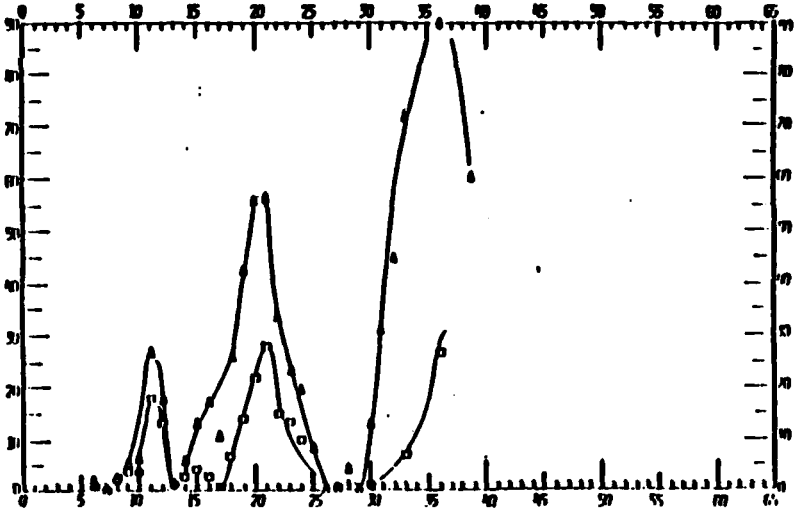
a) Least separated pair heuristics: total work



b) First level heuristics: total work



c) Least separated pair heuristics: percent work to solution



d) First level heuristics: percent work to solution

Figure 4. Results for the square problems.

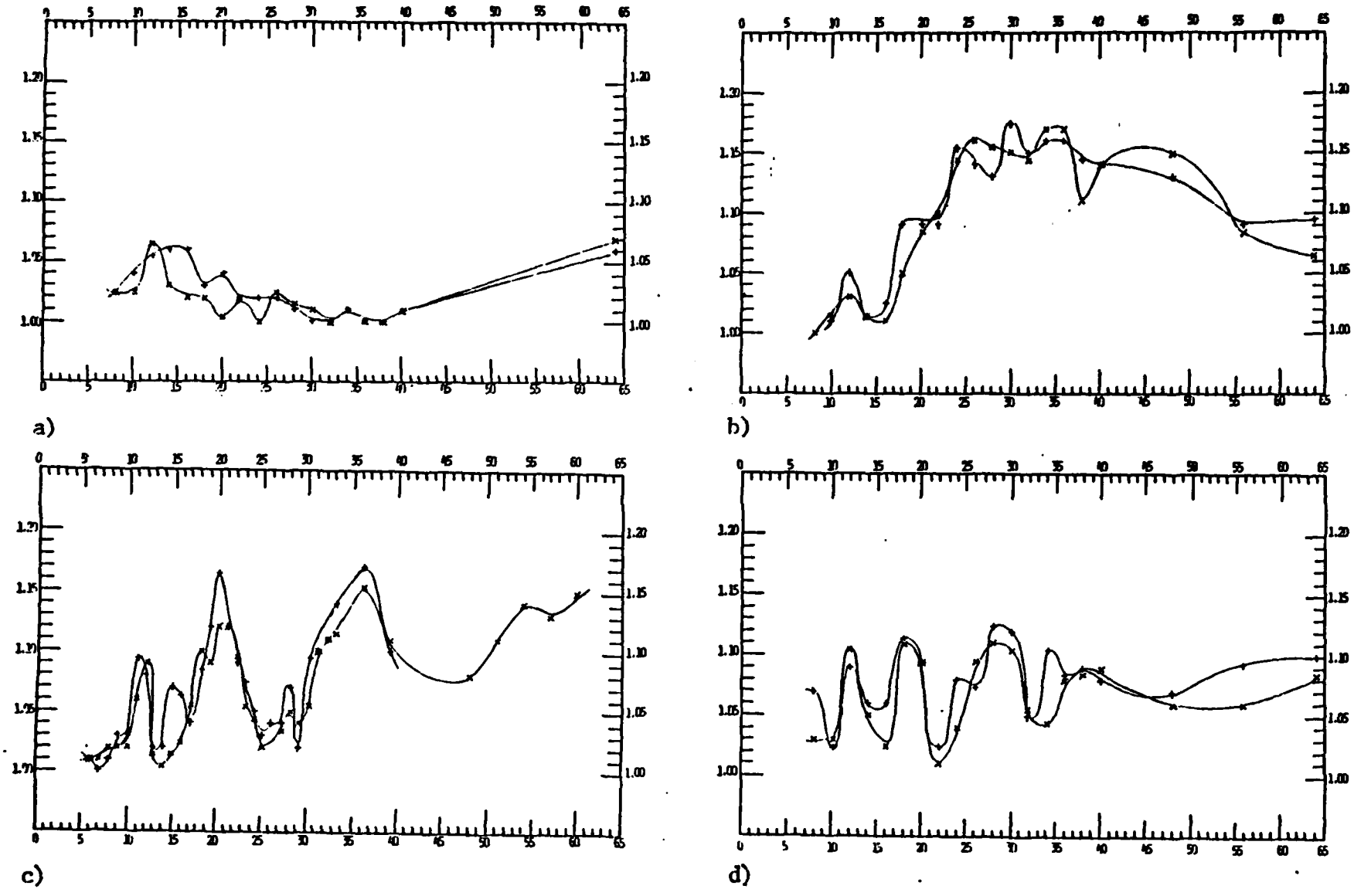


Figure 5. The ratio of the size of the greedy solution to that of the optimal solution.

a) 16 objects, varying number of tests
b) 22 objects, varying number of tests
c) square problems
d) 16 tests, varying number of objects

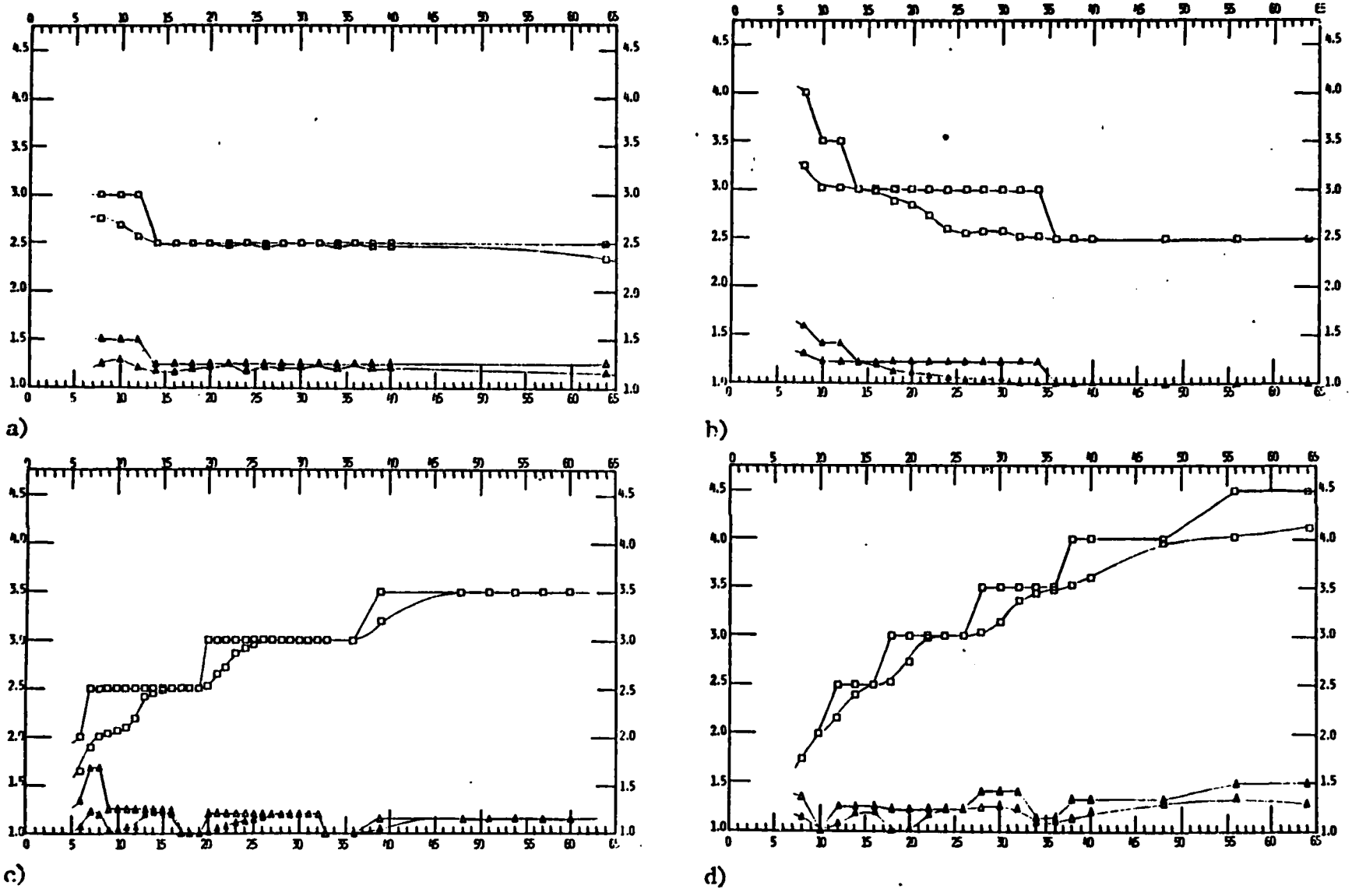


Figure 6. The ratio of the size of the optimal solution to the initial lower bound.

a) 16 objects, varying number of tests
b) 22 objects, varying number of tests
c) square problems
d) 16 tests, varying number of objects

SECTION V

OPTIMAL SOLUTION OF LINEAR INEQUALITIES

IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. PAMI-3, NO. 6, NOVEMBER 1981 643

Optimal Solution of Linear Inequalities with Applications to Pattern Recognition

D. C. CLARK AND R. C. GONZALEZ, SENIOR MEMBER, IEEE

Abstract—An algorithm for the optimal solution of consistent and inconsistent linear inequalities is presented, where the optimality criterion is the maximization of the number of constraints satisfied. In the terminology of pattern recognition, the algorithm finds a linear decision function which minimizes the number of patterns misclassified. The algorithm is developed as a nonenumerative search procedure based on several new results established in this paper. Bounds on the search are also developed and the method is experimentally evaluated and shown to be computationally superior to other techniques for finding minimum-error solutions.

Index Terms—Algorithm, decision function, discriminant function, inequalities, linear, minimum error, optimal, pattern recognition.

I. Introduction

FORMAL APPROACHES to pattern recognition may be divided into two principal categories: syntactic and decision-theoretic [1], [2]. The syntactic approach is based on the use of automata and language theory to process patterns that have been expressed in terms of structural primitives. The decision-theoretic approach, on the other hand, deals with techniques for obtaining decision functions capable of partitioning sets of pattern vectors whose components are real, numerical measurements or features.

Central to the decision-theoretic approach are methods for finding decision functions that are optimal in some sense. In statistical formulations, the approaches due to Fisher [3] and Bayes [1] are the most commonly used in pattern recognition. Fisher's classic paper establishes a procedure for finding a linear discriminant function with the maximum ratio of interclass to intraclass scatter. Bayes' decision rule yields the minimum expected loss and, in the Gaussian case, reduces to a quadratic function determined by the mean vectors and covariance matrices of the classes under consideration.

A criterion of optimality that has been receiving increased attention in recent years is based on finding decision functions which minimize the number of errors between two pattern classes. Unlike the formulations mentioned above, the approaches that have been proposed in this area are procedures which employ the training patterns directly to find minimum-error decision functions. The most noteworthy efforts in this area are the algorithms proposed by Ibaraki and Muroga [4], Warmack and Gonzalez [5], Miyake and Shinmura [6], [7], and Miyake [8]. Other schemes which "tend" to minimize the number of errors have been proposed by Mengert [9] and Smith [10] (see also the comment by Grinold [11]). Finally, we mention the stochastic optimization techniques by Wassel [17] and by Do-Tu and Installe [18] for minimizing the error rate.

A two-class linear decision problem may be expressed as a system of linear inequalities [1], [5]. In this paper, we develop an algorithm for finding an optimal solution of consistent (corresponding to separable pattern classes) and inconsistent (corresponding to inseparable classes) linear inequalities, where the optimality criterion is the maximization of the number of constraints satisfied by the solution. This criterion is directly analogous to minimizing the number of misclassified patterns. The algorithm is developed as a nonenumerative search procedure based on several new results established in this paper. Bounds on the search are also developed and the procedure is experimentally evaluated and shown to be computationally superior to other published techniques for finding minimum-error solutions.

II. Foundation

Consider the system of homogeneous linear inequalities

$$Aw \geq 0 \tag{2.1}$$

where A is an $m \times (n+1)$ matrix with $m \geq (n+1)$, and w is an $(n+1)$-vector in R^{n+1}. It will be assumed throughout the following discussions that A satisfies the Haar condition [11]; that is, every $(n+1) \times (n+1)$ submatrix of A is of rank $(n+1)$.

In the terminology of pattern recognition, w is a *weight vector*, each row of A corresponds to an *augmented pattern vector* so that $a_{i,n+1} = \pm 1$, and (2.1) is the statement of a two-class, m-pattern problem in which the augmented patterns of one class have been multiplied by -1. The Haar condition implies that the patterns are in general position [1].

Two pattern classes are said to be *linearly separable* if there exists a w with the property that $Aw > 0$. (As indicated below a w that satisfies (2.1) can be displaced so that it also satisfies the strict inequalities $Aw > 0$). There exist a number of well-known algorithms for finding a solution weight vector when the classes are separable [1]. In the inseparable case, we are interested in finding a weight vector that is optimal in the

Manuscript received April 3, 1980; revised February 4, 1981. This work was supported in part by the Office of Naval Research under Contract N00014-78-C-0311.

D. C. Clark was with the Department of Computer Science, University of Tennessee, Knoxville, TN 37916. He is now with the Pattern Analysis and Recognition Corporation, Los Angeles, CA 90045.

R. C. Gonzalez is with the Department of Electrical Engineering, University of Tennessee, Knoxville, TN 37916.

0162-8828/81/1100-0643$00.75

sense that it satisfies the largest possible number of row inequalities in (2.1), and thus minimizes the number of patterns that are misclassified.

Each row vector a_i of A determines a hyperplane in R^{n+1}:

$$H_i = \{w \in R^{n+1} \mid a_i \cdot w = 0\} \tag{2.2}$$

for $i = 1, 2, \cdots, m$, and

$$a_i \cdot w = \sum_{j=1}^{n+1} a_{ij} w_j. \tag{2.3}$$

Each hyperplane H_i is an n-dimensional subspace of R^{n+1} passing through the origin, and (2.2) also indicates that H_i is the n-dimensional *orthogonal complement* of a_i.

Since H_i bifurcates R^{n+1}, there is a quartet of open and closed half-spaces corresponding to each hyperplane. They are denoted by

$$\begin{aligned} S_{io} &= \{w \in R^{n+1} \mid a_i \cdot w > 0\} \\ \bar{S}_{io} &= \{w \in R^{n+1} \mid a_i \cdot w < 0\} \\ S_{ic} &= \{w \in R^{n+1} \mid a_i \cdot w \geq 0\} \\ \bar{S}_{ic} &= \{w \in R^{n+1} \mid a_i \cdot w \leq 0\}. \end{aligned} \tag{2.4}$$

It is easily demonstrated that each of these half-spaces is convex, and that the intersection of an arbitrary collection of convex sets is itself convex. It is also noted that H_i is the *boundary* (or *frontier*) of each of the half-spaces defined in (2.4).

A *convex polyhedral set* is defined as the intersection of a finite number of closed half-spaces. Furthermore, because they are closed under addition and nonnegative scalar multiplication, the partitions of R^{n+1} generated by the closed half-spaces in (2.4) also satisfy the definition of convex cones [13]-[15]. Therefore, the partitioning of R^{n+1} by $\{H_i \mid i = 1, 2, \cdots, m\}$ establishes a finite set of convex *polyhedral cones*. Each of these cones, generated by a finite number of *supporting hyperplanes*, contains the origin, is nonempty, closed, and unbounded along its principal axis. The boundary of a cone, formed by sections of its supporting hyperplanes, is called the *frontier* of the cone. The intersection of n hyperplanes define an *edge* on the frontier of a cone.

The concepts introduced in the above discussion are illustrated in Fig. 1. It is noted that a vector w contained in the interior of a cone would yield strict inequalities, while a vector contained in an edge would yield a zero inner product with all the hyperplanes that define that edge. It is shown in [5] that displacing an edge vector into the interior of a cone without changing the sense of the strict inequalities is a simple matter when A satisfies the Haar condition. It is also illustrated in Fig. 1 that every cone C in R^{n+1} has an *image*, denoted by C^-, about the origin. If the number of inequalities satisfied by a vector w in C is less than or equal to k_m, where

$$k_m = \begin{cases} m/2 & \text{for } m \text{ even} \\ (m+1)/2 & \text{for } m \text{ odd,} \end{cases} \tag{2.5}$$

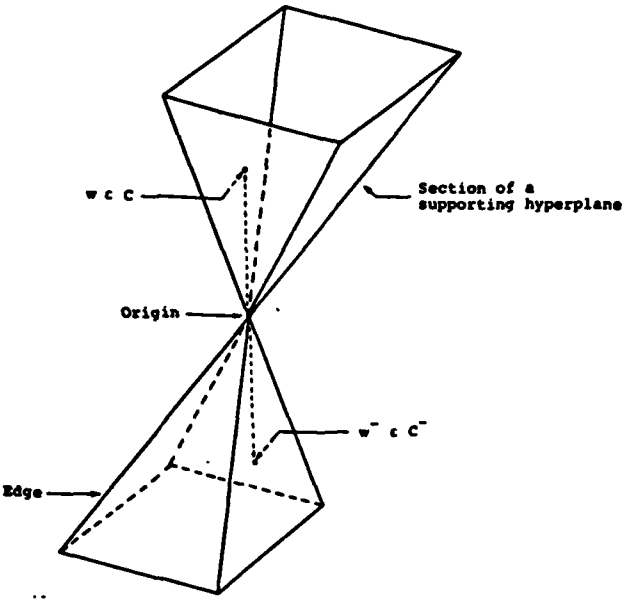


Fig. 1. Illustration of a three-dimensional convex polyhedral cone, its image, frontier, and edges.

then the number of inequalities satisfied by the image of w (i.e., $w^- = -w$) is greater than or equal to k_m.

III. Development of the Algorithm

In the following discussion we make extensive use of the *index set* $I(w)$ of a vector w in R^{n+1}, which is defined as the set of integer values between 1 and m such that i is in $I(w)$ if $a_i \cdot w < 0$, where a_i is the ith row vector of A. The *error* of w, denoted by $\text{err}(w)$, is defined as the cardinality of $I(w)$; in other words, the number of values of i for which $a_i \cdot w < 0$. In order to simplify the notation, and since we are interested only in optimal solutions, it will be assumed throughout that a vector w will be replaced by $-w$ if $\text{err}(-w) < \text{err}(w)$. We begin the development with the following lemmas.

A. Two Basic Lemmas

Lemma 1: Let w be a nonzero vector in R^{n+1}. Then either w is an optimal solution of (2.1) or one of the hyperplanes H_i, $i \in I(w)$, contains an optimal solution of (2.1).

Proof: Let z by any optimal solution of (2.1) and let L be a straight line segment extending from w to z. We will show that either w is an optimal solution of (2.1) or $L \cap H_i$ is an optimal solution for some $i \in I(w)$, which is sufficient to prove the lemma.

If $w = cz$ for some $c < 0$ then $-w$ is an optimal solution and we replace $-w$ by w. If $w \neq cz$ for $c < 0$, then L does not pass through the origin. In this case the set of optimal solutions lying on L consists of a series of one or more subsegments of L, on each of which the same number of inequalities of (2.1) is satisfied by each vector on that subsegment. Consider the subsegment containing z. One endpoint of this subsegment is z. Let the other endpoint be denoted by v, which is not 0 since L does not pass through the origin and is optimal because it is on the subsegment containing z. If $w = v$, then w is an optimal

solution. Since v is an optimal solution and it is also the endpoint of an optimal subsegment it follows that, if $w \neq v$, then for some i, $v \in L \cap H_i$ and $a_i \cdot w < 0$; that is, w is on the other side of the hyperplane defining the end of the optimal subsegment. Since $a_i \cdot w < 0$, then we have $i \in I(w)$. This concludes the proof. □

In the following discussion it will be useful to consider the notion of a minimum-error solution relative to a subspace S of R^{n+1}. By this we mean a nonzero vector v contained in S and with the property $\text{err}(v) \leq \text{err}(w)$ for all nonzero $w \in S$. It is noted that a minimum-error solution of (2.1) is a minimum-error solution relative to R^{n+1}.

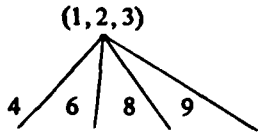
Lemma 2: Let w be a nonzero vector in S, a subspace of R^{n+1}. Then either w is an optimal solution of (2.1) relative to S, or at least one of the subspaces $S \cap H_i, i \in I(w)$, contains an optimal solution of (2.1) relative to S.

Proof: The proof is analogous to that of Lemma 1. We let z be any optimal solution relative to S and L the straight line segment extending from z to w. Then L is contained in S and the proof proceeds as before, but with the words "optimal solution" replaced by "optimal solution relative to S." □

Given a nonzero vector w in R^{n+1} and a hyperplane $H_i, i \in I(w)$, if z is an optimal solution of (2.1) that lies in H_i, then z is also optimal relative to H_i. Hence, Lemma 1 implies that if a set of vectors B contains w and at least one relative optimal vector for each hyperplane $H_i, i \in I(w)$, then B contains at least an optimal vector relative to R^{n+1}. Thus, the search for an optimal vector can be reduced to a search for relative optimal vectors for each of the subspaces $H_i, i \in I(w)$. This search for relative optimal vectors will be guided by the concepts established in Lemma 2.

B. Search Trees

The search for an optimal vector may be expressed in terms of a tree diagram. In order to illustrate this, assume that n + 1 = 4 and that we begin the search with a vector $w(1)$ lying in the intersection of hyperplanes H_1, H_2, H_3; that is, $w(1) \in H_1 \cap H_2 \cap H_3$. If, upon performing the products $a_i \cdot w(1)$, $i = 1, 2, \cdots, m$, we find that $w(1)$ lies on the negative side of hyperplanes H_4, H_6, H_8, and H_9, then $I[w(1)] = \{4, 6, 8, 9\}$. This information is summarized in the following tree diagram.



where (1, 2, 3) specifies the hyperplanes determining the starting edge and each labeled branch represents a subspace (hyperplane) to be searched for a relative optimal solution. Once relative optimal solutions are found for H_4, H_6, H_8, and H_9, Lemma 2 guarantees that either $w(1)$, or at least one of these relative optimal solutions, is an optimal solution of (2.1).

In order to search H_4 we apply Lemma 2, which requires that we find a vector lying on the subspace $S = H_4$. This can easily be achieved by exchanging H_1 for H_4 so that the vector will lie in $H_4 \cap H_2 \cap H_3$. Let us denote this vector by $w(2)$

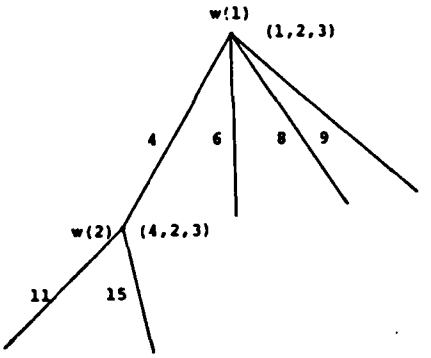


Fig. 2. A simple search tree after computation of two edge vectors.

and assume, for example, that $I[w(2)] = \{11, 15\}$. Then, Lemma 2 indicates that the search for an optimal solution relative to H_4 may be reduced to that of searching the subspaces $H_4 \cap H_{11}$ and $H_4 \cap H_{15}$. The status of the search is summarized in Fig. 2. It is noted that the dimensionality of the subspaces to be searched has been reduced by one; in other words, we started searching H_4 (an n-dimensional subspace) and the problem now is one of searching the subspaces $H_4 \cap H_{11}$ and $H_4 \cap H_{15}$ which are $(n - 1)$-dimensional.

In order to find an edge on $H_4 \cap H_{11}$, we can replace H_2 to obtain $w(3) \in H_4 \cap H_{11} \cap H_3$. Suppose that $I[w(3)] = \{6, 12\}$. According to Lemma 2, the search for an optimal solution relative to $H_4 \cap H_{11}$ is reduced to that of searching the subspaces $H_4 \cap H_{11} \cap H_6$ and $H_4 \cap H_{11} \cap H_{12}$. In our example, these are one-dimensional subspaces since $n + 1 = 4$. Thus, the problem at this point is simply that of finding a vector in $H_4 \cap H_{11} \cap H_6$ and a vector in $H_4 \cap H_{11} \cap H_{12}$. Let us denote these vectors by v_1 and v_2.

In order to continue the search, we find $w(4) \in H_4 \cap H_{15} \cap H_3$. Suppose that $I[w(4)] = \{5, 7, 10\}$. Following an argument identical to the one just given for $w(3)$, we would find the problem reduced to that of finding three vectors lying in $H_4 \cap H_{15} \cap H_5$, $H_4 \cap H_{15} \cap H_7$, and $H_4 \cap H_{15} \cap H_{10}$, respectively. Let us denote these vectors by v_3, v_4, and v_6. The initial problem of finding an optimal solution relative to H_4 has now been reduced to that of selecting from among the vectors v_1 through v_5 the one with the lowest error. The search up to this point is summarized in Fig. 3. It is noted that this completes the examination of subspace H_4 and that no other vectors contained in this subspace can yield a lower error than the best vector in the set v_1 through v_5. Thus, at this point in the search, an optimal solution relative to H_4 has been found.

In order to continue the search, we would next consider another hyperplane from the set $\{H_6, H_8, H_9\}$ and repeat the procedure discussed above for obtaining a relative optimal solution. When all these hyperplanes have been considered, either $w(1)$ or at least one of the optimal solutions relative to H_4, H_6, H_8, or H_9, would be an optimal solution to (2.1). It is noted that the number of levels traversed in the tree in order to examine each hyperplane for a relative optimal solution is $n + 1$. In the following discussion we formally define the concept of a search tree and prove (in Theorem 1) that a search

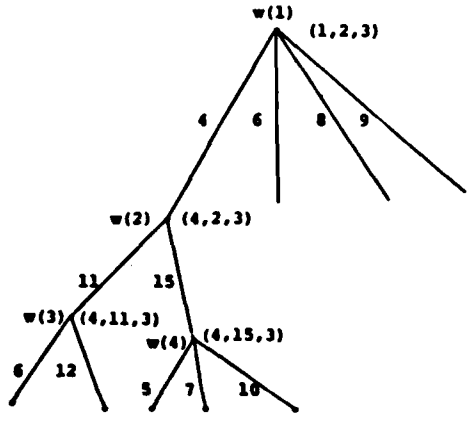


Fig. 3. Search tree after subspace H_4 has been examined.

tree leads to an optimal solution of (2.1). Procedures for reducing the size of a search tree are discussed in Sections III-C and III-D.

Let $w(1), w(2), \cdots, w(p)$ be nonzero vectors in R^{n+1}, and let $I_i = I[w(i)]$, $i = 1, 2, \cdots, p$. Assume that for $k = 2, 3, \cdots, p$, we have

$$w(k) \in H_{l_1} \cap H_{l_2} \cap \cdots \cap H_{l_{k-1}} \tag{3.1}$$

for $l_1 \in I_1, l_2 \in I_2, \cdots, l_{k-1} \in I_{k-1}$. We then define a *search tree* as the vectors $w(1), w(2), \cdots, w(p)$ together with the following subspaces:

$$\begin{aligned}
&H_{l_1} && i_1 \in I_1, \ (i_1 \neq l_1),\\
&H_{l_1} \cap H_{l_2} && i_2 \in I_2, \ (i_2 \neq l_2),\\
&H_{l_1} \cap H_{l_2} \cap H_{l_3} && i_3 \in I_3, \ (i_3 \neq l_3)\\
&\quad \vdots\\
&H_{l_1} \cap H_{l_2} \cdots \cap H_{l_{p-2}} \cap H_{l_{p-1}} && \\
&\qquad i_{p-1} \in I_{p-1}, \ (i_{p-1} \neq l_{p-1}),\\
&H_{l_1} \cap H_{l_2} \cap \cdots \cap H_{l_{p-1}} \cap H_{l_p} && i_p \in I_p.
\end{aligned} \tag{3.2}$$

The above set of vectors and subspaces can be represented schematically by the diagram shown in Fig. 4. We may think of $w(1), w(2), \cdots, w(p)$ as "starting vectors" for examining a series of subspaces for relative optimal solutions. Thus, $w(1)$ is the starting vector for R^{n+1}, $w(2)$ the starting vector for H_{l_1}, $w(3)$ the starting vector for $H_{l_1} \cap H_{l_2}$, and so on. These vectors play the role of w in Lemmas 1 and 2. The top-level branches in Fig. 4 represent the subspaces H_i, $i \in I_1$, the second-level branches the subspaces $H_{l_1} \cap H_i$, $i \in I_2$, and so on. According to Lemmas 1 and 2, these are the subspaces that must be examined for their relative optimal solutions in order to obtain relative optimal solutions for R^{n+1}, H_{l_1}, $H_{l_1} \cap H_{l_2}, \cdots$. For clarity we have not labeled all the branches in the diagram of Fig. 4, but have instead indicated the index sets from which these labels would come.

The following theorem generalizes Lemma 2 and establishes that a search tree leads to an optimal solution of (2.1).

Theorem 1: Let $w(1), w(2), \cdots, w(p)$ satisfy (3.1) and let T_p be the union of all the subspaces in (3.2). Then, there is at least one optimal vector relative to R^{n+1} in the set

$$T_p \cup \{w(k) | k = 1, 2, \cdots, p\} \tag{3.3}$$

for $p = 1, 2, \cdots, n$.

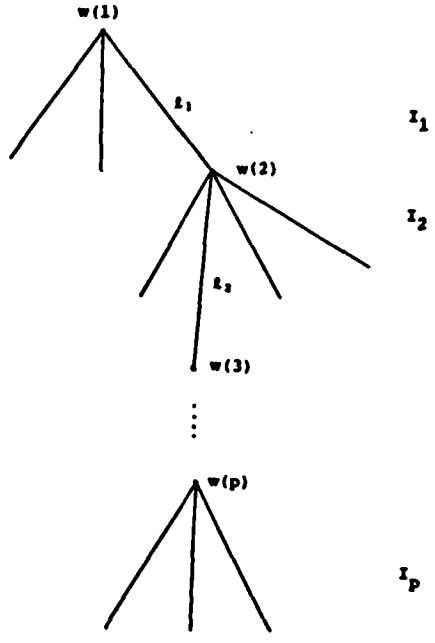


Fig. 4. Schematic diagram of a general search tree.

Proof: Consider the following subspaces:

$$\begin{aligned}
Q_1 &= \cup \{H_i | i_1 \in I_1, i_1 \neq l_1\}\\
Q_2 &= \cup \{H_{l_1} \cap H_{i_2} | i_2 \in I_2, i_2 \neq l_2\}\\
&\vdots\\
Q_{p-1} &= \cup \{H_{l_1} \cap \cdots \cap H_{l_{p-2}}\\
&\qquad \cap H_{i_{p-1}} | i_{p-1} \in I_{p-1}, i_{p-1} \neq l_{p-1}\}\\
\hat{Q}_p &= \cup \{H_{l_1} \cap \cdots \cap H_{l_{p-1}} \cap H_{ip} | i_p \in I_p\}
\end{aligned} \tag{3.4}$$

where, in the last expression, inclusion of the "$\wedge$" means that i_p is allowed to equal l_p. It then follows from the statement of the theorem that

$$T_p = Q_1 \cup Q_2 \cup \cdots \cup \hat{Q}_p. \tag{3.5}$$

For $p = 1$, $T_1 = \cup \{H_{i_1} | i_1 \in I_1\}$ and the theorem reduces to Lemma 1. For $p > 1$ we use Lemma 2 and induction on p. Assume that the theorem is true for $p = q$; we then wish to prove that it is also true for $p = q + 1$. In other words, we assume that

$$T_q \cup \{w(1), w(2), \cdots, w(q)\} \tag{3.6}$$

contains at least one optimal solution and we wish to prove that this is also true for

$$T_{q+1} \cup \{w(1), w(2), \cdots, w(q), w(q+1)\}. \tag{3.7}$$

From the above use of the symbol "$\wedge$", we have that

$$\begin{aligned}
\hat{Q}_q &= Q_q \cup \{H_{l_1} \cap H_{l_2} \cap \cdots \cap H_{l_{q-1}} \cap H_{iq} | i_q = l_q\}\\
&= Q_q \cup S
\end{aligned} \tag{3.8}$$

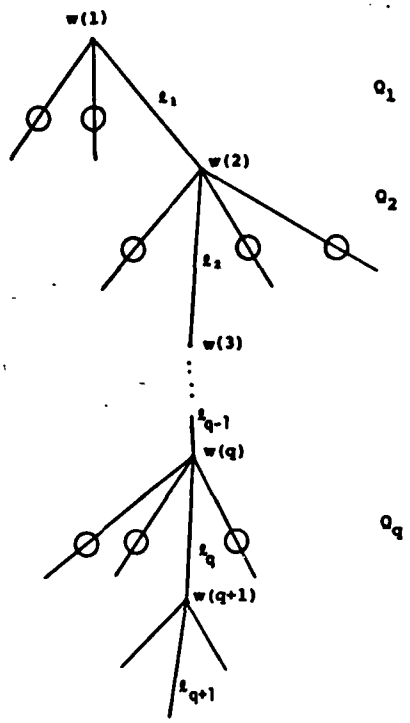


Fig. 5. Search tree diagram used in the proof of Theorem 1. The circled branches in each level represent the subspaces (hyperplanes) used in forming the set Q at that level.

where

$$S = H_{l_1} \cap H_{l_2} \cap \cdots \cap H_{l_{q-1}} \cap H_{l_q}. \tag{3.9}$$

The situation is shown in Fig. 5, in which $Q_1, Q_2, \cdots, Q_q$ are the subspaces formed by the union of hyperplanes represented by the circled branches. To form $\hat{Q}_q$ we simply include hyperplane H_{l_q} in the union of the hyperplanes forming Q_q.

It is noted in Fig. 5 that $w(q+1) \in H_{l_1} \cap H_{l_2} \cap \cdots \cap H_{l_{q-1}} \cap H_{l_q}$. In other words, $w(q+1) \in S$. It then follows from Lemma 2 that either $w(q+1)$ is an optimal solution relative to S or there is at least one optimal solution relative to S in the subspaces $S \cap H_{i_{q+1}}$, $i_{q+1} \in I_{q+1}$. Representing these subspaces by $\cup \{S \cap H_{i_{q+1}} | i_{q+1} \in I_{q+1}\}$, we note that

$$\cup \{S \cap H_{i_{q+1}} | i_{q+1} \in I_{q+1}\} = \hat{Q}_{q+1}. \tag{3.10}$$

Since

$$T_q = Q_1 \cup Q_2 \cup \cdots \cup \hat{Q}_q,$$

$$T_{q+1} = Q_1 \cup Q_2 \cup \cdots \cup Q_q \cup \hat{Q}_{q+1},$$

and $Q_q = \hat{Q}_q - S$ (where "$-$" represents set subtraction) it follows that

$$T_{q+1} = (T_q - S) \cup \hat{Q}_{q+1} \tag{3.11}$$

so that

$$\begin{aligned} T_{q+1} \cup \{w(1), w(2), \cdots, w(q), w(q+1)\} \\ = [(T_q - S) \cup \{w(1), w(2), \cdots, w(q)\}] \\ \cup [\hat{Q}_{q+1} \cup w(q+1)]. \end{aligned} \tag{3.12}$$

In order to finish the proof, we only need to show that either of the subspaces $[(T_q - S) \cup \{w(1), w(2), \cdots, w(q)\}]$ or $[\hat{Q}_{q+1} \cup w(q+1)]$ contains an optimal solution relative to R^{n+1}. Letting z represent an optimal solution relative to S, we know from Lemma 2 that either $z = w(q+1)$ or $z \in \hat{Q}_{q+1}$; that is, $\hat{Q}_{q+1} \cup w(q+1)$ contains an optimal solution relative to S. If z is also an optimal solution relative to R^{n+1} we are finished with the proof. If this is not the case, then S does not contain an optimal solution relative to R^{n+1} and it may be deleted from further consideration in the proof, leaving the subspace $T_q \cup \{w(1), w(2), \cdots, w(q)\}$. However, we know from the induction hypothesis that this subspace contains at least one optimal solution relative to R^{n+1}. This concludes the proof. □

C. Reduced Search Trees

The number of branches that are investigated in a search tree can be reduced by keeping a record of the subspaces that have already been searched. This will eliminate computation of the same information more than once and thus reduce the time required to complete the search for an optimal solution. In this section we consider techniques for reducing search trees and prove that a reduced search tree will lead to an optimal solution of (2.1).

Let $w(1), w(2), \cdots, w(p)$ be nonzero vectors in R^{n+1}, and let $I_1', I_2', \cdots, I_p'$ and $J_1, J_2, \cdots, J_p$ be sets of integers between 1 and m satisfying the following conditions:

$$I_i' \subset I_i, \qquad i = 1, 2, \cdots, p \tag{3.13}$$

$$I_i' \cap J_j = \phi, \quad i = 1, 2, \cdots, p; \quad j = 1, 2, \cdots, i \tag{3.14}$$

$$\begin{aligned} &I_1 \subset (I_1' \cup J_1) \\ &I_2 \subset (I_2' \cup J_1 \cup J_2) \\ &\vdots \\ &I_p \subset (I_p' \cup J_1 \cup \cdots \cup J_p). \end{aligned} \tag{3.15}$$

A *reduced search tree* is defined as a set of vectors $w(k), k = 1, 2, \cdots, p$ satisfying (3.1) for some values $l_1 \in I_1', l_2 \in I_2', \cdots, l_{p-1} \in I_{p-1}'$ together with subspaces of the form

$$\begin{aligned} &H_{l_1} && i_1 \in I_1' \cup J_1, i_1 \neq l_1 \\ &H_{l_1} \cap H_{i_2} && i_2 \in I_2' \cup J_2, i_2 \neq l_2 \\ &\vdots \\ &H_{l_1} \cap \cdots \cap H_{l_{p-2}} \cap H_{i_{p-1}} && i_{p-1} \in I_{p-1}' \cup J_{p-1}, \quad i_{p-1} \neq l_{p-1} \\ &H_{l_1} \cap \cdots \cap H_{l_{p-1}} \cap H_{i_p} && i_p \in I_p' \cup J_p \end{aligned} \tag{3.16}$$

where I_k' and $J_k, k = 1, 2, \cdots, p$, satisfy (3.13)-(3.15).

The diagram of the search tree just defined is shown in Fig. 6. The interpretation given to the sets I_k' and J_k is that elements of J_k indicate subspaces (shown as dashed branches) already examined for a relative optimal solution, while elements of I_k' indicate subspaces yet to be examined, or in the process of being examined. These subspaces are denoted by solid branches in Fig. 6. Condition (3.13) indicates that in-

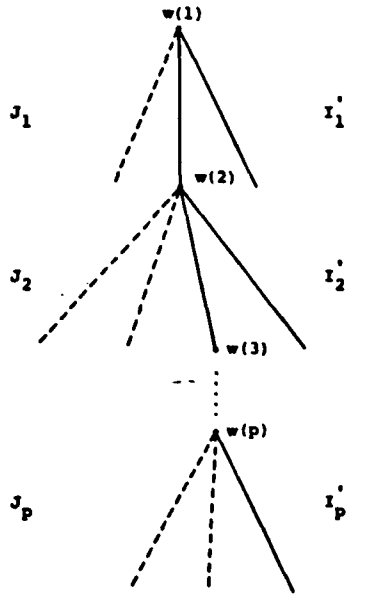


Fig. 6. Diagram of a reduced search tree.

dexes corresponding to subspaces to be examined are elements of a valid index set. Condition (3.14) indicates that we need not examine any subspace of a subspace already examined. Condition (3.15) is a requirement that we examine each of the subspaces corresponding to the index set I_i that have not already been examined. It is noted that the set I_i' (which gives the indexes of hyperplanes to be considered at the ith level) is formed by deleting from I_i any indexes contained in J_j, $j = 1, 2, \cdots, i$.

Corollary 1: Let $w(1), w(2), \cdots, w(p)$ satisfy (3.1) for $l_1 \in I_1'$, $l_2 \in I_2', \cdots, l_p \in I_p'$, where $I_1', I_2', \cdots, I_p'$ satisfy (3.13), and let V_p be the union of all the sets in (3.16), where $I_1', I_2', \cdots, I_p', J_1, J_2, \cdots, J_p$, satisfy (3.14) and (3.15). Then, there is at least one optimal solution vector in the set

$$V_p \cup \{w(k) | k = 1, 2, \cdots, p\}. \tag{3.17}$$

Proof: The proof follows from Theorem 1 by noting that $I_1 \subset (I_1' \cup J_1), \cdots, I_p \subset (I_p' \cup J_1 \cup \cdots \cup J_p)$ and that $T_p \subset V_p$. □

Another important property of search trees that leads to further reductions in computation is that any of the starting vectors and its index set may be replaced by a bottom-level starting vector and its index set. The result will still be a search tree that satisfies Theorem 1 and its corollary.

In order to illustrate how the condition given in (3.14), along with the above replacement procedure, can be used to reduce the search for an optimal solution, consider Fig. 7(a) which shows a tree at some stage of a hypothetical search. The branches taken from left to right and top to bottom represent, respectively,

$$H_1, H_4, H_5$$

$$H_4 \cap H_3, H_4 \cap H_8, H_4 \cap H_7, H_4 \cap H_9$$

and the dashed branches represent subspaces that have already been investigated for relative optimal vectors. It is noted that the index sets of subspaces to be investigated (i.e., I_1' and I_2') do not contain the indexes of subspaces already searched at, or above the second level of the tree, as indicated in condition (3.14). Suppose now that a new vector, $u = w(3)$, lying in $H_4 \cap H_7$ has been computed and its index set is $I_3 = \{1, 8, 10\}$, as shown in Fig. 7(b). In order to continue the search using this new vector, we have the three possibilities shown in Fig. 7(c)-(e). In Fig. 7(c) we simply leave $w(3)$ in position and delete the branches labeled 1 and 8 because they represent subspaces that were already investigated at a higher level in the tree. In Fig. 7(d), $w(2)$ and its descendants were replaced by u and its descendants, deleting at this level any dashed branches that appear at a higher level (i.e., the branch labeled 1 in this case). It is noted that any dashed branches that do not satisfy this condition (i.e., the branch labeled 3) are retained to show later in the search that they have been investigated at that level of the tree. Finally, Fig. 7(e) shows the entire tree replaced by u and its descendants. At this level only H_1 has been investigated so the dashed branch labeled 1 is retained and the branches labeled 8 and 10 are solid, indicating that they still have to be searched for a relative optimal solution. It is noted that all three possibilities in Fig. 7(c)-(e) satisfy the conditions of Theorem 1 and its corollary; thus, either of the three choices to continue the search will lead to an optimal solution. Our goal is to choose the candidate with the most potential for reducing the search. The criterion we will use is to place u at the highest possible level in the tree such that the number of solid branches at that level is less than before. This criterion seeks to reduce the search by trimming off, at the highest possible level, subspaces that would have been investigated in the original tree. We are thus lead to the following rule.

Replacement and Deletion Rule: Let $u = w(r)$ with original index set $I(u)$ be a vector computed at the bottom level r of a search tree. For values $q = 1, 2, \cdots, r-1$, we let

$$I_q'(u) = I(u) - \{J_1 \cup J_2 \cup \cdots \cup J_q\} \tag{3.18}$$

and

$$J_q(u) = J_q \tag{3.19}$$

where "-" indicates set subtraction. We then choose the smallest value of q, if any, for which cardinality $[I_q'(u)] <$ cardinality $[I_q']$, and replace $w(q)$ and its index sets by u and the index sets given in (3.18) and (3.19), deleting all descendants of $w(q)$. If no such value of q exists, no replacement takes place and the index sets at the rth level are given by

$$I_r' = I(u) - \{J_1 \cup J_2 \cup \cdots \cup J_{r-1}\} \tag{3.20}$$

and

$$J_r = \phi. \tag{3.21}$$

It is noted that any indexes of subspaces already investigated at a higher level in the tree are deleted from $I(u)$ to form $I_q'(u)$ and that $J_q(u)$ retains all indexes of subspaces already investigated at level q. In (3.21), $J_r = \phi$ because r is at the bottom level of the search tree and no subspaces have yet been investigated at that level.

Returning to the example in Fig. 7, we see that the above

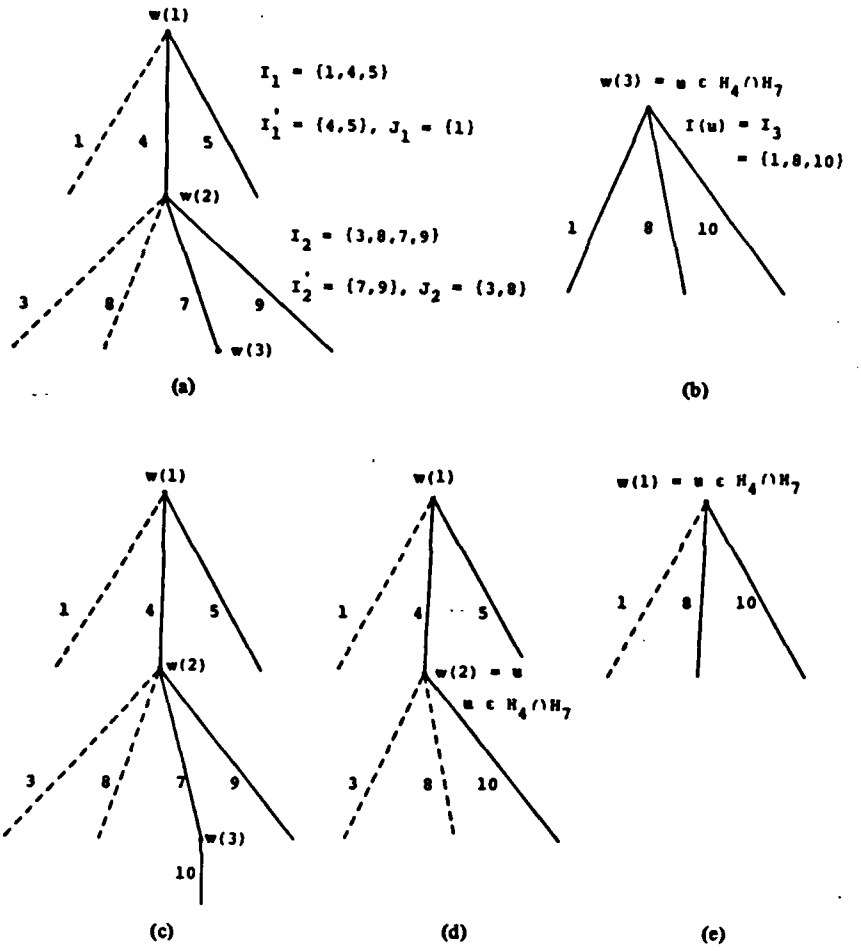


Fig. 7. (a) Search tree. (b) New vector computed in $H_4 \cap H_7$. (c) through (e). The three possibilities involving $u = w$ (3) in the continuation of the search.

rule would choose the configuration shown in Fig. 7(d) to continue the search.

D. Ordering of the Index Sets

Each node in a search tree represents a vector established by the intersection of n hyperplanes. As shown in Fig. 8, a vector at the kth level in the tree satisfies the requirement

$$w(k) \in H_{l_1} \cap H_{l_2} \cap \cdots \cap H_{l_{k-1}} \cap H_{l_k} \cap H_{l_{k+1}} \cap \cdots \cap H_{l_n} \tag{3.22}$$

where $1 \leq k \leq n$ and $l_i \in I_i'$, $i = 1, 2, \cdots, k-1$. As indicated in Section III-B, we compute a vector $w(k+1)$ at the next level by replacing H_{l_k} by a hyperplane H_{i_k} with $i_k \in I_k'$. In order to stress the dependence of $w(k+1)$ on i_k we will, in this section, represent these vectors by $w(k+1, i_k)$. Then,

$$w(k+1, i_k) \in H_{l_1} \cap H_{l_2} \cap \cdots H_{l_{k-1}} \cap H_{l_k} \cap H_{l_{k+1}} \cap \cdots \cap H_{l_n} \tag{3.23}$$

for some $i_k \in I_k'$. There are as many of these vectors as there are elements in I_k'. In order to continue the search we may take any of these, obtain I_{k+1}', compute $w(k+2)$, and so on.

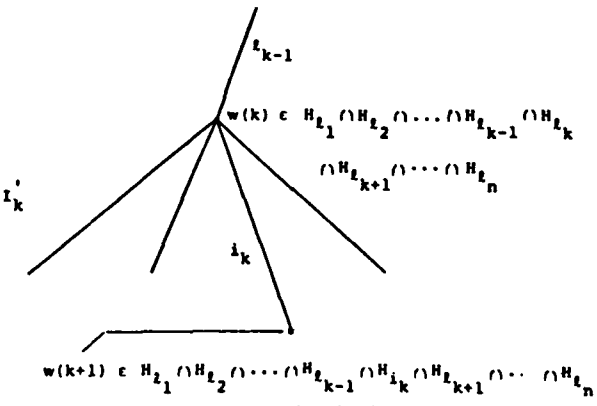


Fig. 8. Illustration of a vector $w(k)$ at level k of a search tree and the computation of $w(k+1)$ by replacing H_{l_k} by H_{i_k}, $i_k \in I_k'$.

However, since we are seeking a minimum-error solution, it would be advantageous to be able to select the $w(k+1, i_k)$ with the smallest error as the next candidate in the search. The brute-force method of computing all $w(k+1, i_k)$ and choosing the best one would be in general unacceptable in view of the fact that the *replacement and deletion rule* will in

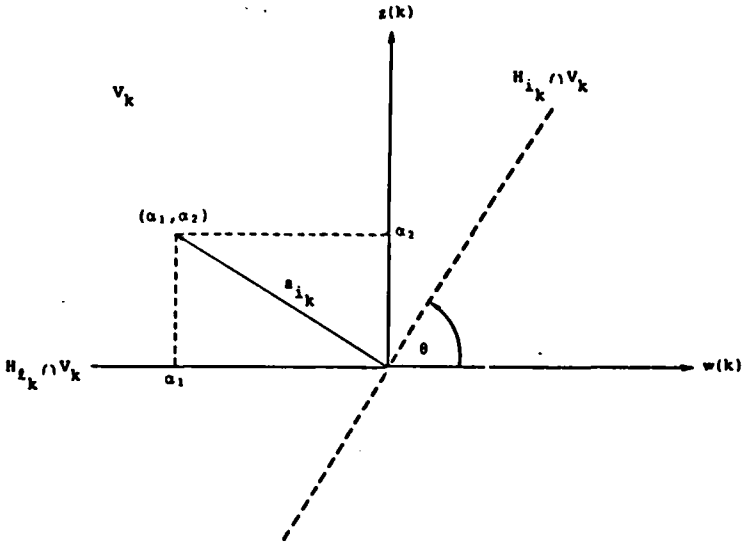


Fig. 9. Example of subspace V_k.

many cases eliminate the consideration of some of these vectors in the first place. In this section we develop a technique for obtaining the errors of all $w(k+1, i_k)$, $i_k \in I_k'$, without actually having to obtain these vectors directly. These error values can then be used to rearrange the elements of I_k' so that they are considered in order of increasing error. As shown below, the method is very economical from a computational point of view.

The subspace

$$V_k = H_{l_1} \cap H_{l_2} \cap \cdots \cap H_{l_{k-1}} \cap H_{l_{k+1}} \cap \cdots \cap H_{l_n} \tag{3.24}$$

is two-dimensional and it contains both $w(k)$, which is known, and $w(k+1, i_k)$, $i_k \in I_k'$, which are unknown. Let $z(k)$ be a vector contained in V_k and orthogonal to $w(k)$. (The vector $z(k)$ may be found, for example, by Gaussian elimination.) It then follows that each $w(k+1, i_k)$ may be expressed as a linear combination of $w(k)$ and $z(k)$.

Fig. 9 illustrates a typical geometrical configuration within the two-dimensional subspace V_k. A vector $w(k+1, i_k)$ lies in the intersection (shown as a dashed line) of H_{i_k} with V_k, and $w(k)$ lies in the intersection of H_{l_k} with V_k. The projection of the normal to H_{i_k} (see Section II) onto the V_k plane is shown with coordinates (α_1, α_2), where a_{i_k} is the row vector of A determined by the value of i_k. From simple geometrical considerations we have that

$$(\alpha_1, \alpha_2) = (a_{i_k} \cdot w(k)/\|w(k)\|, a_{i_k} \cdot z(k)/\|z(k)\|) \tag{3.25}$$

and

$$\frac{\alpha_1}{\alpha_2} = -\cot \Theta \tag{3.26}$$

where Θ is the angle from the $w(k)$ axis counterclockwise to the dashed half-line. Finally, we define the quantity $\gamma(j)$ as

$$\gamma(j) = a_j \cdot w(k)/a_j \cdot z(k) = -C \cot \Theta \tag{3.27}$$

where $C = \|w(k)\|/\|z(k)\|$.

Based on the foregoing concepts, we define the following *ordering rule.*

1) Let D be the set of indexes of the hyperplanes determining $w(k)$; that is, from (3.22), $D = \{l_1, l_2, \cdots, l_n\}$. A vector $w(k+1, i_k)$ lies on the hyperplanes with indexes in this set (except H_{l_k}) since $H_{l_1}, H_{l_2}, \cdots, H_{l_{k-1}}, H_{l_{k+1}}, \cdots, H_{l_n}$ define V_k.

2) Let $b(k)$ be given by

$$b(k) = \begin{cases} 0 & \text{if } a_{l_k} \cdot z(k) > 0 \\ 1 & \text{if } a_{l_k} \cdot z(k) < 0. \end{cases}$$

3) For each $i_k \in I_k'$, let

$M(i_k)$ = number of hyperplanes H_j for which $\gamma(j) < \gamma(i_k)$, $1 \leq j \leq m$, $j \notin I_k \cup D$

$N(i_k)$ = number of hyperplanes H_q for which $\gamma(q) < \gamma(i_k)$, $q \in I_k$.

4) For each $i_k \in I_k'$, compute the error of $w(k+1, i_k)$, as follows:

$$\text{err}[w(k+1, i_k)] = \text{err}[w(k)] + M(i_k) - (N(i_k) + 1) + b(k).$$

5) Rearrange the elements of I_k' in order of increasing error. Elements with the same error are ordered arbitrarily.

As an illustration of the ordering rule, consider a problem in which $m = 10$, $n = 4$, and suppose that we are at the second level in the search tree with $w(2) = H_2 \cap H_4 \cap H_8 \cap H_{10}$, $I_2 = \{3, 5, 6\}$, and $I_2' = \{5, 6\}$. In this case $V_2 = H_2 \cap H_8 \cap H_{10}$, $D = \{2, 4, 8, 10\}$, and suppose that $a_4 \cdot z(2) > 0$. The situation is shown in Fig. 10, where the hyperplanes with indexes in I_2' are circled. Note that H_2, H_8, and H_{10} are not shown because they define V_2. By definition, $w(2)$ lies on the negative side of the hyperplanes with indexes in I_2 and on the positive side of all other hyperplanes with indexes not in D. Since $a \cdot z(2) > 0$, $z(2)$ lies on the positive side of $H_4 \cap V_2$. The error of $w(2)$ is 3 because I_2 contains three elements. We also note that, since $-\cot \Theta$ is an increasing function of Θ for $0 < \Theta < \pi$, $\gamma(1) < \gamma(5) < \gamma(9) < \gamma(3) < \gamma(6) < \gamma(7)$. Thus, to compute err $[w(3, 5)]$ we first compute $M(5) = 1$ and

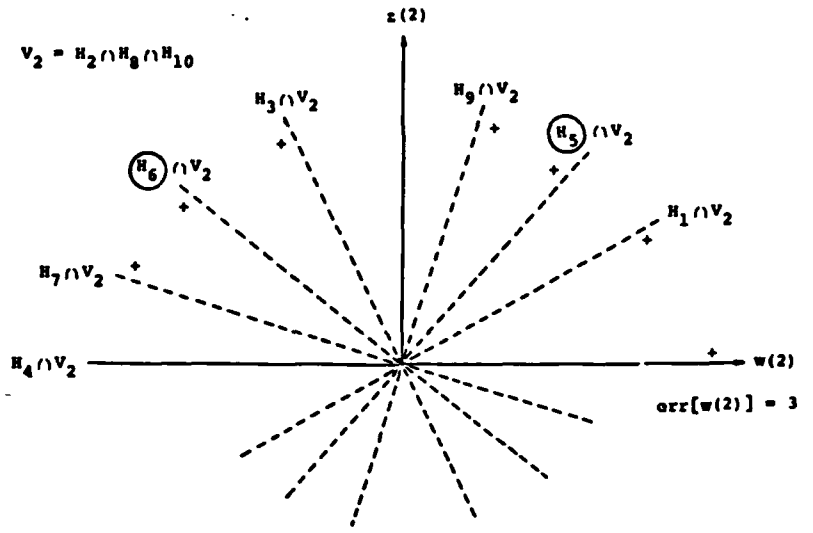


Fig. 10. Illustration of the *ordering rule.*

$N(5) = 0$. Then, since $b(2) = 0$, it follows from step 4) that err $[w(3, 5)] = 3$. Similarly, $M(6) = 2, N(6) = 2$, and err $[w(3, 6)] = 2$. Thus, the ordered index set becomes $I_2' = \{6, 5\}$.

The key to the above procedure lies in the fact that $-C \cot \Theta$ is an increasing function of Θ for $0 < \Theta < \pi$ and, thus, can be used to determine the positive and negative side of any hyperplane with respect to a vector contained in the one-dimensional subspace $H_{i_k} \cap V_k$ and oriented in the $0 < \Theta < \pi$ direction. For instance, in the example just described, it is noted that $M(5)$ gives the number of hyperplanes for which $\gamma(j) < \gamma(5)$, $j \notin I_k \cup D$; that is, $M(5)$ is the number of hyperplanes whose intersection with V_2 are to the right of a vector contained in $H_5 \cap V_2$, excluding hyperplanes with indexes in I_2 and D. Since these exclusions guarantee that all hyperplanes used in the computation of $M(5)$ have their positive side facing $w(2)$, it follows that the intersection of any of these hyperplanes which is to the right of a vector contained in $H_5 \cap V_2$ will yield an error (negative product) with respect to a vector $w(k + 1, i_k)$, $k = 2$, $i_k = 5$, contained in that one-dimensional subspace and oriented in the $0 < \Theta < \pi$ direction. In the above example only one hyperplane, H_1, satisfies the conditions necessary for use in the computation of $M(5)$, yielding $M(5) = 1$. Similarly, $N(i_k)$ is the number of hyperplanes whose intersections with V_k are to the right of a vector $w(k + 1, i_k)$ in $H_{i_k} \cap V_k$, but whose indexes are in I_k and, therefore, have their negative sides facing $w(k)$. Since err $[w(k)]$ gives the total number of these hyperplanes, it follows that the quantity err $[w(k)] - N(i_k) - 1$ is the error with respect to $w(k + 1, i_k)$ of all hyperplanes with indexes in I_k. (The -1 is included to reduce err $[w(k)]$ by one because H_{i_k} contributed to this error since $i_k \in I_k$; however, H_{i_k} contains $w(k + 1, i_k)$ and, therefore, does not contribute to err $[w(k + 1, i_k)]$). In the present example with $k = 2$ and $i_k = 5$ we have that $N(5) = 0$.

At this point all hyperplane intersections with V_k, except $H_{i_k} \cap V_k$ ($I_k = 4$), have been taken into account. In order to establish the contribution of H_{i_k} to err $[w(k + 1), i_k)]$, it is only necessary to determine the positive side of $H_{i_k} \cap V_k$ with respect to the half space $0 < \Theta < \pi$. This is easily accomplished by using $z(k)$. If $a_{i_k} \cdot z(k) > 0$, the positive side of $H_{i_k} \cap V_k$ faces the half space just mentioned and H_{i_k} does not contribute to err $[w(k + 1, i_k)]$; that is, $b(k) = 0$. Otherwise, the error is increased by one by letting $b(k) = 1$. In the present example $I_k = 4$ and we have that $b(2) = 0$ because it was assumed that $a_4 \cdot z(2) > 0$.

With reference to step 4) of the *ordering rule*, and based on foregoing discussion, it is seen that all contributions to err $[w(k + 1, i_k)]$ have been taken into account. The hyperplanes with indexes in D were not considered because, with the exception of H_{i_k}, they combine with H_{i_k} to form the edge containing $w(k + 1, i_k)$, as shown in (3.23). As indicated in Section II, $w(k + 1, i_k)$ can be displaced so that it yields a positive product with all the edge hyperplanes, so they would not contribute to the error of this vector.

E. Statement of the Algorithm

The concepts developed in Sections III-A–D lead to the following algorithm for finding an optimal solution of (2.1).

Notation:

w^*	an optimal solution of (2.1) at the termination of the algorithm
k	level in the search tree
E_k	set of indexes of n hyperplanes used to compute an edge vector at the kth level; this vector can be computed by Gaussian elimination or by using the method discussed in [5]
$w(k)$	edge vector computed at level k
I_k	index set of $w(k)$
J_k	index set of subspaces already examined at level k
I_k'	index set of subspaces to be examined at level k
ϕ	empty set

Step 1 (*Initialization*): Let

a) w^* = arbitrary starting vector[1];

b) $J_j = \phi$, $j = 1, 2, \cdots, n - 1$;

[1] An alternative is to start with a quasi-optimal vector determined, for example, by a procedure such as Fisher's [3]. We have found, however, that progress toward an optimal solution is at first rapid, thus partially negating any advantage that may be gained by using additional (and more complex) techniques to estimate a "better" starting vector.

c) $E_1 = \{1, 2, \cdots, n\}$; that is, select the first n hyperplanes to compute $w(1)$;

d) $k = 1$.

Step 2:

a) Compute $w(k)$ using the hyperplanes with indexes in E_k.

b) Compute I_k.

c) If $\text{err}[w(k)] < \text{err}(w^*)$, let $w^* = w(k)$.

d) If $\text{err}[w(k)] = 0$, go to Step 9.

e) If $k = n + 1$, set $k = n$ and go to Step 8.

Step 3: Apply the *replacement and deletion rule*, denoting by q the level at which replacement took place. The index sets I'_q and J_q are given by (3.18) and (3.19).

Step 4:

a) Let $p = k, k = q, I'_k = I'_q$ and $J_k = J_q$.

b) Let $J_j = \phi$ for all $j > k$.

c) If $I'_k = \phi$, go to Step 7. Otherwise, rotate the elements in E_p starting in the kth position so that the element in the pth position goes into the kth position. The elements to the left of the kth position are not disturbed. Replace the elements in E_k by the elements in E_p after rotation.[2]

Step 5:

a) Rearrange the elements of I'_k in order of increasing error by applying the *ordering rule*. Let $e_{\min}$ be the minimum error found.

b) Let $j = 1$.

Step 6:

a) If $k = n$ and $e_{\min} > \text{err}(w^*)$, go to Step 8. Otherwise, continue.

b) Let $E_{k+1} = E_k$ and then replace the kth element of E_{k+1} by the jth element of I'_k.

c) Increment k and j by 1.

d) Go to Step 2.

Step 7: If $k = 1$, go to Step 9. Otherwise, set $J_k = \phi$ and continue.

Step 8:

a) Decrement k by 1.

b) Transfer the jth element of I'_k to J_k to indicate that another subspace has been searched at level k.

c) If the jth element was the last element of I'_k, go to Step 7. Otherwise, go to Step 6.

Step 9: Stop with edge vector w^* as an optimal solution of (2.1). Displace w^* into the interior of the optimal cone by using the procedure described in [5].

IV. Bounds on the Number of Edges Investigated by the Algorithm

An exhaustive search for an optimal solution carried out by obtaining one vector in each cone edge would require the computation of C_n^m such vectors. For algorithms employing the concept of an index set to reduce the search, it has been shown [5] that the ratio $W_{m,n}/C_n^m$ is strictly less than 1, where $W_{m,n}$ is the number of edge vectors computed under worst case conditions at each step in the search. Experimental results [5] indicate that the actual number of vectors computed in a search can be expected to be considerably less than the theoretical upper bound.

The theoretical upper bound derived in [5] would apply to the algorithm developed in Section III (since it too is based on the use of an index set) in the case when the search is never restarted and none of the index sets are ordered. When use is made of restarting and ordering we would expect the number of vectors that need to be investigated to be significantly reduced, and the results presented in the next section bear this out. Aside from the fact that a theoretical upper bound can be established under worst case conditions, derivation of an upper bound that takes into account restarting and ordering does not appear feasible because the advantages derived from these procedures are data dependent. It is possible, however, to obtain an expression for the lower bound (as a function of the error of an optimal solution) on the number of vectors that *must* be investigated by the algorithm. This result, given as a corollary of the following theorem, is quite useful because it establishes a guideline for the minimum amount of computational work required to find an optimal solution of (2.1).

Theorem 2: Let $e = \text{err}(w^*)$ be the number of errors incurred by an optimal solution, w^*, of (2.1). Suppose that the algorithm commences searching a subspace $\hat{H}$ of dimension $p \geq 2$ when r subspaces of dimension $\geq p$ have already been searched, where $r \leq e$. Then, in order to complete the search of $\hat{H}$, the algorithm must compute at least $C_{p-2}^{e-r+p-2}$ edge vectors contained in $\hat{H}$.

Proof: The proof is by induction on p. When $p = 2$, the theorem asserts that the algorithm computes at least one edge in $\hat{H}$ to complete the search of $\hat{H}$. This assertion is obviously true. Suppose the assertion of the theorem is true for $p = k$; we wish to prove that it is true for $p = k + 1$, where $2 \leq p \leq n + 1$.

When the algorithm commences searching $\hat{H}$, it first computes an edge vector $w \in \hat{H}$ and the set $I(w)$. The cardinality of $I(w)$ is at least e, since the minimum error of any edge vector in R^{n+1} is e. According to the induction hypothesis, r subspaces of dimension $\geq k + 1$ have already been searched, where $r \leq e$.

If $r = e$, it is possible that all of the subspaces of the form $\hat{H} \cap H_i$, $i \in I(w)$, are subspaces of the r subspaces which have already been searched, in which case the algorithm has already completed the search of $\hat{H}$, having computed one edge w in $\hat{H}$. The assertion of the lemma is that the algorithm must compute at least

$$C_{p-2}^{e-r+p-2} = C_{p-2}^{p-2} = 1 \tag{4.1}$$

edges in $\hat{H}$; hence, it is a true assertion in this case.

If $r < e$, then the algorithm must search some subspaces of the form $\hat{H} \cap H_i$. Consider first the case where throughout the search there is no replacement of the initial vector w and index set $I(w)$. Then the algorithm must search at least $e - r$ of the subspaces of the form $\hat{H} \cap H_i$, $i \in I(w)$, each of which is of dimension k. Upon commencing to search the first of

[2] For example, suppose $k = 2$, $p = 4$, and $E_4 = \{e_1, e_2, e_3, e_4, e_5\}$. We rotate the elements starting in the second position so that the element in the fourth position goes into the second position. The elements to the left of the second position are not disturbed. After rotation and letting $E_2 = E_4$ we then have $E_2 = \{e_1, e_4, e_5, e_2, e_3\}$. This step updates the hyperplane indexes at level k by taking into account the fact that a vector at level p was brought up to level k.

these subspaces, there are r subspaces of dimension $\geq k$ already searched. Upon commencing the search of the qth of these subspaces, $1 \leq q \leq e - r$, there are $r + q - 1$ subspaces of dimension $\geq k$ already searched. Therefore, by use of the induction hypothesis, we see that to search the qth of the subspaces $\hat{H} \cap H_i$ requires the computation of at least $C_{k-2}^{e-r-q+1+k-2}$ edges, and that the search of $\hat{H}$ requires at least

$$1 + \sum_{q=1}^{e-r} C_{k-2}^{e-r-q+1+k-2} \tag{4.2}$$

edge computations. By letting $j = e - r - q + 1$, this quantity can also be expressed as

$$1 + \sum_{j=1}^{e-r} C_{k-2}^{j+k-2}. \tag{4.3}$$

It is easy to show that (4.3) is equal to

$$C_{k-1}^{e-r+k-1}. \tag{4.4}$$

Hence the lower bound of Theorem 2 has been verified for $p = k + 1$ in the case where there is no replacement of the initial vector w.

In the case where w is replaced, it can be seen that the same lower bound is still valid because the algorithm must still search at least $e - r$ subspaces of the form $\hat{H} \cap H_i$, and the qth subspace searched requires the computation of at least $C_{k-2}^{e-r-q+k-2}$ edges, as before. This concludes the proof. □

Corollary 2: Let $e = \text{err}(w^*)$ be the number of errors incurred by an optimal solution, w^*, of (2.1). Then the lower bound on the number of edge vectors computed during execution of the algorithm is given by the binomial coefficient

$$C_{n-1}^{e+n-1} = \frac{(e+n-1)(e+n-2)\cdots(e+1)}{(n-1)!}. \tag{4.5}$$

Proof: The proof follows from Theorem 2 with $\hat{H} = R^{n+1}$, $p = n + 1$, and $r = 0$. □

The lower bound given by (4.5) is shown in Table I for various values of e and n. These values are compared in the next section against the number of edge vectors actually computed by the algorithm in a number of examples.

TABLE I
LOWER BOUND ON THE NUMBER OF EDGES COMPUTED BY THE ALGORITHM

e \ n	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
2	3	6	10	15	21	28	36	45	55
3	4	10	20	35	56	84	120	165	220
4	5	15	35	70	126	210	330	495	715
5	6	21	56	126	252	462	792	1,287	2,002
6	7	28	84	210	462	924	1,716	3,003	5,005
7	8	36	120	330	792	1,716	3,432	6,435	11,440
8	9	45	165	495	1,287	3,003	6,435	12,870	24,310
9	10	55	220	715	2,002	5,005	11,440	24,310	48,620
10	11	66	286	1,001	3,003	8,008	19,448	43,758	92,378

TABLE II
PAIRWISE DISCRIMINATION OF THE IRIS DATA CLASSES

Classes discriminated:	Versicolor, Virginica				
Weight vector:	0.0038	0.0070	-0.0208	-0.0251	1.0000
Error:	1				
Lower bound:	4				
No. of edges computed:	49				
CPU time:	1.61 sec.				
Classes discriminated:	Virginica, Setosa				
Weight vector:	-0.0087	0.0131	0.0257	-0.0033	-1.0000
Error:	0				
No. of edges computed:	10				
CPU time:	0.34 sec.				
Classes discriminated:	Setosa, Versicolor				
Weight vector:	0.0378	-0.0118	-0.0164	-0.0355	-1.0000
Error:	0				
No. of edges computed:	19				
CPU time:	0.59 sec.				

V. EXPERIMENTAL RESULTS

The algorithm developed in Section III was programmed in Fortran IV and run on an IBM 370/3031. The following results illustrate the performance of the procedure in separable and inseparable situations.

Experiment 1: The first example is based on the measurements performed by Fisher [3] on three types of iris flowers: Iris Versicolor, Iris Virginica, and Iris Setosa. For each type of flower, four measurements (petal length and width and sepal length and width) were taken on 50 specimens. This leads us to consider three two-class discrimination problems with $m = 100$ and $n = 4$. The results summarized in Table II agree with the well-known fact that two of the pairs are separable, and one pair is inseparable with the optimal solution yielding one error. It is of interest to compare the lower bound given in Table I with the actual number of edge vectors computed by the algorithm in the inseparable case. In the separable case, the minimum number of edges that the algorithm must compute is, of course, 1.

Experiment 2: In this experiment we compared the performance of the algorithm against the procedure developed by Warmack and Gonzalez [5] which is also based on the use of an index set. Two four-dimensional Gaussian classes of 50 patterns each were generated using a program developed by Bryan and Tebbe [16]. The two pattern populations were

TABLE III
RESULT OF EXPERIMENT WITH FOUR-DIMENSIONAL, INSEPARABLE GAUSSIAN DATA

Weight vector:	-0.4903	-0.2100	-0.3896	-0.2276	1.0000
Error:	4				
Lower bound:	35				
No. of edges computed:	235				
CPU time:	8.85 sec.				

TABLE IV
RESULT OF EXPERIMENT WITH SIX-DIMENSIONAL, SEPARABLE GAUSSIAN DATA

Weight vector:	-7.1933	-9.6779	1.5035	-2.7608	-4.2783
	-3.6190	1.0000			
Error:	0				
No. of edges computed:	36				
CPU time:	1.67 sec.				

TABLE V
RESULT OF EXPERIMENT WITH TWO FIVE-DIMENSIONAL PATTERN CLASSES SEPARATED BY THE BOUNDARY $x_1 + x_2 + x_3 + x_4 + x_5 - 1.5 = 0$

Weight vector:	-0.7255	-0.2162	-0.6915	-0.6420	-0.6328	1.0000
Error:	0					
No. of edges computed:	82					
CPU time:	3.80 sec.					

generated by specifying an identity covariance matrix for each class with mean vectors (0, 0, 0, 0) and (1.5, 1.5, 1.5, 1.5), respectively. The results are summarized in Table III, which shows that 235 edges were investigated, requiring 8.85 s of CPU time. By contrast, the Warmack-Gonzalez algorithm investigated 15 854 edges requiring a total of 206 s of CPU time.

In another experiment we generated two groups of six-dimensional patterns with unity covariance matrices and mean vectors with components equal to -0.75 and 0.75, respectively. In this case the classes were linearly separable, and the results as summarized in Table IV, which shows that 36 edges, requiring 1.67 s, were investigated by our algorithm. By contrast, the Warmack-Gonzalez algorithm computed 51 826 edges, requiring approximately 13 min of CPU time.

Experiment 3: In this example we compare our algorithm against a procedure developed by Miyake [8]. Using the multivariate Gaussian generator mentioned in the previous example, two groups of 100, five-dimensional patterns with unity covariance matrices and means at 0 and (0.6, 0.6, 0.6, 0.6, 0.6) were generated satisfying the conditions $x_1 + x_2 + x_3 + x_4 + x_5 - 1.5 > 0$ and $x_1 + x_2 + x_3 + x_4 + x_5 - 1.5 < 0$, respectively, which is analogous to the separable data set used in [8]. The results are summarized in Table V, which shows that 82 edges were investigated, requiring 3.80 s of CPU time. Miyake reported CPU times of 10–25 min (depending on the starting weight vector) on a data set generated with the same parameters. He used a FACOM 230-45S, which is approximately four times slower than the IBM 370/3031. Taking into account the difference between the two machines, and using the lower 10 min figure to allow for programming and other variations in implementation between the two experiments, it appears that the procedure reported in [8] is, conservatively, on the order of 40 times slower than ours.

VI. CONCLUSIONS

The computational advantage of the algorithm developed in Section III over other direct procedures for finding an optimal solution of (2.1) is based on two principal factors: the *replacement and deletion rule* discussed in Section III-C, and the *ordering rule* developed in Section III-D. The first rule reduces the size of the search tree by trimming branches at the highest possible level and by keeping an account at each level of the subspaces that have been previously searched at that level. The ordering rule arranges subspaces in order of increasing error. This procedure enhances the computational efficiency of the algorithm by increasing the frequency with which lower-error edges are encountered during the search. As indicated in Section III-D, the use of an ordering procedure is made feasible by the fact that we are able to obtain the error of an edge vector without actually having to compute that vector.

The lower bound developed in Section IV for the minimum number of edges that must be computed by the algorithm provides a useful measure of the minimum amount of computational work required to find an optimal solution of (2.1). As in any search procedure, the number of edges investigated should be expected to grow rapidly as a function of the number and dimensionality of the patterns, as well as the error rate. Although no theoretical bound involving these parameters appears feasible, we have found in practice that the number of edges actually computed in inseparable cases in typically on the order of 10 to 30 times the lower bound. In separable situations, the algorithm has consistently found an optimal solution after computing a fraction of the number of edges required by the other direct procedures against which it has been compared.

Finally, we point out that although all discussions have been limited to a linear-discriminant function formulation, the concepts and procedures developed in this paper are also applicable to nonlinear discriminant functions by the standard preprocessing technique of using the nonlinear functions to map the patterns onto another space [1].

REFERENCES

[1] J. T. Tou and R. C. Gonzalez, *Pattern Recognition Principles*. Reading, MA: Addison-Wesley, 1974.
[2] R. C. Gonzalez and M. G. Thomason, *Syntactic Pattern Recognition: An Introduction*. Reading, MA: Addison-Wesley, 1978.
[3] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Ann. Eugenics*, vol. 7, pt. 2, pp. 179–188, 1936.
[4] T. Ibaraki and S. Muroga, "Adaptive linear classifier by linear programming," *IEEE Trans. Syst. Sci. Cybern.*, vol. SSC-6, no. 1, pp. 53–62, 1970.

[5] R. E. Warmack and R. C. Gonzalez, "An algorithm for the optimal solution of linear inequalities and its application to pattern recognition," *IEEE Trans. Comput.*, vol. C-22, no. 12, pp. 1065-1075, 1973.
[6] A. Miyake and S. Shinmura, "An algorithm for the optimal linear discriminant functions," in *Proc. Int. Conf. Cybern. Soc.*, vol. 2, 1978, pp. 1447-1450.
[7] S. Shinmura and A. Miyake, "Optimal linear discriminant functions and their application," in *Proc. Comput. Software Appl. Conf.*, Chicago, IL, Nov. 6-8, 1979, pp. 167-172.
[8] A. Miyake, "Mathematical aspects of optimal linear discriminant functions," in *Proc. Comput. Software Appl. Conf.*, Chicago, IL, Nov. 6-8, 1979, pp. 161-166.
[9] H. Mengert, "Solution of linear inequalities," *IEEE Trans. Comput.*, vol. C-19, no. 2, pp. 124-131, 1970.
[10] F. W. Smith, "Pattern classifier design by linear programming," *IEEE Trans. Comput.*, vol. C-17, no. 4, pp. 367-372, 1968.
[11] R. G. Grinold, "Comment on 'Pattern classifier design by linear programming,'" *IEEE Trans. Comput.*, vol. C-18, no. 4, pp. 378-379, 1969.
[12] E. W. Cheney, *Introduction to Approximation Theory*. New York: McGraw-Hill, 1966.
[13] V. Klee, "What is a convex set," *Amer. Math. Mon.*, vol. 78, pp. 617-631, June-July, 1971.
[14] R. T. Rockafellar, *Convex Analysis*. Princeton, NJ: Princeton Univ. Press, 1970.
[15] T. M. Cover, "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition," *IEEE Trans. Electron. Comput.*, vol. EC-14, no. 6, pp. 326-334, 1965.
[16] J. K. Bryan and D. L. Tebbe, "Generation of multivariate Gaussian data," Dep. Elec. Eng., Univ. Missouri, Colombia, Tech. Rep., Apr. 1970.
[17] G. N. Wassel, "Training a linear classifier to optimize the error probability," Ph.D. dissertation, Univ. California, Irvine, Dec. 1972.
[18] H. Do-Tu and M. Installe, "Learning algorithms for nonparametric solution to the minimum-error classification problem," *IEEE Trans. Comput.*, vol. C-27, pp. 648-659, 1978.

D. C. Clark received the B.S. degree in mathematics from the California Institute of Technology, Pasadena, in 1963, the Ph.D. degree in mathematics from Stanford University, Stanford, CA, in 1968, and the M.S. degree in computer science from the University of Tennessee, Knoxville, in 1980.

He has taught in the Departments of Mathematics at Rutgers University, Newark, NJ, and the University of Puerto Rico, Mayaguez. Currently, he is with the Los Angeles office of Pattern Analysis and Recognition Corporation.

R. C. Gonzalez (S'65-M'70) was born in Havana, Cuba, on August 26, 1942. He received the B.S.E.E. degree from the University of Miami, Coral Gables, FL, in 1965 and the M.E. and Ph.D. degrees in electrical engineering from the University of Florida, Gainesville, in 1967 and 1970, respectively.

He has been affiliated with the GT&E Corporation, the Center for Information Research at the University of Florida, NASA, and is presently IBM Professor of Electrical Engineering and Computer Science at the University of Tennessee, Knoxville. He is a frequent consultant to industry and government in the areas of pattern recognition, image processing, and machine learning. He received the 1978 UTK Chancellor's Research Scholar Award, the 1980 Magnavox Engineering Professor Award, and the 1980 M. E. Brooks Distinguished Professor Award for his work in these fields. He is coauthor of the books *Pattern Recognition Principles*, *Digital Image Processing*, and *Syntactic Pattern Recognition: An Introduction*, all published by Addison-Wesley. He is an Associate Editor for the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS and the *International Journal of Computer and Information Sciences*.

Dr. Gonzalez is a member of several professional and honorary societies, including Tau Beta Pi, Phi Kappa Phi, Eta Kappa Nu, and Sigma Xi.

SECTION VI

MOMENTS OF THE INTERCLASS MAHALANOBIS DISTANCE

IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, VOL. SMC-13, NO. 6, NOVEMBER/DECEMBER 1983

Moments of the Interclass Mahalanobis Distance

R. C. GONZALEZ, SENIOR MEMBER, IEEE, AND C. G. WAGNER

Abstract—It is shown that the moments of the interclass Mahalanobis distance between elements of two d-variate Gaussian populations can be

Manuscript received May 22, 1983. This work was supported by the U.S. Department of Energy under contract with the Union Carbide Corporation, and in part by the Office of Naval Research, Arlington, VA.

R. C. Gonzalez is with the Electrical Engineering Department, the University of Tennessee, Knoxville, TN 37996.

C. G. Wagner is with the Department of Mathematics, the University of Tennessee, Knoxville, TN 37996.

0018-9472/83/1100-1135$01.00

expressed in a simple polynomial form. The *n*th moment is expressible as a polynomial of order *n* whose variable depends on the mean vectors and eigenvalues of the covariance matrices. A closed-form solution is given for computing the coefficients of the polynomial expressions.

I. Introduction

Pattern recognition and image processing techniques based on the Mahalanobis distance have found wide applicability, ranging from nuclear reactor surveillance and automated analysis of image texture data to discrimination problems in biomedical observations [1], [2], [3].

The importance of the Mahalanobis distance classifier lies in the fact that, under a Gaussian assumption, it is an optimal discriminant in the Bayes sense [4]. The estimation of the probability density function (pdf) of the interclass Mahalanobis distance has been a topic of active interest for a number of years because of its direct relation to the probability of error of Bayes' classifier [5]. For Gaussian data with equal covariance matrices, the solution of this problem is straightforward [6]. When the covariance matrices are not equal, however, the problem becomes considerably more complicated, requiring the use of numerical integration techniques for computing the pdf [7].

In many applications (e.g., cluster seeking, texture analysis, and measuring spatial stationarity of multivariate data) it is often of interest to compute the moments of the interclass Mahalanobis distance without having to estimate its underlying pdf as an intermediate step. It is shown in this paper that these moments can be expressed directly in terms of a polynomial whose coefficients are given by a straightforward closed-form expression. The relative simplicity of these results has important implications in terms of implementation in a digital computer.

II. Background

Consider two d-dimensional Gaussian vector populations $\{x\}$ and $\{y\}$ with mean vectors and covariance matrices m_x, m_y, C_x, and C_y, respectively. The *intraclass Mahalanobis distance*[1] between any member of $\{x\}$ and m_x is given by the familiar equation [1]

$$R(x, m_x) = (x - m_x)^T C_x^{-1} (x - m_x), \tag{1}$$

and, similarly,

$$R(y, m_y) = (y - m_y)^T C_y^{-1} (y - m_y), \tag{2}$$

where "T" indicates the transpose.

As indicated in the previous section, (1) and (2) have been applied extensively in pattern recognition. In this paper, we are interested in characterizing the *interclass Mahalanobis distance* between members of x and the mean m_y, which is given by

$$R(x, m_y) = (x - m_y)^T C_y^{-1} (x - m_y) \tag{3}$$

and similarly,

$$R(y, m_x) = (y - m_x)^T C_x^{-1} (y - m_x). \tag{4}$$

For any nonsingular, real transformation matrix A it is easily shown that if

$$r = Ax \tag{5}$$

and

$$s = Ay, \tag{6}$$

then r and s are Gaussian random variables with mean vectors

$$m_r = Am_x \tag{7}$$

$$m_s = Am_y \tag{8}$$

and covariance matrices

$$C_r = AC_xA^T \tag{9}$$

$$C_s = AC_yA^T. \tag{10}$$

It is also easily shown that

$$R(r, m_s) = R(x, m_y) \tag{11}$$

and

$$R(s, m_r) = R(y, m_x). \tag{12}$$

Furthermore, as described in [6] and [16], the transformation matrix A can be chosen so that

$$C_r = AC_xA^T = I \tag{13}$$

and

$$C_s = AC_yA^T = D, \tag{14}$$

where I is the identity matrix and D is a diagonal matrix with elements $\gamma(i)$, $i = 1, 2, \cdots, d$, along the main diagonal. The elements $\gamma(i)$ are the eigenvalues of $C_x^{-1}C_y$. From (13), it is noted that the elements of r are uncorrelated which, in view of our Gaussian assumption, implies statistical independence. The same holds true for the elements of s.

Using (3), (11), and (14), it follows that

$$\begin{aligned} R(x, m_y) &= R(r, m_s) \\ &= (r - m_s)^T D^{-1} (r - m_s) \\ &= \sum_{i=1}^{d} (r_i - m_{si})^2 \gamma^{-1}(i), \end{aligned} \tag{15}$$

where r_i and m_{si}, $i = 1, 2, \cdots, d$, are the components of vectors r and m_s, respectively. Since r is a Gaussian random vector and $C_r = I$, we have that the variable $z_i = (r_i - m_{si})$ is Gaussian with mean $(m_{ri} - m_{si})$ and unit variance. It then follows [9] that

$$w_i = z_i^2 = (r_i - m_{si})^2 \tag{16}$$

is a noncentral chi-square variable with density

$$p(w_i) = e^{-\lambda_i} \sum_{k=0}^{\infty} \frac{\lambda_i^k w_i^{(1+2k)/2} e^{-w_i/2}}{k! 2^{(1+2k)/2} \Gamma\left(\frac{1+2k}{2}\right)} \tag{17}$$

and moment generating function

$$\phi_{w_i}(t) = e^{-\lambda_i} \sum_{k=0}^{\infty} \frac{\lambda_i^k}{k!} (1 - 2t)^{-(1+2k)/2}, \tag{18}$$

where

$$\lambda_i = \frac{1}{2}(m_{ri} - m_{si})^2. \tag{19}$$

Since r_i, $i = 1, 2, \cdots, d$, are statistically independent, it follows that the w_i defined in (16) are also statistically independent.

A similar development can be carried out for $R(y, m_x)$:

$$\begin{aligned} R(y, m_x) &= R(s, m_r) \\ &= (s - m_r)^T I^{-1} (s - m_r) \\ &= \sum_{i=1}^{d} (s_i - m_{ri})^2. \end{aligned} \tag{20}$$

The variable $z_i = (s_i - m_{ri})/\sqrt{\gamma(i)}$ is Gaussian with mean $(m_{si} - m_{ri})/\sqrt{\gamma(i)}$ and unit variance. As above, the variable

$$w_i = z_i^2 = \frac{1}{\gamma(i)}(s_i - m_{ri})^2 \tag{21}$$

[1] This is in reality a *squared* distance. However, it has become customary to refer to this measure simply as the *Mahalanobis distance*.

has the density and moment generating function given in (17) and (18), but λ_i is now given by

$$\lambda_i = \frac{1}{2\gamma(i)}(m_{si} - m_{ri})^2. \tag{22}$$

III. Moments of the Interclass Mahalanobis Distance.

It is shown in this section that the moments about zero of w_i can be expressed in terms of λ_i. Once these moments have been obtained, they will be used to obtain the moments of the interclass Mahalanobis distance.

A. Moments about Zero of w_i

We begin the development of noting that the nth moment about zero of w_i is given by

$$\alpha_n(w_i) = E\{w_i^n\} = \phi_{w_i}^{[n]}(0) \tag{23}$$

where $\phi_{w_i}^{[n]}(0)$ is the nth derivative of (18) with respect to t, evaluated at $t = 0$[10]. Evaluating (23) with the moment generating function given in (18) leads to the following theorem.

Theorem 1: Let $\alpha_n(w_i)$ denote the nth moment about zero of w_i. Then

$$\alpha_n(w_i) = \sum_{r=0}^{n} c(n,r)\lambda_i^r \tag{24}$$

where

$$c(n,r) = 2^r\binom{n}{r}\prod_{j=r}^{n-1}(2j+1) \tag{25}$$

for all $n \geq 1$ and $0 \leq r \leq n-1$, and

$$c(n,n) = 2^n \tag{26}$$

for all $n \geq 0$.

Proof: From (18) and (23),

$$\alpha_n(w_i) = \phi_{w_i}^{[n]}(0)$$

$$= e^{-\lambda_i}\sum_{k=0}^{\infty}\frac{\lambda_i^k}{k!}\frac{\partial^n}{\partial t^n}(1-2t)^{-(1+2k)/2}\Big|t=0 \tag{27}$$

$$= e^{-\lambda_i}\sum_{k=0}^{\infty}\frac{\lambda_i^k}{k!}\prod_{s=1}^{n}(2k+2s-1) \tag{28}$$

$$= \sum_{j=0}^{\infty}\frac{(-1)^j\lambda_i^j}{j!}\sum_{k=0}^{\infty}\frac{\lambda_i^k}{k!}\prod_{s=1}^{n}(2k+2s-1) \tag{29}$$

$$= \sum_{r=0}^{\infty}\frac{\lambda_i^r}{r!}\sum_{m=0}^{r}(-1)^{r-m}\binom{r}{m}\prod_{s=1}^{n}(2m+2s-1), \tag{30}$$

where (30) follows from (29) by the standard rule for multiplying Taylor series.

By a well-known formula from the calculus of finite differences [15]

$$b(n,r) = \sum_{m=0}^{r}(-1)^{r-m}\binom{r}{m}\prod_{s=1}^{n}(2m+2s-1)$$

$$= \Delta^r\prod_{s=1}^{n}(2x+2s-1)\Big|_{x=0} \tag{31}$$

where $\Delta^0 f(x) = f(x)$, $\Delta^1 f(x) = \Delta f(x) = f(x+1) - f(x)$, and

$$\Delta^r f(x) = \Delta(\Delta^{r-1}f(x)) \tag{32}$$

$$= \sum_{k=0}^{r}(-1)^{r-k}\binom{r}{k}f(x+k), \tag{33}$$

for $r \geq 2$.

Since $\prod_{s=1}^{n}(2x+2s-1)$ is a polynomial in x of degree n, it follows that $b(n,r) = 0$ if $r > n$, and hence from (30) that

$$\alpha_n(w_i) = \sum_{r=0}^{n} c(n,r)\lambda_i^r, \tag{34}$$

where

$$c(n,r) = b(n,r)/r!. \tag{35}$$

Now

$$\prod_{s=1}^{n}(2x+2s-1) = 2^n\left(\left(x+\frac{1}{2}\right)\left(x+\frac{3}{2}\right)\cdots\left(x+\frac{2n-1}{2}\right)\right) \tag{36}$$

$$= 2^n u(u-1)\cdots(u-n+1) \tag{37}$$

where

$$u = x + (2n-1)/2. \tag{38}$$

It follows easily by induction from (33) that

$$\Delta^r 2^n u(u-1)\cdots(u-n+1) = 2^n n(n-1)\cdots$$
$$\cdot(n-r+1)u(u-1)\cdots(u-n+r+1) \tag{39}$$

when $r < n$, and that

$$\Delta^n 2^n u(u-1)\cdots(u-n+1) = 2^n n! \tag{40}$$

Hence, (31), (37), (38), and (39) yield

$$b(n,r) = 2^r n(n-1)\cdots(n-r+1)\prod_{j=r}^{n-1}(2j+1) \tag{41}$$

if $n \geq 1$ and $0 \leq r < n$, and (31), (37), (38), and (40) yield

$$b(n,n) = 2^n n! \tag{42}$$

Dividing (41) by $r!$ and (42) by $n!$ yields (25) and (26), as desired.

In particular, with $c(0,0) = 1$,

$$c(n,0) = \prod_{j=0}^{n-1}(2j+1) = (2n-1)c(n-1,0) \tag{43}$$

for $n \geq 1$, and

$$c(n,r) = \frac{2(n-r+1)}{r(2r-1)}c(n,r-1) \tag{44}$$

for $n \geq 1$ and $1 \leq r \leq n$. The recurrence relations (43) and (44) enable one to generate the triangular array of numbers $c(n,r)$ with ease. Hence one may easily calculate the polynomials $\alpha_n(w_i)$, which are listed below for $0 \leq n \leq 5$:

$$\alpha_0(w_i) = 1$$
$$\alpha_1(w_i) = 1 + 2\lambda_i$$
$$\alpha_2(w_i) = 3 + 12\lambda_i + 4\lambda_i^2$$
$$\alpha_3(w_i) = 15 + 90\lambda_i + 60\lambda_i^2 + 8\lambda_i^3$$
$$\alpha_4(w_i) = 105 + 840\lambda_i + 840\lambda_i^2 + 224\lambda_i^3 + 16\lambda_i^4$$
$$\alpha_5(w_i) = 945 + 9450\lambda_i + 12600\lambda_i^2 + 5040\lambda_i^3 + 720\lambda_i^4 + 32\lambda_i^5. \tag{45}$$

B. Moments about Zero of R

From (15) and (16),

$$R(x, m_y) = \sum_{i=1}^{d} w_i \gamma^{-1}(i). \tag{46}$$

The nth moment about zero of R is then

$$\alpha_n[R(x, m_y)] = E\{R^n(x, m_y)\} = E\left\{\left[\sum_{i=1}^{d} w_i/\gamma(i)\right]^n\right\}. \tag{47}$$

The coefficients of a sum raised to the nth power are given by the multinomial theorem [13]; that is,

$$\alpha_n[R(x, m_y)] = E\left\{\sum \frac{n!}{e_1!e_2!\cdots e_d!} \prod_{i=1}^{d} [w_i/\gamma(i)]^{e_i}\right\}, \tag{48}$$

where the summation is taken over all nonnegative values of $e_1, e_2, \cdots, e_d$ such that $e_1 + e_2 + \cdots + e_d = n$.

In view of the independence of the w_i's, it follows that

$$\alpha_n[R(x, m_y)] = \sum \left(\frac{n!}{e_1!e_2!\cdots e_d!}\right) \prod_{i=1}^{d} [\alpha_{e_i}(w_i)/(\gamma(i))^{e_i}] \tag{49}$$

where the $\alpha_{e_i}(w_i)$, $i = 1, 2, \cdots, d$, are given by (24), using values of λ_i given by (19).

Since

$$R(y, m_x) = \sum_{i=1}^{d} (s_i - m_{ri})^2 = \sum_{i=1}^{d} w_i \gamma(i), \tag{50}$$

where w_i is given by (21), it follows using a similar development that

$$\alpha_n[R(y, m_x)] = \sum \left(\frac{n!}{e_1!e_2!\cdots e_d!}\right) \prod_{i=1}^{d} [\alpha_{e_i}(w_i)(\gamma(i))^{e_i}], \tag{51}$$

where the $\alpha_{e_i}(w_i)$, $i = 1, 2, \cdots, d$, are given by (24) using values of λ_i as given in (22).

IV. Special Cases

In this section we consider special cases involving various arrangements of mean vectors and covariance matrices of two pattern populations.

Equal Covariance Matrices

When $C_x = C_y = C$, it follows from (13) and (14) that $C_r = C_s = I$. Consequently, $\gamma(i) = 1$, $i = 1, 2, \cdots, d$, and both forms of λ_i ((19) and (2.22)) become the same. This leads to equal moments via (49) and (51).

Equal Mean Vectors

When $m_x = m_y$, it follows from (7) and (8) that $m_r = m_s$ and, consequently, $\lambda_i = 0$ in (19) and (22). Then from (24) and (25)

$$\alpha_n(w_i) = c(n, 0) = \prod_{s=1}^{n} (2s - 1) \tag{52}$$

for both populations. Substitution of (52) into (49) and (51) yields

$$\alpha_n[R(x, m_y)] = \sum \left(\frac{n!}{e_1!e_2!\cdots e_d!}\right) \prod_{i=1}^{d} [c(e_i, 0)/(\gamma(i))^{e_i}] \tag{53}$$

and

$$\alpha_n[R(y, m_x)] = \sum \left(\frac{n!}{e_1!e_2!\cdots e_d!}\right) \prod_{i=1}^{d} [(c(e_i, 0)\gamma(i)^{e_i}] \tag{54}$$

Equal Mean Vectors and Covariance Matrices (Intraclass Mahalanobis Distance)

When $m_x = m_y = m$ and $C_x = C_y = C$, we have only one population and the problem reduces to computing the moments of the intraclass Mahalanobis distance. It follows from (52) and (53) and the fact that each $\gamma(i) = 1$ (see the remarks above on equal covariance matrices) that

$$\alpha_n[R(x, m_x)] = \sum \frac{n!}{e_1!, e_2!, \cdots, e_d!} \cdot \prod_{i=1}^{e_1} (2i - 1) \cdots \prod_{i=1}^{e_d} (2i - 1) = \prod_{j=0}^{n-1} (d + 2j), \tag{55}$$

where the summation in (54) is over all $e_i \geq 0$ such that $e_1 + e_2 + \cdots + e_d = n$, and (55) follows from (54) by the following argument. By the extended binomial theorem

$$(1 - 2x)^{-1/2} = \sum_{e=0}^{\infty} \binom{-1/2}{e} (-2x)^e = \sum_{e=0}^{\infty} \left(\prod_{e=0}^{e} (2i - 1)\right) \frac{x^e}{e!}. \tag{56}$$

Raising (56) to the dth power yields

$$(1 - 2x)^{-d/2} = \sum_{n=0}^{\infty} \left(\sum \left(\prod_{i=1}^{e_1} (2i - 1) \cdots \prod_{i=1}^{e_d} (2i - 1)\right) \Big/ e_1! \cdots e_d!\right) x^n. \tag{57}$$

On the other hand, the extended binomial theorem yields

$$(1 - 2x)^{-d/2} = \sum_{n=0}^{\infty} \binom{-d/2}{n} (-2x)^n = \sum_{n=0}^{\infty} \frac{(d)(d + 2)\cdots(d + 2n - 2)}{n!} x^n. \tag{58}$$

Comparison of the coefficients of x^n in (57) and (58) yields the desired result.

One observes from (55) that the moment in question depends only on the order of the moment and the dimension of the pattern vectors.

V. Conclusion

The expressions given in (24), (25), (26), (49), and (51) lead to a straightforward algorithm for computing the moments about zero of the interclass Mahalanobis distance. If desired, the central moments can be obtained from these results by means of a well-known transformation [10].

The importance of these results is that they allow direct computation of the moments without having to resort to the intermediate step of obtaining the pdf which, as indicated in Section I, is not a trivial problem in the case of unequal covariance matrices.

The expressions for the moments were considerably simplified in the special cases discussed in Section IV. In particular, the intraclass Mahalanobis distance was shown to lead to expressions which depend only on the order of the moments and the dimension of the vector populations.

Acknowledgment

The authors express their appreciation to Professor Balram Rajput, Department of Mathematics, University of Tennessee for many helpful discussions.

References

[1] R. C. Gonzalez and L. C. Howington, "Machine recognition of abnormal behavior in nuclear reactors," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-7, no. 10, pp. 717–728.

[2] S. Y. Hus, "The Mahalanobis classifier with the generalized inverse approach for automated analysis of imagery texture data," *Comput. Graph. Image Proc.*, vol. 9, no. 2, pp. 117–134, 1979.

[3] R. E. Pogue, "Some investigations of multivariate discrimination procedures with applications to diagnosis clinical electrocardiography," Ph.D. dissertation, Univ. Minnesota, Minneapolis, 1966.

[4] J. T. Tou and R. C. Gonzalez, *Pattern Recognition Principles*. Reading, MA: Addison-Wesley, 1974.

[5] G. T. Toussaint, "Bibliography on estimation of misclassification," *IEEE Trans. Inform. Theory*, vol. IT-20, no. 4, pp. 472–479, 1974.

[6] T. W. Anderson, *An Introduction to Multivariate Statistical Analysis*. New York: Wiley, 1962.

[7] K. Fukunaga and T. F. Krile, "Calculation of Bayes' recognition error for two multivariate gaussian distributions," *IEEE Trans. Comput.*, vol. C-18, no. 3, pp. 220–229, 1969.

[8] R. Bellman, *Introduction to Matrix Analysis*. New York: McGraw-Hill, 1970.

[9] F. A. Graybill, *An Introduction to Linear Statistical Models*, Vol. 1. New York: McGraw-Hill, 1961.

[10] H. Cramer, *Mathematical Methods of Statistics*. Princeton, NJ: Princeton University, 1946.

[11] C. Jordan, *Calculus of Finite Differences*. New York: Chelsea Publishing, 1947.

[12] C. Berge, *Principles of Combinatorics*. New York: Academic, 1971.

[13] G. Berman and K. D. Fryer, *Introduction to Combinatorics*. New York: Academic, 1972.

[14] B. V. Gnedenko, *The Theory of Probability*. New York: Chelsea Publishing, 1967.

[15] D. M. Young and R. T. Gregory, *Survey of Numerical Mathematics*, Vol. I. Reading, MA: Addison-Wesley, 1972.

[16] K. Fukunaga, *Introduction to Statistical Pattern Recognition*. New York: Academic, 1972.

0018-9472/83/1100-1139$01.00

SECTION VII

SEMI-INVARIANTS OF THE INTERCLASS MAHALANOBIS DISTANCE

Semi-invariants of the Interclass Mahalanobis Distance

R. C. GONZALEZ, FELLOW, IEEE, AND C. G. WAGNER

Abstract—A new closed-form expression for the semi-invariants of the interclass Mahalanobis distance is derived. Typically, in the analysis of two multivariate Gaussian populations with different covariance matrices, simultaneous diagonalization of these matrices is required. The semi-invariants are given directly in terms of the mean vectors and inverse covariance matrices by the results established in this correspondence. In addition, a new iterative algorithm is derived for computing the moments of the interclass Mahalanobis distance from the semi-invariants.

Manuscript received November 11, 1982. This work was supported in part by the U.S. Department of Energy under contract with the Union Carbide Corp., and by the Office of Naval Research, Arlington, Virginia.

R. C. Gonzalez is with the Electrical Engineering Department, University of Tennessee, Knoxville, TN 37996.

C. G. Wagner is with the Department of Mathematics, University of Tennessee, Knoxville, TN 37996.

0018-9472/84/0500-0534$01.00

I. Introduction

Pattern recognition and image processing techniques based on the Mahalanobis distance have found wide applicability, ranging from nuclear reactor surveillance and automated analysis of image texture data to discrimination problems in biomedical observations [1], [2], [3].

The importance of the Mahalanobis distance classifier lies in the fact that under a Gaussian assumption it is an optimal discriminant in the Bayes sense [4]. The estimation of the probability density function (pdf) of the interclass Mahalanobis distance has been a topic of active interest for a number of years because of its direct relation to the probability of error of the Bayes classifier [5]. For Gaussian data with equal covariance matrices, the solution of this problem is straightforward [6]. When the covariance matrices are not equal, however, the problem becomes considerably more complicated, requiring the use of numerical integration techniques for computing the pdf [7].

In many applications (e.g., cluster seeking, texture analysis, and measuring spatial stationarity of multivariate data) it is often of interest to compute descriptors based on the interclass Mahalanobis distance. Two such descriptors are the moments and semi-invariants. In an earlier paper, we established that the nth moment could be expressed as a polynomial of degree n and gave a closed-form solution for computing the coefficients [17]. This procedure, however, requires that the covariance matrices of the two populations be simultaneously diagonalized.

The present work deals with the derivation of a closed-form expression for the semi-invariants of the Mahalanobis distance. This expression involves the mean vectors and inverse covariance matrices directly and does not require the diagonalization of these matrices. It is well known that the moments and semi-invariants are related by expressions that, though theoretically simple, are quite inefficient in terms of computer implementation [11]. A new, iterative algorithm that is easily implementable in a digital computer is presented in Section IV for computing the moments from a given set of semi-invariants.

II. Background

Consider two d-dimensional Gaussian vector populations $\{x\}$ and $\{y\}$ with mean vectors and covariance matrices m_x, m_y, C_x, and C_y, respectively. The *intraclass Mahalanobis distance*[1] between any member of $\{x\}$ and m_x is given by the familiar equation [1]

$$R(x,m_x) = (x-m_x)^T C_x^{-1}(x-m_x) \tag{1}$$

and similarly,

$$R(y,m_y) = (y-m_y)^T C_x^{-1}(y-m_y), \tag{2}$$

where T indicates the transpose.

As indicated in the previous section, (1) and (2) have been applied extensively in pattern recognition. In this work, we are interested in characterizing the *interclass Mahalanobis distance* between members of x and the mean m_y, which is given by

$$R(x,m_y) = (x-m_y)^T C_y^{-1}(x-m_y) \tag{3}$$

and similarly,

$$R(y,m_x) = (y-m_x)^T C_x^{-1}(y-m_x). \tag{4}$$

For any nonsingular, real transformation matrix A, it is easily shown that if

$$r = Ax \tag{5}$$

and

$$s = Ay \tag{6}$$

then r and s are Gaussian random variables with mean vectors

$$m_r = Am_x \tag{7}$$

$$m_s = Am_y \tag{8}$$

and covariance matrices

$$C_r = AC_xA^T \tag{9}$$

$$C_s = AC_yA^T. \tag{10}$$

It is also easily shown that

$$R(r,m_s) = R(x,m_y) \tag{11}$$

and

$$R(s,m_r) = R(y,m_x). \tag{12}$$

Furthermore, as described in [6] and [16], the transformation matrix A can be chosen so that

$$C_r = AC_xA^T = I \tag{13}$$

and

$$C_s = AC_yA^T = D \tag{14}$$

where I is the identity matrix and D is a diagonal matrix with elements $\gamma(i)$, $i = 1,2,\cdots,d$, along the main diagonal.[2] The elements $\gamma(i)$ are the eigenvalues of $C_x^{-1}C_y$. From (13), it is noted that the elements of r are uncorrelated which, in view of our Gaussian assumption, implies statistical independence. The same holds true for the elements of s.

Using (3), (11), and (14), it follows that

$$\begin{aligned} R(x,m_y) &= R(r,m_s) \\ &= (r-m_s)^T D^{-1}(r-m_s) \\ &= \sum_{i=1}^{d} (r_i - m_{si})^2 \gamma^{-1}(i), \end{aligned} \tag{15}$$

where r_i and m_{si}, $i = 1,2,\cdots,d$, are the components of vectors r and m_s, respectively. Since r is a Gaussian random vector and $C_r = I$, we have that the variable $z_i = (r_i - m_{si})$ is Gaussian with mean $(m_{ri} - m_{si})$ and unit variance. It then follows [9] that

$$w_i = z_i^2 = (r_i - m_{si})^2 \tag{16}$$

is a non-central chi-square variable with density

$$p(w_i) = e^{-\lambda_i} \sum_{k=0}^{\infty} \frac{\lambda_i^k w_i^{(1+2k)/2} e^{-w_i/2}}{k!\,2^{(1+2k)/2}\Gamma\left(\frac{1+2k}{2}\right)} \tag{17}$$

and moment generating function

$$\phi_{w_i}(t) = e^{-\lambda_i} \sum_{k=0}^{\infty} \frac{\lambda_i^k}{k!}(1-2t)^{-(1+2k)/2} \tag{18}$$

where

$$\lambda_i = \frac{1}{2}(m_{ri} - m_{si})^2. \tag{19}$$

Since r_i, $i = 1,2,\cdots,d$, are statistically independent, it follows that the w_i defined in (16) are also statistically independent.

[1] This is in reality a *squared* distance. However, it has become customary to refer to this measure simply as the *Mahalanobis distance*.

[2] Although diagonalization is not required in our final results, (13) and (14) are used in proving the theorem given in the next section.

A similar development can be carried out for $R(y, m_x)$:

$$R(y, m_x) = R(s, m_r) = (s - m_r)^T \Gamma^{-1}(s - m_r) = \sum_{i=1}^{d} (s_i - m_{ri})^2. \quad (20)$$

The variable $z_i = (s_i - m_{ri})/\sqrt{\gamma(i)}$ is Gaussian with mean $(m_{si} - m_{ri})/\sqrt{\gamma(i)}$ and unit variance. As above, the variable

$$w_i = z_i^2 = \frac{1}{\gamma(i)}(s_i - m_{ri})^2 \quad (21)$$

has the density and moment generating function given in (17) and (18), but λ_i is now given by

$$\lambda_i = \frac{1}{2\gamma(i)}(m_{si} - m_{ri})^2. \quad (22)$$

III. Semi-Invariants of the Interclass Mahalanobis Distance

One of the most important properties of the semi-invariants is that the nth semi-invariant of a sum of independent random variables is equal to the sum of the n semi-invariants of the individual variables [10], [11]. As will be seen in the following discussion, this property leads to a straightforward procedure for computing the semi-invariants of the interclass Mahalanobis distance, using only the original mean vectors and inverse covariance matrices of the given populations.

A. Semi-invariants of w_i.

We first obtain the semi-invariants of w_i and then extend the results to the general case involving R. The nth semi-invariant of a random variable w_i with moment generating function $\phi_{w_i}(t)$ is defined [14] as

$$X_n(w_i) = \frac{\partial^n}{\partial t^n}\left[\ln \phi_{w_i}(t)\right]_{t=0}. \quad (23)$$

Use of (18) in this definition leads to the following result involving λ_i.

Lemma: The nth semi-invariant of w_i is given by the expression

$$X_n(w_i) = 2^{n-1}(n-1)!\,[1 + 2n\lambda_i]. \quad (24)$$

Proof: From (18)

$$\phi_{w_i}(t) = e^{-\lambda_i}\sum_{k=0}^{\infty}\frac{\lambda_i^k}{k!}(1-2t)^{-(1+2k)/2} = e^{-\lambda_i}(1-2t)^{-1/2}\sum_{k=0}^{\infty}\frac{\lambda_i^k}{k!}(1-2t)^{-k} = e^{-\lambda_i}(1-2t)^{-1/2}\sum_{k=0}^{\infty}\frac{1}{k!}\left(\frac{\lambda_i}{1-2t}\right)^k.$$

The infinite summation is recognized as the Taylor expansion of $e^{\lambda_i/(1-2t)}$; therefore

$$\phi_{w_i}(t) = e^{-\lambda_i}(1-2t)^{-1/2}e^{\lambda_i/(1-2t)}. \quad (25)$$

Use of (23) yields

$$X_n(w_i) = \frac{\partial^n}{\partial t^n}\left[\ln e^{-\lambda_i} + \ln(1-2t)^{-1/2} + \frac{\lambda_i}{1-2t}\right]_{t=0} = \frac{\partial^n}{\partial t^n}\left[\ln(1-2t)^{-1/2}\right]_{t=0} + \frac{\partial^n}{\partial t^n}\left[\frac{\lambda_i}{1-2t}\right]_{t=0} = 2^{n-1}(n-1)! + 2^n n!\lambda_i = 2^{n-1}(n-1)!\,[1 + 2n\lambda_i].$$

This concludes the proof.

As an illustration, the first five semi-invariants of w_i are

$$X_1(w_i) = 1 + 2\lambda_i$$
$$X_2(w_i) = 2 + 8\lambda_i$$
$$X_3(w_i) = 8 + 48\lambda_i$$
$$X_4(w_i) = 48 + 384\lambda_i$$
$$X_5(w_i) = 384 + 3840\lambda_i. \quad (26)$$

B. Semi-invariants of R

The semi-invariants of the interclass Mahalanobis distance $R(x, m_y)$ are given by the following theorem.

Theorem: The nth semi-invariants of $R(x, m_y)$ and $R(y, m_x)$ are given, respectively, by

$$X_n[R(x, m_y)] = 2^{n-1}(n-1)!\left[\operatorname{tr}\left\{\left(C_y^{-1}C_x\right)^n\right\} + n(m_x - m_y)^T C_y^{-1}\left(C_x C_y^{-1}\right)^{n-1}(m_x - m_y)\right] \quad (27)$$

and

$$X_n[R(y, m_x)] = 2^{n-1}(n-1)!\left[\operatorname{tr}\left\{\left(C_x^{-1}C_y\right)^n\right\} + n(m_y - m_x)^T C_x^{-1}\left(C_y C_x^{-1}\right)^{n-1}(m_y - m_x)\right]. \quad (28)$$

Proof: From (15) and (16)

$$R(x, m_y) = \sum_{i=1}^{d} w_i \gamma^{-1}(i).$$

Since the nth semi-invariant of a sum of independent random variables is the sum of the semi-invariants, and $X_n[w_i\gamma^{-1}(i)] = X_n(w_i)\gamma^{-n}(i)$ [10], we have

$$X_n[R(x, m_y)] = \sum_{i=1}^{d} X_n(w_i)\gamma^{-n}(i). \quad (29)$$

Then, from the lemma in Section III-A,

$$X_n[R(x, m_y)] = \sum_{i=1}^{d} 2^{n-1}(n-1)!\,[1 + 2n\lambda_i]\gamma^{-n}(i) = 2^{n-1}(n-1)!\sum_{i=1}^{d}\gamma^{-n}(i) + 2^n n!\sum_{i=1}^{d}\lambda_i\gamma^{-n}(i). \quad (30)$$

Since $\gamma(i)$, $i = 1, 2, \cdots, d$ are the eigenvalues of $C_x^{-1}C_y$, the sum of the eigenvalues $\gamma^{-n}(i)$ for $i = 1, 2, \cdots, d$, is the trace of $(C_y^{-1}C_x)^n$. Modifying (30) by this observation, and using the definition of λ_i given in (19), yields

$$X_n[R(x, m_y)] = 2^{n-1}(n-1)!\operatorname{tr}\left\{\left(C_y^{-1}C_x\right)^n\right\} + 2^{n-1}n!\sum_{i=1}^{d}\frac{(m_{ri} - m_{si})^2}{\gamma^n(i)}. \quad (31)$$

The summation in (31) may be expressed as

$$\sum_{i=1}^{d}\frac{(m_{ri} - m_{si})^2}{\gamma^n(i)} = (m_r - m_s)^T(D^{-1})^n(m_r - m_s). \quad (32)$$

However, from (7) and (8),

$$(m_r - m_s) = A(m_x - m_y) \quad (33)$$

and, from (14),

$$(D^{-1})^n = \left[\left(AC_yA^T\right)^{-1}\right]^n$$
$$= \left[(A^T)^{-1}C_y^{-1}A^{-1}\right]^n.$$

Expansion of the right side of this equation yields

$$\left[(A^T)^{-1}C_y^{-1}A^{-1}\right]^n$$
$$= (A^T)^{-1}C_y^{-1}A^{-1}(A^T)^{-1}C_y^{-1}A^{-1}\cdots(A^T)^{-1}C_y^{-1}A^{-1}$$

But from (13), $A^{-1}(A^T)^{-1} = C_x$, so that

$$(D^{-1})^n = (A^T)^{-1}C_y^{-1}\left(C_xC_y^{-1}\right)^{n-1}A^{-1}. \quad (34)$$

By substituting (33) and (34) into (31) we obtain

$$\sum_{i=1}^{d}\frac{(m_{ri}-m_{si})^2}{\gamma^n(i)} = (m_x-m_y)^TC_y^{-1}\left(C_xC_y^{-1}\right)^{n-1}(m_x-m_y). \quad (35)$$

Finally, substituting this result into (31) completes the proof of (27).

The proof of (28) follows essentially the same line of reasoning, with the exception that it involves $\gamma^n(i)$ instead of $\gamma^{-n}(i)$, and the definitions of w_i and λ_i are different, as given in (21) and (22). From (20) and (21) and the distributive properties of the semi-invariants stated earlier,

$$X_n[R(y,m_x)] = \sum_{i=1}^{d}X_n(w_i)\gamma^n(i), \quad (36)$$

and, from the lemma in Section III-A,

$$X_n[R(y,m_x)] = \sum_{i=1}^{d}2^{n-1}(n-1)![1+2n\lambda_i]\gamma^n(i). \quad (37)$$

The validity of the trace portion of (28) follows directly from the fact that $\gamma^n(i)$ are eigenvalues of $(C_x^{-1}C_y)^n$. To prove the validity of the second term on the right side of (28), we note that

$$2^nn!\sum_{i=1}^{d}\lambda_i\gamma^n(i) = 2^{n-1}n!\sum_{i=1}^{d}(m_{si}-m_{ri})^2\gamma^{n-1}(i). \quad (38)$$

The summation term can be expressed as

$$\sum_{i=1}^{d}(m_{si}-m_{ri})^2\gamma^{n-1}(i) = (m_x-m_y)^TA^TD^{n-1}A(m_x-m_y). \quad (39)$$

Expansion of the matrix D^{n-1} (see (14)) gives

$$D^{n-1} = AC_yA^TAC_yA^T\cdots AC_yA^T. \quad (40)$$

However, from (13), $A^TA = C_x^{-1}$. Using this fact in (40) and (39) completes the proof.

It is important to note that the semi-invariants in (27) and (28) are given directly in terms of the original population parameters m_x, m_y, C_x, and C_y and, therefore, do not require computation of the transformation matrix A.

C. Special Cases

In this section we consider special cases involving various arrangements of mean vectors and covariance matrices of two populations.

Equal Covariance Matrices: For equal covariance matrices, $C_x = C_y = C$, it is easily shown that (27) and (28) reduce to

$$X_n[R(x,m_y)] = X_n[R(y,m_x)] = 2^{n-1}(n-1)!\cdot\left[d+n(m_x-m_y)^TC^{-1}(m_x-m_y)\right]. \quad (41)$$

Equal Mean Vectors: When $m_x = m_y$, we have from (27) and (28) that

$$X_n[R(x,m_y)] = 2^{n-1}(n-1)!\left[\mathrm{tr}\left\{\left(C_y^{-1}C_x\right)^n\right\}\right] \quad (42)$$

and

$$X_n[R(y,m_x)] = 2^{n-1}(n-1)!\left[\mathrm{tr}\left\{\left(C_x^{-1}C_y\right)^n\right\}\right]. \quad (43)$$

Equal Mean Vectors and Covariance Matrices (Intraclass Mahalanobis Distance): In this case we are considering the same Gaussian population and obtain the semi-invariants by letting $x = y$, $m_x = m_y$, $C_x = C_y$ in either (27) or (28). This results in the simple expression

$$X_n[R(x,m_x)] = 2^{n-1}(n-1)!d \quad (44)$$

which depends only on the order of the semi-invariant and the dimensionality of the vectors.

IV. Obtaining the Moments from the Semi-Invariants

The nth moment α_n of the interclass Mahalanobis distance can be obtained directly from the semi-invariants $X_1, X_2, \cdots, X_n$ by using the expression

$$\alpha_n = \sum\frac{n!}{a_1!a_2!\cdots a_n!}\prod_{r=1}^{n}[X_r/r!]^{a_r} \quad (45)$$

where the sum is taken over values of a_i such that $a_1 + 2a_2 + \cdots + na_n = n$ [11]. Equation (45) can be used to obtain the moments of either w_i or R, given the semi-invariants corresponding to one of these two variables [11]. A similar relationship exists for computing the semi-invariants given the moments [11].

As an illustration of the above relationship we have

$$\alpha_1 = X_1$$
$$\alpha_2 = X_2 + X_1^2$$
$$\alpha_3 = X_3 + 3X_1X_2 + X_1^3$$
$$\alpha_4 = X_4 + 4X_1X_3 + 3X_2^2 + 6X_1^2X_2 + X_1^4.$$

A direct implementation of (45) in a digital computer is inherently inefficient, involving, among other things, the determination of all n tuples of nonnegative integers $(a_1, \cdots, a_n)$ satisfying $a_1 + 2a_2 + \cdots na_n = n$. Fortunately, there is a very efficient recursive algorithm, described below, for computing these moments.

Let

$$Z_n = X_n/n!2^n \quad (46)$$

where X_n is given by (27) or (28), as determined by the relevant semi-invariant. Under this change of variables (45) becomes

$$\alpha_n = \sum\frac{n!}{a_1!\cdots a_n!}\prod_{r=1}^{n}(2^rZ_r)^{a_r}$$
$$= n!2^n\sum\prod_{r=1}^{n}(Z_r^{a_r}/a_r!) \quad (47)$$

taken over all n tuples of nonnegative integers $(a_1, \cdots, a_n)$ satisfying $a_1 + 2a_2 + \cdots + na_n = n$. If we define a rectangular array $\beta(n,k)$ by

$$\beta(n,k) = \sum\prod_{r=1}^{n}(Z_r^{a_r}/a_r!), \qquad n \geq 0, \qquad k \geq 1 \quad (48)$$

taken over all k tuples of nonnegative integers $(a_1, \cdots, a_k)$ satisfying $a_1 + 2a_2 + \cdots + ka_k = n$, then by (47)

$$\alpha_n = n!2^n\beta(n,n), \qquad n \geq 1, \quad (49)$$

so that the quantities α_n may easily be computed from the

TABLE I
$\beta(n,k)$: $0 \leq n \leq 3, 1 \leq k \leq 3$

n \ k	1	2	3
0	1	1	1
1	Z_1	Z_1	Z_1
2	$Z_1^2/2$	$Z_1^2/2 + Z_2$	$Z_1^2/2 + Z_2$
3	$Z_1^3/6$	$Z_1^3/6 + Z_1Z_2$	$Z_1^3/6 + Z_1Z_2 + Z_3$

elements lying just below the main diagonal (the main diagonal elements of $\beta(n,k)$ are $\beta(n-1,n)$, since $n \geq 0$ and $k \geq 1$) of the arrays $\beta(n,k)$.

It is clear from (48) that

$$\beta(0,k) = 1, \qquad k \geq 1 \tag{50}$$

that

$$\beta(n,1) = Z_1^n/n!, \qquad n \geq 0 \tag{51}$$

and that

$$\beta(n,k) = \beta(n,n), \qquad k > n > 0. \tag{52}$$

The remaining elements of the array $\beta(n,k)$ are generated by the recurrence relation

$$\beta(n,k) = \sum_{j=0}^{[n/k]} \beta(n-jk, k-1) Z_k^j/j! \tag{53}$$

where $[n/k]$ denotes the greatest integer $\leq n/k$. Hence the elements in column k of the array $\beta(n,k)$ are just linear combinations of certain elements in column $k-1$. Equation (53) is justified by observing that the nonnegative integral solutions of $a_1 + 2a_2 + \cdots + ka_k = n$ may be partitioned according to the possible values $j = 0, 1, \cdots, [n/k]$ of a_k, the terms in (53) corresponding to those $[n/k]+1$ possible values of a_k. Values of $\beta(n,k)$ for $0 \leq n \leq 3$ and $1 \leq k \leq 3$ are listed in Table I.

V. Conclusion

The expressions given in (27) and (28) provide a straight-forward solution to the problem of computing the semi-invariants of the interclass Mahalanobis distance. As indicated in Section I, the semi-invariants are useful descriptions of the underlying interclass distance pdf.

Although the semi-invariants do not in general have the familiar "physical" interpretation possessed by the moments (e.g., spread, skew, and curtosis), the distributive property of the semi-invariants resulted in a computational procedure involving only the mean vectors and inverse covariance matrices of two populations, without the need for the simultaneous diagonalization required to obtain the moments [17]. The algorithm given in Section IV provides a rather simple iterative technique for computing the moments once the semi-invariants have been obtained via (27) and (28).

The semi-invariants were considerably simplified in the special cases discussed in Section III-C. In particular, the semi-invariants of the intraclass Mahalanobis distance was shown to be dependent only on the order of the semi-invariants and on the dimensionality of the vector populations.

Acknowledgment

The authors express their appreciation to Professor Balram Rajput, Department of Mathematics, the University of Tennessee, for many helpful discussions.

References

[1] R. C. Gonzalez, and L. C. Howington, "Machine recognition of abnormal behavior in nuclear reactors," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-7, no. 10, pp. 717-728, 1977.
[2] S. Y. Hus, "The Mahalanobis classifier with the generalized inverse approach for automated analysis of imagery texture data," *Comput. Graph. Image Proc.*, vol. 9, no. 2, pp. 117-134, 1979.
[3] R. E. Pogue, "Some investigations of multivariate discrimination procedures with applications to diagnosis clinical electrocardiography," Ph.D. dissertation, Univ. Minnesota, Minneapolis, 1966.
[4] J. T. Tou, and R. C. Gonzalez, *Pattern Recognition Principles*. Reading, MA: Addison-Wesley, 1974.
[5] G. T. Toussaint, "Bibliography on estimation of misclassification," *IEEE Trans. Inform. Theory*, vol. IT-20, no. 4, pp. 472-479, 1974.
[6] T. W. Anderson, *An Introduction to Multivariate Statistical Analysis*. New York: Wiley, 1962.
[7] K. Fukunaga, and T. F. Krile, "Calculation of Bayes' recognition error for two multivariate Gaussian distributions," *IEEE Trans. Comput.*, vol. C-18, no. 3, pp. 220-229.
[8] R. Bellman, *Introduction to Matrix Analysis*, New York: McGraw-Hill, 1970.
[9] F. A. Graybill, *An Introduction to Linear Statistical Models*, vol. 1. New York: McGraw-Hill, 1961.
[10] H. Cramer, *Mathematical Methods of Statistics*. New Jersey: Princeton Univ. Press, 1946.
[11] C. Jordan, *Calculus of Finite Differences*. New York: Chelsea, 1947.
[12] C. Berge, *Principles of Combinatorics*. New York: Academic, 1971.
[13] G. Berman and K. D. Fryer, *Introduction to Combinatorics*. New York: Academic, 1972.
[14] B. V. Gnedenko, *The Theory of Probability*. New York: Chelsea 1967.
[15] D. M. Young and R. T. Gregory, *Survey of Numerical Mathematics*, vol. I, Reading, MA: Addison-Wesley, 1972.
[16] K. Fukunaga, *Introduction to Statistical Pattern Recognition*. New York: Academic, 1972.
[17] R. C. Gonzalez and C. G. Wagner, "Moments of the interclass Mahalanobis distance," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-13, no. 6, pp. 1135-1139, 1983.

0018-9472/84/0500-0538$01.00

END

FILMED

1-85

DTIC