Available online at www.sciencedirect.com
SCIENCE DIRECT®
Discrete Applied Mathematics 142 (2004) 189–205
DISCRETE APPLIED MATHEMATICS
www.elsevier.com/locate/dam

Sequential testing of complex systems: a review

Tonguç Ünlüyurt

Faculty of Engineering and Natural Sciences, Sabanci University, Orhanlı, Tuzla, 81474 Istanbul, Turkey

Received 9 January 2001; received in revised form 30 July 2002; accepted 12 August 2002

Abstract

We consider the problem of testing sequentially the components of a multi-component system in order to learn the state of the system, when the tests are costly. In this review paper, we describe and analyze a framework for the problem, results for certain classes of problems and related widespread applications, including distributed computing, artificial intelligence, manufacturing and telecommunications. We also report variations and possible extensions of the problem.

Keywords: Sequential testing; Boolean functions

1. Introduction

The problem of diagnosing a complex system through a series of tests of its components arises in many practical situations. In many of these cases, testing a component is costly (it may cost money, take time, or incur pain on the patient, etc.), and therefore it is important to determine a best diagnosing policy that minimizes the average cost of testing in the long run.

More precisely, we consider complex systems, consisting of several components, in which the state of the system (e.g. working or failing, healthy or unhealthy, etc.) depends on the states of its components. We shall assume that this dependency is known, either by an explicit functional description, or via an oracle. The procedure of diagnosing such a system consists of testing the components one-by-one, until the state of the whole system is correctly determined.

There are many variations for such a diagnosis problem. We shall consider one frequently arising type, in which it is assumed that the cost of testing a component as well as the (a priori) probability that it is working correctly, are known in advance. Furthermore, the components are assumed to work or fail independently of each other. Since the inspection of the system is usually repeated many times (the same system is inspected in various different situations; several equivalent systems are inspected; etc.) it is customary to consider the problem of finding a policy that minimizes the associated expected cost. This corresponds to the practical desire of minimizing the total cost in the long run.

Such problems arise in a wide area of applications, including telecommunications (testing reliability systems, connectivity of networks, etc. see e.g. [16]), manufacturing (testing machines before shipping, or testing for replacement in technical service centers, see e.g. [18]), design of screening procedures (see e.g. [26]), electrical engineering (wafer probe testing, see [10]), artificial intelligence (finding optimal derivation strategies in knowledge bases, see e.g. [29]; best-value or satisficing search algorithms, see e.g. [40,52,59,60]), medicine (testing incoming patients against the possibility of some rare but dangerous disease, see e.g. [17]), or even in quiz shows (choosing the right order of categories, see, e.g. [38]).

Since the applications are so widespread, often the researchers in one area have been unaware of the results that are obtained by researchers in another area. In this paper, first, we describe the mathematical framework for the model.

E-mail address: tonguc@sabanciuniv.edu (T. Ünlüyurt).

0166-218X/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2002.08.001

Then we bring together applications from different areas, issues and results. Finally, we describe some of the possible extensions and variations.

2. Problem definition

We shall consider *systems* composed of a set $N = \{u_1, u_2, \ldots, u_n\}$ of n components, each of which can be in one of the two states, *working* or *failing*. A *state vector* $\mathbf{x} = (x_1, \ldots, x_n)$ is a binary vector, corresponding to a particular state of the system by recording the state of each of the components, i.e. $x_i = 1$ if component u_i is working, and $x_i = 0$ otherwise, for $i = 1, \ldots, n$. The *system function* is the Boolean mapping $f : \mathbb{B}^n \mapsto \mathbb{B}$ characterizing the states in which the system is functioning, i.e.

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if the system is functioning, and} \\ 0 & \text{otherwise} \end{cases}$$

for every state vector $\mathbf{x} \in \mathbb{B}^n$. Such a Boolean mapping $f : \mathbb{B}^n \mapsto \mathbb{B}$ is called *monotone* if $f(\mathbf{x}) \geqslant f(\mathbf{y})$ whenever $\mathbf{x} \geqslant \mathbf{y}$. In other words, by fixing some of the failing components of a functioning monotone system, one cannot make it fail.

Let us denote a system by $\Gamma(N, f)$, where N is the set of its components and f is its system function. Given a system $\Gamma(N, f)$, we shall say that its ith component u_i is relevant (with respect to f) if there exists a state vector such that a change occurring only in u_i's state changes the state of the whole system. The system $\Gamma(N, f)$ is called *coherent* if its system function f is monotone, and all of its components are relevant with respect to f.

Sequential diagnosis is a procedure in which the components of a system are inspected, one by one, in order to find out the functionality of the system at the current (not yet known) state. We assume that, when inspecting a component, we can learn the correct state of that component. An *inspection strategy* $\mathbf{S}$ is a rule, which specifies, on the basis of the states of the already inspected components, which component is to be inspected next or stops by recognizing the correct value of the system function at the current state vector. Such an inspection strategy can naturally be represented as a binary decision tree.

Example 1. Let us consider a system consisting of 3 components $N = \{u_1, u_2, u_3\}$ and having

$$f(x_1, x_2, x_3) = x_1x_2 \vee x_2x_3 \tag{1}$$

as its system function. Then, Fig. 1 represents a possible inspection strategy, in which component u_1 is inspected first, and depending on whether it is working or not, components u_2 or u_3 are inspected next, etc. E.g., if the system happens to be in state $\mathbf{x} = (1, 1, 0)$, then this strategy learns that the system is functioning after inspecting components u_1 and u_2, whereas if the system's state is $\mathbf{y} = (0, 1, 0)$, then this strategy will stop after inspecting components u_1 and u_3, recognizing that the system does not function.

The main reason to terminate the inspection algorithm as early as possible, and not to check all components, is that inspection is usually costly. Let us denote by c_i the cost of inspecting component u_i, (i.e. learning whether or not u_i is working) for $i = 1, \ldots, n$, and let $\mathbf{c} = (c_1, \ldots, c_n)$.

Our main goal is to find a strategy for a given system, that minimizes the cost of inspection. This however, is not a well defined problem, yet, since for different states a different strategy would be optimal.

Example 2. Let us return to the system of Example 1. For the state $\mathbf{x} = (1, 1, 0)$ the strategy represented in Fig. 1 is optimal, since it checks only the first two components (and inspecting less than that would not be sufficient). However, if the system is in state $\mathbf{y} = (0, 0, 1)$, then this strategy inspects all components to learn that the system is not functioning, although, it would be enough to check only component u_2.

Clearly, the cost of inspection depends not only on our strategy, but also on the state the system is at. For this reason, let us assume that an a priori distribution of the states of the components is known. We shall assume in this paper that the component u_i works with probability p_i and fails with probability $q_i = 1 - p_i$, for $i = 1, \ldots, n$, and let $\mathbf{p} = (p_1, \ldots, p_n)$. We shall also assume that the components work or fail independently of each other. It could happen, more generally, that the cost of inspecting component u_i depends on the state the component is in, e.g. let us say that c_i^T is the cost of inspection when the component functions properly, and c_i^F is the cost of inspection when the component fails. One can see easily that in fact, when one is interested only in expected cost, this is not a more general case, and it can be reduced to the previous problem by introducing $c_i = p_i c_i^T + q_i c_i^F$ as a common inspection cost ($i = 1, 2, \ldots, n$).

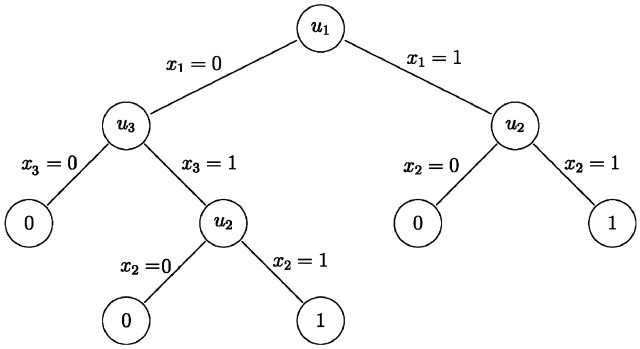

Fig. 1. An inspection strategy for the system of Example 1.

Given an instance of the sequential diagnosis problem, i.e. a system $\Gamma(N, f)$ and the vectors $\mathbf{c}$ and $\mathbf{p}$ of costs and probabilities, respectively, we shall consider the problem of finding an inspection strategy $\mathbf{S}$ of the given system which has the minimum expected cost.

When describing an inspection strategy, one could go further, in principle, and inspect other components, even if the functionality of the system can already be determined. Let us call a strategy *reasonable* if it stops as soon as it learns the functionality of the system. For instance, the strategy in Fig. 1 is a reasonable inspection strategy. Since it is also obvious that optimal strategies are reasonable, unless some of the components have zero inspection cost, we shall consider only reasonable strategies in the sequel, and will drop the word *reasonable*, whenever it will not cause ambiguity.

Since we assume that the function is given by an explicit functional description or via an oracle, and that the system is coherent, it is always easy to learn the functionality of the system, after inspecting a subset of the components, or to conclude that it is not yet determined. Namely, by approximating the current state from above by $\mathbf{x}^+$ where all uninspected components are assumed to work, and from below by $\mathbf{x}^-$, where all uninspected components are assumed to fail, we can claim that if $f(\mathbf{x}^-) = 1$ then the system is working, and if $f(\mathbf{x}^+) = 0$ then the system fails (regardless of the states of the uninspected components), and hence we can terminate the inspection. In the remaining case of $f(\mathbf{x}^-) = 0$ and $f(\mathbf{x}^+) = 1$ the system's state is not yet determined, and further components have to be inspected. Hence, computing the value of f (or executing the oracle) at two vectors suffices to recognize if the inspection can be terminated. Let us remark that for general, non-monotone systems the above is not true, and in fact the same recognition problem is NP-complete.

3. Classes of monotone system (Boolean) functions

Let us first recall some basic definitions and well known properties of monotone Boolean functions. For a given Boolean function f let us denote by $T(f)$ and $F(f)$ the sets of binary vectors at which f takes value 1 and 0, respectively. The vectors in $T(f)$ are called the *true points*, while the elements of $F(f)$ are called the *false points* of f.

Given a subset $S \subseteq \{1, 2, \ldots, n\}$, let us denote its characteristic vector by $\mathbf{1}^S$, i.e.

$$\mathbf{1}_j^S = \begin{cases} 1 & \text{if } j \in S, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

The dual of f denoted by f^d is again a monotone Boolean function for which

$$T(f^d) = \{\mathbf{1}^S \mid \mathbf{1}^S \in F(f)\}, \tag{3}$$

where $S = \{1, 2, \ldots, n\} \backslash S$.

We shall call a subset $I \subseteq \{1, 2, \ldots, n\}$ an *implicant* of the monotone Boolean function f, if $\mathbf{1}^I \in T(f)$. The subset I is called a *prime implicant* if it is an implicant, and no proper subset of it is an implicant.

Given a monotone Boolean function f, let us denote by

$$\mathbf{I}(f) = \{I_1, \ldots, I_s\} \tag{4}$$

the family of its prime implicants, and let

$$\mathbf{J}(f) = \{J_1, \ldots, J_t\} \tag{5}$$

be the family of the prime implicants of its dual, i.e. $\mathbf{J}(f)=\mathbf{I}(f^d)$. The binary vectors $\mathbf{1}^I$ for $I\in\mathbf{I}(f)$ are called *minimal true points* of f, while vectors of the form $\mathbf{0}^J$ for $J\in\mathbf{J}(f)$ are called *maximal false points* of f, where

$$\mathbf{0}_j^S=\begin{cases}0 & \text{if } j\in S,\\ 1 & \text{otherwise.}\end{cases} \tag{6}$$

Proposition 1. *A monotone Boolean function f and its dual f^d can be represented by the* (*unique minimal*) *positive disjunctive normal forms*:

$$f(\mathbf{x})=\bigvee_{I\in\mathbf{I}(f)}\left(\bigwedge_{j\in I}x_j\right)\quad \textit{and}\quad f^d(\mathbf{x})=\bigvee_{J\in\mathbf{J}(f)}\left(\bigwedge_{j\in J}x_j\right), \tag{7}$$

respectively.

Let us remark next that given a monotone Boolean function f by its list $\mathbf{I}(f)$ of prime implicants, it may be very difficult to obtain $\mathbf{J}(f)$, the list of its dual prime implicants. First of all, the size of $\mathbf{J}(f)$ can be exponentially larger than the size of $\mathbf{I}(f)$. Secondly, there is no known general algorithm to obtain $\mathbf{J}(f)$ from $\mathbf{I}(f)$ in polynomial time even in these sizes (see e.g. [4,19,20,42]). On the other hand, the algorithm of [20] is quasi-polynomial. There are also many important special cases where dualization is polynomial (see e.g. [19,5,31]).

We now describe and define certain classes of Monotone System (Boolean) Functions for which the *Sequential Diagnosis* problem is tractable. The application areas that they arise and the related results will be discussed in Section 5.

3.1. Simple series and parallel systems

Series and parallel systems are the simplest systems. A series system fails if any one of its components fails while a parallel system functions if any one of its components functions. If x_i is the variable associated with the ith component, the structure function of the series and parallel systems can be written (respectively) in the following way:

$$f(\mathbf{x})=x_1\wedge x_2\wedge\cdots\wedge x_n\quad\text{and}\quad f(\mathbf{x})=x_1\vee x_2\vee\cdots\vee x_n.$$

In spite of their simplicity, series and parallel systems arise in numerous applications. In addition, all other known solvable cases are generalizations of series and parallel systems. Next, we will describe these more general classes of systems.

3.2. k-out-of-n and Double Regular Systems

Let us first define the well known class of threshold functions, that arise in a lot of applications including electrical circuits. A Boolean mapping $f:\mathbb{B}^n\mapsto\mathbb{B}$ is called *threshold* if there exists non-negative weights $w_1,w_2,\ldots,w_n$ and a constant t such that

$$f(\mathbf{x})=\begin{cases}1 & \sum_{i=1}^n w_ix_i\geqslant t,\\ 0 & \text{otherwise.}\end{cases}$$

A k-out-of-n system functions if at least k of its n components function. In particular a series system is an n-out-of-n system and a parallel system is a 1-out-of-n system.

Example 3. One can easily see that a k-out-of-n function is a threshold function for which all weights are equal (namely, $w_i=t/k$, $i=1,\ldots,n$). Yet, not all threshold functions are k-out-of-n. One can verify that e.g. $f(\mathbf{x})=x_1x_2\vee x_1x_3$ is such a threshold function (with $w_1=2$, $w_2=1$, $w_3=1$ and $t=3$).

Let us define *Regular Systems* before describing *Double Regular Systems*. Given a monotone Boolean function $f:\mathbb{B}^n\mapsto\mathbb{B}$, we shall say that the variable x_i is *stronger* than x_j ($i\neq j$) with respect to f, and write $x_i\geqslant_f x_j$, if for all binary vectors $\mathbf{x}\in\mathbb{B}^n$ with $x_i=x_j=0$ we have $f(\mathbf{x}\vee\mathbf{e}_i)\geqslant f(\mathbf{x}\vee\mathbf{e}_j)$, where $\mathbf{e}_i$ denotes the unit ith unit vector. If $x_i\geqslant_f x_j$ and $x_j\geqslant_f x_i$ we shall say that x_i and x_j are *equivalent with respect to* f and write $x_i\approx_f x_j$. It is well-known that this strength relation is a pre-order on the set of variables for any monotone Boolean function f (see, e.g. [50,66]). Given a permutation ρ of the indices $\{1,2,\ldots,n\}$, a Boolean function f is called *ρ-regular*, if its strength pre-order is consistent with ρ, i.e. if $x_{\rho_1}\geqslant_f x_{\rho_2}\geqslant_f\cdots\geqslant_f x_{\rho_n}$. A function is simply said to be *regular*, if it is ρ-regular for some permutation ρ. Let us note that if f is ρ-regular so is f^d.

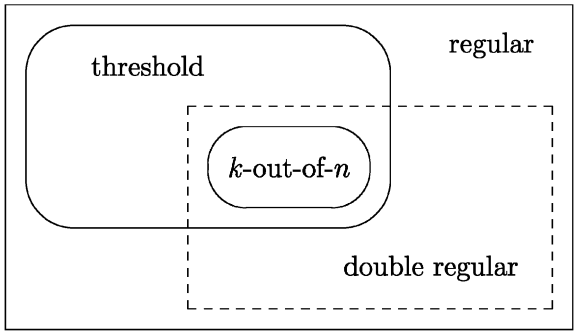

Fig. 2. Relations among the considered classes of system functions.

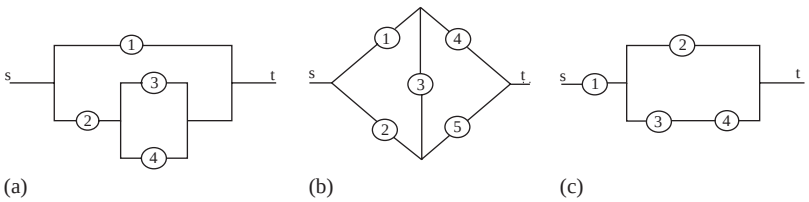

Fig. 3. Examples for an SPS (a), a non-series-parallel network (b) and dual SPS (c).

Remark 1. Threshold functions are regular. In particular, if f is threshold with weights $w_1 \geqslant w_2 \geqslant \cdots \geqslant w_n$, then f is regular with respect to $\rho = (1, 2, \ldots, n)$. Since a k-out-of-n function is threshold with equal weights, it is regular with respect to any permutation.

We shall say that a function is *Double Regular* if it is ρ-regular and π-regular for some permutations ρ and π. (We will later require ρ and π to satisfy certain conditions. If it turns out that $\rho = \pi$, then with respect to this permutation, all regular functions are also double regular.)

The following Fig. 2 illustrates the inclusion relations among these classes of functions.

3.3. Series-Parallel Systems (SPSs)

Another class of Boolean functions we shall consider is the class of *Read-Once functions*. These are the structure function of *Series-Parallel Systems* (or SPSs in short). An SPS is a specially structured network between two terminal nodes, called the *source* and the *sink*. The structure function takes the value 1 if there is a path from the *source* to the *sink* that consists of functioning components. The simplest SPS consists of only one component: a single link connecting the source to the sink. All other SPSs can be defined recursively as a series or parallel connection of smaller SPSs. A *series connection* identifies the sink of one SPS with the source of the another one, while a *parallel connection* identifies the sources and the sinks of the two systems. Formally, if $\Gamma(N_1, f_1)$ and $\Gamma(N_2, f_2)$ are two systems with no common components, then $\Gamma(N_1 \cup N_2, f_1 \vee f_2)$ is their parallel connection, and $\Gamma(N_1 \cup N_2, f_1 \wedge f_2)$ is their series connection, where N_1 and N_2 are disjoint. All smaller SPSs, appearing recursively in the above definition, are called the *subsystems* of the considered SPS.

For example, the network $\Gamma(N, f)$ in Fig. 3(a) works if u_1 is working, or if u_2 and at least one of u_3 and u_4 are working. As customary, we shall represent the structure function by a logical expression in terms of the states of its components. For this example, we have

$$f(x_1, x_2, x_3, x_4) = x_1 \vee (x_2 \wedge (x_3 \vee x_4)). \tag{8}$$

In Fig. 3, the second network is not an SPS, since it cannot be obtained by the recursive procedure described above, while the first one is an SPS, formed by a parallel connection of the two subsystems $\{u_1\}$ and $\{u_2, u_3, u_4\}$. The latter one itself is a series connection of two smaller subsystems $\{u_2\}$ and $\{u_3, u_4\}$, where this last subsystem is formed again by a parallel connection of the two single component subsystems $\{u_3\}$ and $\{u_4\}$. Let us observe that the expression in (8) corresponds to the recursive definition of this system, and in particular, the brackets enclose its (non-singleton) subsystems.

The *depth* of an SPS is a measure of the number of series and/or parallel combinations in the system and is defined as follows: The depth of a simple series or a simple parallel system is 1. The depth of any other SPS is $1 + \max\{\text{depth of proper subsystems}\}$. For instance, the SPS in Fig. 3(a) has depth 3. Thus, every SPS is either a parallel or series connection of other SPS whose depths are at least 1 less than the original SPS.

It is important to note that the Boolean dual f^d of the structure function of an SPS $\Gamma(N, f)$ is again a structure function of another SPS $\Gamma^d(N, f^d)$, which we will call the *dual system* for the original SPS. The dual system can be obtained simply by interchanging the $\vee$ (parallel) and $\wedge$ (series) signs in the system function. If the global problem is (series) parallel for the original SPS then the global problem for the dual is parallel (series). For example, the dual of the system given in Fig. 3(a) is shown in Fig. 3(c), and its system function is

$$f(x_1, \ldots, x_4) = (x_1 \wedge (x_2 \vee (x_3 \wedge x_4))).$$

Let us note that, any SPS and its dual have the same depth. We also mention that all the results and algorithms in the following sections can straightforwardly be translated for the dual system, and hence without loss of generality, we can restrict our attention only to SPSs in which the global system is parallel.

4. Strategies and binary decision trees

As we noted earlier, inspection strategies can naturally be represented as *binary decision trees*, i.e. as rooted trees, in which every node has two or zero successors. Such a tree $\mathbf{T}$ is representing the system function f in the following way. Nodes of $\mathbf{T}$ with two successors are labelled by the components of the system. The two arcs leading to the successors from a node labelled by $u_i \in N$ represent the two states of u_i—the right arc corresponding to working, while the left arc corresponding to failing. Let us then label the right arc by x_i and the left arc by x_i. To every node a of the tree $\mathbf{T}$ let us associate the set $P(a) \subseteq \{x_1, x_1, x_2, x_2, \ldots, x_n, x_n\}$ consisting of the labels along the arcs on the unique path connecting the root of $\mathbf{T}$ to node a. Nodes of $\mathbf{T}$ with zero successors are called *leaves*.

Given a binary vector $\mathbf{x} \in \mathbb{B}^n$, let us follow a path starting from the root of $\mathbf{T}$ until we reach a leaf in the following way. At a node a of the tree labelled by $u_i \in N$ we follow the left arc, if $x_i = 0$, and the right arc if $x_i = 1$. Hence for every binary vector $\mathbf{x}$ we follow a unique path leading to a leaf node of $\mathbf{T}$. For a node a of $\mathbf{T}$ let us denote by $B(a)$ the set of those binary vectors for which the above path contains node a. Since $\mathbf{T}$ represents an inspection strategy for the system $\Gamma(N, f)$, all vectors in a set $B(a)$ corresponding to the leaf node a are state vectors of the same state (i.e. either all of them are working, or all of them are failing states), since otherwise the state of f would not be determined at this node uniquely. Let us label the leaf nodes of $\mathbf{T}$ according to the corresponding state of the system, i.e. by 1 those which correspond to working states, and by 0 those corresponding to the failing of the system. Let us denote furthermore by $L^1(\mathbf{T})$ the set of leaves of $\mathbf{T}$ labelled by 1, and let $L^0(\mathbf{T})$ denote the set of leaves labelled by 0. Nodes in $L^1(\mathbf{T})$ are called the *true-leaves* of $\mathbf{T}$, while those in $L^0(\mathbf{T})$ are called the *false leaves*.

We shall call a binary decision tree of f *reasonable*, if it is a reasonable inspection strategy for a system with system function f, i.e. if any subtree rooted at a non-leaf node contains both true and false leaves. Let us denote by $\mathbb{T}(f)$ the family of all reasonable binary decision trees which represent the Boolean function f.

Let us consider now a coherent system $\Gamma(N, f)$, a cost vector $\mathbf{c}$ and a probability vector $\mathbf{p}$, and a (reasonable) inspection strategy $\mathbf{S} \in \mathbb{T}(f)$ of this system.

For every state $\mathbf{x} \in \mathbb{B}^n$ the inspection algorithm will terminate in one of the leaf nodes of $\mathbf{S}$, and $B(a)$ is exactly the set of those binary vectors for which the algorithm stops in the leaf node a. Clearly, $B(a) \cap B(b) = \emptyset$ if a and b are different leaves of $\mathbf{S}$, and we have

$$T(f) = \bigcup_{a \in L^1(\mathbf{S})} B(a) \quad \text{and} \quad F(f) = \bigcup_{b \in L^0(\mathbf{S})} B(b). \tag{9}$$

Let us denote by $C(\mathbf{S}, \mathbf{x})$ the cost of inspection by strategy $\mathbf{S}$ if the system is in state $\mathbf{x}$. According to the above remark, this cost is clearly the same for all state vectors that belong to the same leaf. More precisely, if $\mathbf{x} \in B(a)$ for a leaf a, then

$$C(\mathbf{S}, \mathbf{x}) = Cost(\mathbf{S}, a) = \left(\sum_{i: x_i \in P(a) \text{ or } x_i \in P(a)} c_i \right). \tag{10}$$

Furthermore, the probability that the system is at one of the states belonging to $B(a)$ is

$$Prob(\mathbf{S}, a) = \left(\prod_{i: x_i \in P(a)} p_i \right) \left(\prod_{i: x_i \in P(a)} q_i \right), \tag{11}$$

whereas the probability that the system is in the state represented by the vector $\mathbf{x}$ is

$$Prob(\mathbf{x}) = \left(\prod_{i:x_i=1} p_i\right)\left(\prod_{i:x_i=0} q_i\right). \tag{12}$$

Hence, the expected cost of strategy $\mathbf{S}$ can be expressed as

$$E(\mathbf{S}) = \sum_{a\in L^1(\mathbf{S})\cup L^0(\mathbf{S})} Cost(\mathbf{S},a)Prob(\mathbf{S},a). \tag{13}$$

or equivalently as

$$E(\mathbf{S}) = \sum_{\mathbf{x}\in T(f)\cup F(f)} C(\mathbf{S},\mathbf{x})Prob(\mathbf{x}). \tag{14}$$

Given a system (N, f), a cost vector $\mathbf{c}$ and a probability vector $\mathbf{p}$, the problem of finding the best sequential inspection strategy can be restated as

$$\min_{\mathbf{S}\in\mathbb{T}(f)} E(\mathbf{S}). \tag{15}$$

We shall say that an inspection strategy $\mathbf{S}$ is α-*optimal*, if $\mathbf{S}$ minimizes the expression:

$$E^{\alpha}(\mathbf{S}) = \sum_{a\in L^{\alpha}(\mathbf{S})} Cost(\mathbf{S},a)Prob(\mathbf{S},a), \tag{16}$$

among all strategies in $\mathbb{T}(f)$, where $\alpha = 0$ or 1. Let us note that $E^1(\mathbf{S}) + E^0(\mathbf{S}) = E(\mathbf{S})$ holds for any strategy $\mathbf{S}$.

The concept of 1(0)-optimal strategies is useful in proving some of the results. Furthermore, they arise naturally in the following contexts. Suppose a company provides maintenance services for the machines it sells, which work continuously (e.g. generators). The crew visits a customer where such a machine is used on a regular schedule. If the machine needs maintenance, repair is needed and customer pays for the repair (and thus the cost of a lengthy inspection can be included). However, if the machine is found safe enough, then the customer does not pay anything and the crew proceeds to another customer. Hence the company is primarily interested in efficiently testing working systems, i.e. a 1-optimal strategy for testing its product.

For a possible application where one would be interested in finding a 0-optimal strategy let us consider the following situation. Customers show up at a used car dealer to sell their cars and the dealer employs a mechanic to check if the cars are good enough. If the car is found good enough, the dealer buys it and adds the cost of inspection to the selling price of the car. However, if a car is not found good enough then there is no business and the dealer has to cover the cost of inspection. So the dealer is interested in finding out efficiently the failing cases, which corresponds to a 0-optimal strategy.

Let us note that one can consider different measures other than the expected cost to optimize. For instance, one could try to minimize the worst case cost. In other words

$$\min_{\mathbf{S}\in\mathbb{T}(f)} \max_{\mathbf{a}\in L^1(\mathbf{S})\cup L^0(\mathbf{S})} C(\mathbf{S},\mathbf{a}). \tag{17}$$

There are further measures which are independent of $\mathbf{c}$ and $\mathbf{p}$. These measures are particularly important when the components are identical, i.e. $c_i = 1$ and $p_i = p$. (In particular, one can consider the case where $p_i = 0.5$.) These objectives include minimizing the total number of leaves, finding a lexicographically largest BDT. For a review of objective function issues see [49].

4.1. Representing problem instances and strategies

Let us recall that the input to the *Sequential Diagnosis Problem* consists of a description of the system function f and the cost and probability vectors $\mathbf{c}$ and $\mathbf{p}$. For some algorithms, a description of f^d is also required. In general, the system functions can be described by their *Dinsjuctive Normal Forms* (*DNFs*) or via an oracle, i.e. a black box that can evaluate the function f at any vector, in polynomial time. For certain special classes of functions, there can be more concise ways of representation. For instance, a threshold function can be fully described by specifying the weights associated with each variable and the threshold value, a total of $n + 1$ numbers.

The output of an algorithm is an optimal strategy. We have already seen that strategies are naturally represented by *BDTs*. The problem with this type of representation is that the size of the *BDT* could be exponentially larger than the size of the input, e.g. the size of the *DNF* of the function. From a practical point of view, one may decrease the size by merging the same subtrees together forming Binary Decision Diagrams (*BDD*s). Obviously there is no guarantee that the optimal *BDT* will have common subtrees. On the other hand, for certain classes of functions there exist efficient *BDD*s.

It would be worthwhile to investigate for which **p** and **c** these strategies are optimal. For further discussion on *BDD*s, see e.g. [65,43,49,34].

Another way to avoid this difficulty is the following. Instead of asking for an algorithm that outputs the whole *BDT*, one may look for a concise rule that outputs the next component to inspect given the states of the previously inspected components. By at most n repeated applications of this algorithm, one can find the state of the function for the current state vector, i.e., we find the path from the root to the leaf that is associated with the current state vector only, in an optimal *BDT*. Essentially, we only construct the necessary portion of an optimal tree that we are concerned with. We should note that if we are dealing with a certain class of systems, we need to make sure that the residual systems after fixing a subset of the components belong to the same class for this method to work. A second way to get around the problem of large output size is to look for an algorithm runs in polynomial time in the size of the input and the output. Although this may not be a good solution practically, if it is likely that we are going to deal with almost all 2^n state vectors, this may be an appropriate approach.

On the other hand, for certain classes of problem instances, one may not need to output the *BDT* at all. A good example is when the optimal strategy can be described by a permutation, i.e. the next component to inspect next is always the next relevant component in the permutation. We shall see examples of systems for which the optimal strategy is a permutation strategy in Section 5. Yet another example of such a compact representation of an optimal strategy for k-out-of-n systems is given in [10]. In this case, an optimal strategy is specified by the *block-walking representation* which is of size $O(n^2)$ and which can be generated in time $O(n^2)$.

This issue becomes critical especially in NP-completeness proofs of the variations of the problem, and in some cases, causes ambiguities. One has to be careful in defining the input and output of the problem and make sure that a short proof of a YES/NO certificate is provided for the decision problem to complete a proof of NP-completeness.

5. Literature review

5.1. Series and parallel systems

Let us recall that, if x_i is the variable associated with the ith component, the structure function of the series and parallel systems can be written (respectively) in the following way:

$$f(\mathbf{x}) = x_1 \wedge x_2 \wedge \cdots \wedge x_n \quad \text{and} \quad f(\mathbf{x}) = x_1 \vee x_2 \vee \cdots \vee x_n.$$

Series and parallel systems in n components are duals of each other. Note also that any reasonable tree for a series (parallel) system consists of a spine, since the left (right) branch of any non-terminal node is a false (true) leaf. This implies that a strategy corresponds to a permutation of the components.

Let σ, π be two permutations of $\{1, 2, \ldots, n\}$ such that

$$\frac{c_{\sigma_1}}{p_{\sigma_1}} \leqslant \frac{c_{\sigma_2}}{p_{\sigma_2}} \leqslant \cdots \leqslant \frac{c_{\sigma_n}}{p_{\sigma_n}} \quad \text{and} \quad \frac{c_{\pi_1}}{q_{\pi_1}} \leqslant \frac{c_{\pi_2}}{q_{\pi_2}} \leqslant \cdots \leqslant \frac{c_{\pi_n}}{q_{\pi_n}}. \tag{18}$$

Then an optimal strategy for a series (parallel) system is to inspect the components in the order $\pi(\sigma)$ until the state of the system is determined, i.e. until either a non-working (working) component is found or all the components have been inspected. These strategies are very intuitive in the following way. For a series system one stops inspection whenever a faulty component is found, and the goal is to find the strategy with the minimum expected cost. Hence, it will be advantageous to begin with a component which has low cost and high probability of not working, and this is quantified with the ratio c_i/q_i. One can make a dual argument for a parallel system.

It is also easy to show that the strategies mentioned above are optimal for series or parallel systems by an interchange argument. Regardless of their simplicity, these two systems arose in different contexts modelling various problems. The result mentioned above has been published as part of many papers, though more general cost structures and system functions are considered in some of them. Here are some typical examples and related applications from such papers.

Price [53] considers complex mechanisms which are subjected to a series of non-destructive tests, where the order of tests does not affect the result of the tests. In this study, the expression for the expected cost is given. It is proposed that each of the $n!$ sequences is tried and the one with the minimum expected cost is chosen. One year later Mitten [44] proves in the same journal that the sequence (20) is the optimal solution.

Kadane [38] considers the so-called *quiz show* problem. A quiz show contestant may choose the category of the next question. For each category p_a denotes the probability of knowing the right answer to the question. If he answers the question correctly the contestant will be given a reward x_a and be required to choose a new category (which was not chosen previously). If he answers incorrectly, he will receive the consolation prize and will leave the game with y_a plus his previous earnings. Suppose also that entering a category will require time t_a to recover and be ready to choose another

question. Knowing a discount rate $\beta \geqslant 0$ and the parameters p_a, x_a, y_a and t_a the problem is to determine in what order the contestant should choose categories to maximize his inspected earnings. It is not difficult to see that diagnosing a series system is a special case of this problem with $c_a = p_a x_a + (1 - p_a) y_a$, and $t_a = 0$.

Joyce [37] considers the same problem, in a totally different context. The paper discusses the organization and the criterion for funding of an applied research project that can be regarded as a collection of necessary but potentially unsuccessful tasks. A project consists of n independent tasks, and the project is successful if all the tasks are accomplished successfully. Associated with each task is the cost of the test, and probability of success. The goal is to execute the tasks in such an order that the total cost is minimized. Once a task is unsuccessful the project is unsuccessful. Obviously this problem is again nothing else but diagnosing a series system with the minimum expected cost. In this paper, Joyce generalizes this application to 2-Level SPSs where each global subsystem corresponds to various ways of accomplishing the final goal.

Another interesting paper where the same problem comes up is by Simon and Kadane [59]. This is one of the Artificial Intelligence papers where the optimal strategy for series and parallel systems are presented. In this paper, *optimal algorithms are derived for satisficing problem-solving search*. One of the three models considered in this paper is called "Satisficing Search without Ordering Constraints". The application considered under this model includes an unknown number of chests of Spanish treasure have been buried on a random basis at some of n sites, at a known depth of 3 ft. For each site there is a known unconditional probability, p_i that a chest was buried there and the cost of excavating site i is c_i. The search is terminated as soon as one treasure is found. The problem is to find a sequence of the sites such that the expected cost of excavation is minimized over all sequences.

Natarajan [51] considers the problem as a variation of the general Search of AND-OR Trees problem in artificial intelligence and proves the result for series and parallel systems. A generalization of the optimal algorithm is proposed for the more general class of Series-Parallel Systems.

Another interesting application is mentioned in [1]: suppose there are n acceptable candidates to fill a position, the estimated benefit from candidate i for the next M years is E_i and the probability of acceptance of the job by him/her is R_i. Also assume that this probability does not change during the entire selection process. When a candidate receives an offer, there is a fixed period of time during which he/she can accept or reject the offer; during such time, no other offers are extended to other candidates. As soon as one candidate accepts the offer, the search terminates. The objective is to find an ordering so as to maximize expected profit. One can see easily that this is the same problem as diagnosing a parallel system with $c_i = R_i E_i - C$ and $p_i = -R_i$, where C is the cost of offering the job to a candidate. Actually, Alidae's result is for a more general class of cost functions. Namely, let us consider n objects with no precedence relation. Associated with each object i is a positive weight w_i, a non-zero number p_i, and a cost function $f_i : R^N \to R_+$ where R^N are all the ordered subsets of the set of objects. The objective is to find an ordering $\pi = (j_1, j_2, \ldots, j_n)$ of the objects so as to minimize the total cost given by

$$TC(\pi) = \sum_{i=1}^{n} f_{j_i}(j_1, j_2, \ldots, j_i). \tag{19}$$

It is assumed that f_{j_i}, $i = 1, 2, \ldots, n$ are independent of the ordering of the objects $j_1, j_2, \ldots, j_{i-1}$. For each function f_{j_i}, it is assumed that

$$\{f_{j_i}(j_1, \ldots, j_\alpha, l, j_{\alpha+1}, \ldots, j_i) - f_{j_i}(j_1, j_2, \ldots, j_\alpha, j_{\alpha+1}, \ldots, j_i) = p_l w_{j_i} H(j_1, j_2, \ldots, j_{i-1}) \tag{20}$$

for any $l \notin B = (j_1, j_2, \ldots, j_{i-1})$, where B is a partial sequence and H is a positive set function defined on all subsets of n objects with $H(\emptyset) = 1$. Under these assumptions a sequence π minimizes the total cost (19) iff

$$\frac{p_{j_1}}{w_{j_1}} \leqslant \frac{p_{j_2}}{w_{j_2}} \leqslant \cdots \leqslant \frac{p_{j_n}}{w_{j_n}}. \tag{21}$$

By appropriate choice of f, H, p, and w one cannot only get the result for the series and parallel case but also solutions for some scheduling problems like minimizing the sum of weighted completion times and some variations of this problem. A quite similar result is obtained by Rau in [56].

Other papers which consider the same problem includes [41,55,18], in which the motivation is studying optimal multi-characteristics inspections. Essentially, a product is tested on a number of its characteristics failure of which results in the rejection of the product. There are associated costs for each of the tests and the probability that each test will fail the item is known. In some models, tests are not perfect, and there are associated costs of shipping a faulty product to the customer. The problem is to sequence the tests to minimize the total expected cost.

Another possible application is trying to find strategies to page cellular customers with the minimum expected number of signals sent. In this context, the customer can be in a finite set of areas with certain probabilities (or in none of them in which case he cannot be reached). The cost of sending a signal is same for all areas. This problem is the same as testing a parallel system. The variables correspond to the different areas. The difference from our framework is the

fact that variables are not independent. This dependence is expected due to the physical closeness of different areas (see, e.g. [67]).

5.2. Exact solutions and hardness results

Branch and bound and dynamic programming formulations have been proposed to solve the general problem of sequential diagnosis problem, both of which run in exponential time in general. A dynamic programming approach has been proposed in [16] for threshold functions. That formulation can be extended for general functions by the following recursive equation:

$$EC(f) = \min_i \{c_i + p_i EC(f_{x_i=1}) + q_i EC(f_{x_i=0})\}, \tag{22}$$

where $f_{x_i=j}$ is obtained from f by fixing x_i at $j \in \{0,1\}$. In [16], it is proved that even when the underlying function is a linear threshold function, the problem of finding the next component to inspect by minimizing the expected cost is NP hard. But this result is misleading in the sense that the input for a threshold function is $n+1$ numbers, yet the number of prime implicants can be exponential in the number of variables, n. In general a Boolean function is typically described by its prime implicants. There are more NP-hardness results for generalizations/variations of the problem in [36,14,17,35]

In [2], a branch and bound algorithm for solving the problem for general coherent systems is given. The node for branching is chosen according to a lower bound which is related to reliability importance of components. The reliability importance of component j is defined by $I_j = \partial h(\mathbf{p})/\partial p_j$, where $h(\mathbf{p})$ is the reliability function of the system and $\mathbf{p}$ is the vector of probabilities. Equivalently, $I_j = h(1_j, \mathbf{p}) - h(0_j, \mathbf{p})$. Also, by defining $\overline{I_j} = 1 - I_j$ and $d_j = c_j \overline{I_j}$, d_j can be used as a measure of the expected extra cost due to testing component j even though it is unnecessary to know its state. Finally, let us define $I = \sum_k c_k I_k$. It is proven that $I + d_i$ is a lower bound for the expected cost of all testing procedures which start by testing component i. This lower bound is then used to choose the next node to branch on. In general, the algorithm requires exponential time, moreover to implement the algorithm the value of the reliability function at some points has to be calculated. No experimental results are given. A similar branch and bound procedure is proposed in [57] for a more general problem namely, for the case when there are many classes and each class has many patterns with dependent components (see Section 5.5). Furthermore, a refined branch and bound method is proposed in [8] for the case of identical components with $p = 0.5$, again using similar ideas to those mentioned above.

There are also certain classes of the general problem for which optimal solutions can be obtained in polynomial time. These solvable instances are obtained by imposing restrictions on the type of the underlying Boolean function and/or the cost and probability data. (For definitions of these systems see Section 3.) We can summarize those results as follows:

5.2.1. k-out-of-n Systems and double regular systems

In [3,10], polynomial time algorithms that produce optimal solutions for k-out-of-n systems are given. Let us recall that k-out-of-n systems are regular with respect to any permutation. These algorithms are generalized further in [6] by providing a generalized algorithm that is optimal for *double regular systems* that are regular with respect to σ and π defined as (18). Let us note that this new class of solvable instances include all *double regular systems* with identical components, in particular the widely studied *threshold systems* with identical components as well as general k-out-of-n systems.

Essentially all of these algorithms produce an optimal strategy using 1- and 0-optimal strategies. It turns out that for such systems there exists 1- and 0-optimal permutation strategies. In particular, these permutations are, respectively, σ and π defined as (18). Then it can be shown that one can produce another strategy by "mating" these 1- and 0-optimal strategies. This new strategy turns out to be 1- and 0-optimal simultaneously, hence it is a minimum expected cost strategy.

5.2.2. 2-Level general and 3-level identical SPSs

Let us recall that an SPS is either a series or a parallel connection of smaller SPSs. A natural idea to find optimal strategies for an SPS is to consider subsystems one by one, in some order. Since the subsystems are also SPSs, same ordering process can be used for inspecting the subsystems. If the global problem is parallel (series), then one can stop as soon as a working (failing) subsystem is found concluding that the SPS works (fails). Indeed, this idea is the source for all the efficient strategies found in the literature (see e.g. [51]).

Exact polynomial time algorithms have been obtained for 2-Level deep general SPSs and 3-Level deep SPSs with identical components (i.e. $c_i = c$ and $p_i = p$ for all i) in this manner. These algorithms assign costs and probabilities (recursively going from the deepest level upwards) to the subsystems and order the subsystems either in increasing order of cost/probability of working or cost/probability of failing depending on whether the global problem is parallel or series. (Since at the deepest level we either have a series or parallel system, we can replace this subsystem with a single component having the optimal expected cost of inspecting the subsystem as its inspection cost and the corresponding probability of functioning. We can recursively work our way up to higher levels in a similar fashion. Eventually we will

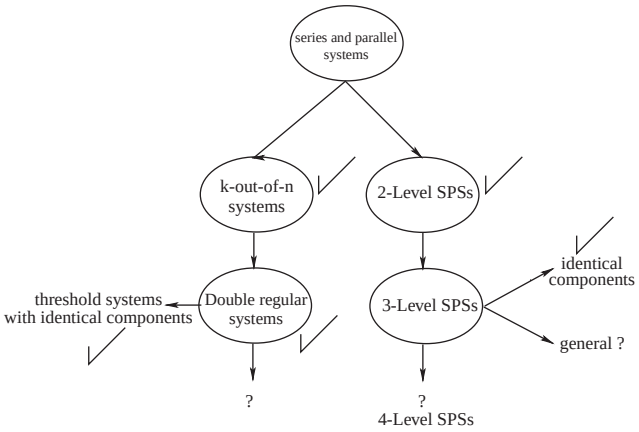

Fig. 4. Summary of solvable and open cases.

be left with a simple series or parallel system.) This is a natural generalization of the optimal strategies for simple series and parallel systems. Then a possible strategy is to inspect the subsystems one by one "optimally" in the appropriate order. As the components get inspected, one has the option to recompute the costs and probabilities of the residual SPS and reorder the subsystems. It turns out that this order does not change for 2- and 3-Level SPSs with identical components. Consequently, for such systems we obtain a permutation strategy. This is not true for 3-Level general SPSs and 4-Level SPSs with identical components. This type of algorithm was proposed for 2-Level general SPSs in [3], but the proof is incomplete. A complete proof for this and the generalization for 3-Level deep SPSs with identical components are given in [7]. A very similar algorithm is proposed in [37] for general SPSs, and is claimed, mistakenly, to be optimal.

It is shown in [51] that among the inspection strategies which inspect subsystems one by one, recursively, without interruptions, the above procedure is indeed the best for globally parallel systems (and analogously, the one generalizing the simple series case is the best for globally series systems). However, it can be demonstrated that an optimal strategy may require the interruption of the inspection of a subsystem, even for depth 3 SPSs (see [7]).

The solvable and open cases can be summarized as in Fig. 4.

5.3. Precedence constraints

In some applications, precedence constraints may be imposed on the tests, i.e. certain tests cannot be performed before certain others are performed. In general, this situation can be described by a directed acyclic graph whose nodes corresponds to the tests. An arc from node i to node j means that test j can be performed if test i has already been performed. As a very natural example, one could consider inspections performed on newly manufactured products. Some tests cannot be performed before others due to the physical shape of the product or due to the way it functions.

As another example, consider the buried Spanish treasure example of Section 5.1, suppose that at each site there are more than one layer that can carry the treasure, then the layers below can be excavated once all the layers that are on top of it have been excavated. So in this case the precedence graph consists of disjoint paths. Precedence constraints of this type are called parallel precedence constraints. In [28], an extension to this model that allows a third possibility when a layer is excavated is described. Namely when nothing is found at that layer, it is possible to realize that there is no treasure beneath that layer.

Another interesting class of precedence graphs, generalizing parallel precedence, is a forest where each tree is rooted and either all arcs are directed towards the root or away from the root.

Most results in the literature are for the case when we have a simple series or parallel system. Garey in [26] gives a polynomial time algorithm that works for the series case under forest-type precedence constraints. (Actually, this paper gives reduction theorems that would reduce the problem with the precedence constraints to another problem with no precedence constraints if the precedence graph is a forest.) Later Monma proves, in [47], that the only type of precedence constraints that Garey's algorithm can produce optimal strategies is exactly the forest type precedence constraints.

Chiu et al. [11] provide a different algorithm for parallel precedence constraints, for series or parallel systems. One can show that when applied to problems with parallel precedence constraints Garey's algorithm gives the same optimal strategy. In [11], they also give 1- and 0-optimal strategies for k-out-of-n systems with parallel precedence constraints

under some restrictive technical assumptions. These "1(0)-optimal" trees correspond to permutation strategies. In particular, they inspect components in the order as if the system were parallel (series) with the same precedence constraints. They show that just like the case without precedence constraints, one can obtain an optimal strategy by using these 1(0)-optimal trees, under the technical assumptions. It is also conjectured in this paper that this strategy is optimal without the restrictive technical assumption. In [59], Simon and Kadane give conditions for a strategy to be optimal for series or parallel systems under general precedence constraints, but they do not prescribe algorithms to execute these strategies.

In [48], an algorithm is given for the simple series case in which series-parallel type precedence constraints generalize the results of Garey. What is meant by series-parallel type precedence constraints in this report, is the following. Suppose $N_1, N_2, \ldots, N_j$ is a partition of the node set, then there is a permutation π of $\{1, 2, \ldots, j\}$ such that a component in N_{π_i} should be inspected after all components of N_{π_h} are inspected for $h \leqslant i$. Furthermore, there are parallel precedence constraints within each N_i. Actually, they give an optimal algorithm when the cost function satisfies certain conditions. This is the most general result we are aware of for the simple series case with precedence constraints.

As far as we know, the complexity of testing a series (parallel) system with general precedence constraints is not known.

5.4. Heuristics

Various heuristics methods have been proposed in the literature for different variations of the problem. Whereas some of these use the cost and probability data to find the next component to inspect, others try to incorporate the "importance" of the components via information theoretic arguments.

Jedrzejowicz in [36] proposes three different heuristics, and reports the results of some experiments. The first heuristic algorithm is pretty simple: the component to be inspected next is the one with the minimum cost among those that have not been inspected as long as the state of the system is not determined. The second algorithm uses the concept of unimportance of a component as defined in Section 5.2. Let f be the residual system function at the point we want to determine what to inspect next. Then the algorithm goes as follows: the component to inspect next is the one with the minimum, d_j among all relevant variables for the residual system. For the third algorithm proposed, let $M_f = P_f \cup C_f$, where P_f is the set of all prime implicants for f and C_f is the set of prime implicants for f^d. Each $p \in P_f$ and $c \in C_f$ has an associated probability. This algorithm picks the element in M_f with maximum probability and next inspects the element in that minimal path (or minimal cut) set with the minimum inspection cost. Once this element is inspected, f and M_f are updated, and the algorithm returns to the selection stage as long as the state of the system is not determined. These algorithms have been tested on 100 cases for eight different coherent systems, which are grouped into 2-Level SPSs, k-out-of-n and complex systems. The problems have 5–10 components. Algorithms 2 and 3 seem to work well according to these results.

Three heuristic methods are proposed in [16]. In particular, these ideas are generalizations of the optimal strategy for a series or parallel system. Optimistic sort, considers each prime implicant as a series system and associates the corresponding optimal expected cost and probability. Then the prime implicants (minimal path sets) are ordered with respect to their expected cost over probability ratios and the next component to inspect is the variable with the minimum ratio of cost over probability of not working. One can make a similar argument for pessimistic sort by replacing prime implicants with prime implicates (minimal cut sets). The basis of the third heuristic is the fact that we do not need to take the optimistic or pessimistic view since the intersection of any minimal path set and any minimal cut set is non-empty. We can find the minimal path set and minimal cut set with the minimum ratios and inspect any component that is in the intersection (intersection sort).

A generalization of the same heuristic is proposed for the case when there are many classes. Some special cases when these heuristics are provably optimal are reported:

1. Path sorting algorithm gives an optimal strategy for k-out-of-n systems (optimistic or pessimistic).
2. Optimistic (pessimistic) path sorting algorithm is optimal for parallel-series (series-parallel) systems.
3. Intersection sort algorithm is optimal for series and parallel systems. If there is a unique solution, it is also optimal for k-out-of-n systems. Further, it is conjectured to be optimal for all k-out-of-n systems.

Simulation results for path-sorting and intersection sort algorithm for linear threshold functions are reported. Probabilities, costs and weights for the function are uniformly sampled. The relative error of the heuristics are about 2% for $n < 10$. The exact solutions are obtained by an exact dynamic programming formulation.

A different approach is proposed by Sun et al. [61]. It involves moving from one strategy tree to another one by means of a delete/add (DA) move. The basic idea of a DA move is to remove a component from a path if it is inessential to the path and add it to other paths if necessary. A component is inessential to a path which leads to a true (false) leaf

if on that path the component is failing (functioning). Two heuristics are proposed, both of which use the DA move. It is proven that any strategy tree can be obtained from another one by a series of DA moves. Both heuristics produce optimal inspection strategies for series and parallel systems. Computational results are reported for small problems for which the exact solution can be obtained by dynamic programming. Both heuristics find the optimal solution more than half of the time. Also, for k-out-of-n systems, the heuristics finds an optimal inspection policy for all the examples used in the experiments.

5.5. Other applications and similar models

In this section, we review other possible applications of the general problem and its variations. One natural generalization to our framework is to consider general discrete valued functions. This version of the problem basically classifies incoming binary vectors into classes $c \in C$, where C is a finite discrete set. A convenient way to describe such a function, is to assign patterns to each class $c \in C$. A pattern is an n-vector consisting of 1s, 0s and blanks. $f(\mathbf{x}) = c$, i.e. $\mathbf{x}$ belongs to class c, if there exists a pattern p corresponding to class c for which $\mathbf{x}$ and p match in all non-blank coordinates. Three cases that have attracted attention of researchers are; when there are many classes but for each class there exists a unique pattern; when there are two classes and there are many patterns for each class and when there are many classes and there are numerous patterns corresponding to each class. In each of these cases, one can consider independent and dependent components in terms of functioning probabilities. For the independent case, probabilities are given as to the functionalities of the components, i.e $p_i = Prob(x_i$ is functioning), whereas for the dependent case probabilities are given for each of the possible state vector.

For instance, in [17], such a model is considered. The components need not be independent. Probabilities are associated with classes and inspection costs are associated with the variables. The goal is to classify an incoming binary vector into one of the possible classes. One to one correspondence between classes and rows is assumed. NP-hardness of the problem of constructing the optimal tree is proven. Two heuristics are proposed. The first one is an information theoretic heuristic which inspects next the component that yields the greatest expected reduction in classification entropy per unit of inspection cost, given the results of all inspections made so far. The second one is called the signature heuristic. A signature is created for each class. A signature for class i is a subset of variable values occurring only in row i. Thus, if these values are observed for a case, then the case must belong to class i. Experimental results for both these heuristics and another one which is a hybrid heuristic of these two are reported. The hybrid heuristic seems to outperform the other two. The heuristics can be implemented in polynomial time in the number of components and number of classes.

In [57], the problem of converting a limited entry decision table to an optimal computer program with minimum average processing time is considered. As a typical example in this case, the set of classes can represent the set of actions that may be taken for a credit card applicant and the binary variables record relevant properties of applicants such as whether or not the applicant's annual income is more than $40,000, or the applicant's age is 25 or more, etc. This problem is equivalent to the case in which there are many classes and each class has many patterns with dependent components. An information theory-based heuristic is proposed in [23] for this particular case which is similar in nature to the information theoretic heuristic explained above. The same application has also been considered in [45,46].

In [32], a file searching/screening application is mentioned: a data file of large population is considered where information on a list of attributes is stored for each member of the population. The information stored is binary, i.e. a member of the population either has or does not have the corresponding attribute. If the presence of one attribute is independent of the state of other attributes then the proportion of members having a certain attribute is a measure of the probability of any member of the population having that attribute. The problem is to list all members satisfying a logical expression (Boolean function). Hence, one has to test each member of the population and see if the given logical condition is satisfied or not. If the condition is satisfied, that member is added to the list of people satisfying the condition. Since number of members of the population is huge, one would like to minimize the total number of questions asked, hence the time spent. The same application has also been considered in [33,30].

Another possible application is optimizing calling service departments that have automatic answering services. One may want to minimize the average time the customer spends before he/she reaches an operator or gets the information needed from a recorded message. The questions asked by the automatic answering service, then, correspond to the tests. Obviously, the answer may be non-binary, nevertheless the set of possible answers is discrete and the goal is to find the strategy tree that gives the minimum possible cost.

In [16,17] the following applications are reported:

- The n components of $\mathbf{x}$ represent the states of n components in a newly manufactured multi-component reliability system, whose inspection is costly. The problem is to find a minimum expected cost strategy for determining whether the system will operate or not.

- The components of $\mathbf{x}$ represent the truth values of n propositions in an expert system and $f(\mathbf{x})$ is a function determining a conclusion from the truth values of these propositions. The problem is efficient inference, i.e., deriving the correct conclusion at the least expected cost.
- $f(\mathbf{x})$ is a bio-statistical discriminant or classification function with independent variables that are costly to observe, e.g. because they require intrusive tests upon the patient being classified, and the problem is adaptively selecting variables for observation so as to minimize the expected cost of computing $f(\mathbf{x})$.
- Another area of application is efficient computation in a distributed database environment. Suppose that a processor in a computer network is assigned the task of computing $f(\mathbf{x})$, but the components of $\mathbf{x}$ are stored at different remote locations in the network. If x_j is stored at a node having a distance (measured in communication cost or accessing time) c_j from the processor computing $f(\mathbf{x})$, then the problem is sequentially selecting components of $\mathbf{x}$ to retrieve over the network so as to minimize expected cost (or time).
- Design of interactive dialogues for diagnostic and advisory systems: in many applications of expert consultation systems, the user and the system should share a goal of minimizing the effort that the user must supply to get a query answered. The user's effort is expended in providing answers to questions asked by the system. The system determines which questions to ask based on its knowledge base and on some reasoning about what conjunctions of facts will suffice to justify conclusions.
- Pattern recognition and string matching: numerous applications in telecommunications, molecular biology, and computer science require a system to repeatedly identify patterns and/or classify strings of incoming symbols. The heuristics presented in this paper provide guidance on which symbols to examine next to match a new string to one of a finite number of known possible one as quickly or cheaply as possible.

A similar model is considered in [24] under the title of *Binary Identification Procedures*. A dynamic programming algorithm is proposed for the general problem. Some basic properties of optimal solutions are investigated. In addition, improvements over the general dynamic programming application are reported for a subclass of problems. Other results are reported in [25] for the case when all tests are in the form "is the object being classified in class c?". Another subclass of the same type of problem is investigated in [27], where all the classes are assigned equal probability, the cost of any test is unity. In this paper, the heuristic algorithm which essentially inspects next a component for which the probability of ending up on the subtree hanging on the right-hand side is closest to the probability of ending up on the subtree hanging on the left-hand side of that node. Intuitively, this condition says that the next test separates the classes in the residual problem as much as possible. It is shown that this heuristic idea can miss the optimal solution by any factor.

Another class of Boolean functions which frequently comes up in applications, especially in AI, is Horn functions. For instance, in [21,22] how model-based reasoning knowledge represented in the form of Horn clauses can be transformed into efficient diagnostic procedures, is described. The class of Horn functions consists of Boolean functions that admit a DNF for which there is at most one negated variable in each implicant. So this class is a more general class than the monotone Boolean functions. The importance of this class is due to the fact that it can be used to model "if...then..." rules, i.e. a set of positive premises implies a positive conclusion and many knowledge-based systems are composed of only Horn clauses. Because of the size of the knowledge bases, one needs to develop question-asking strategies to minimize the (weighted) average number of questions asked in the long run. Most of the results in this case are heuristics, most of which use mainly the ideas of the solvable subclasses of monotone Boolean functions (see e.g. [63,62,64]). A similar application arises in programming with database languages (e.g. SQL), where one frequently has to evaluate logical conditions. These are mostly implemented as if...then statements, and each ordering of these statements would suffice to evaluate the logical condition. One would be interested in this case in an implementation that would have the least running time. This corresponds to finding a good strategy, that prescribes when to execute which if...then statement. Other similar applications from AI literature include model-based diagnosis. Here, one is provided with a description of the system and observation(s) regarding how the system should behave in certain cases. The problem is to find a subset of the components, such that the assumption that these components do not function correctly will explain the discrepancies between the observations and the system description (see, e.g. [39,58]).

In [9], a practical approach is described for generating on-board diagnostics of dynamic automotive systems based on qualitative models. The approach is based on qualitative deviation models for the automatic derivation of on-board diagnostics based on decision trees. The Common Rail fuel injection system is used as an example and the prototype demonstration on board of a Lancia car is also discussed.

A similar problem comes up in logic design verification and correction. The goal is to verify a gate-level implementation of a digital system in terms of its functional specification. If the implementation does not function according its specification, it is important to correct the implementation automatically. For instance, in [12], an efficient exact search and pruning algorithm is proposed using Boolean equation techniques when there is a single simple design error.

Cox, in [54], considers an extension of the model in which components are allowed to have non-integer values between 0 and 1. The system structure is uniquely determined by its minimal path sets $P = \{P_1, P_2, \ldots, P_\alpha\}$ or its minimal cut sets $C = \{C_1, C_2, \ldots, C_\beta\}$. The system function is then defined to be

$$z = f(\mathbf{x}) = \max_{1 \leqslant r \leqslant \alpha} \min_{i \in P_r} x_i = \min_{1 \leqslant s \leqslant \beta} \max_{i \in K_s} x_i.$$

Intuitively, the system state (the degree to which it works) is the state of the 'worst' component in the 'best' minimal path set or, equivalently, the state of the 'best' component in the 'worst' minimal cut set. Also, we are given a cumulative distribution function $F_i(x_i)$ for the state of each component and the cost c_i for observing the true value of x_i. The problem is to find a sequential inspection procedure that minimizes the expected cost of determining whether $z > a$ for a given a. More generally, we may want to discover whether the system performance level falls within a certain interval $a < z \leqslant b$.

The following example illustrates one of many possible applications. Suppose we have a telecommunications network whose nodes are either service centers or customers. A service center and a customer are connected by a line if that service center can serve the customer. Suppose that the service centers are providing the same service but with different quality levels that can be determined only by costly inspections. The problem is to determine a sequential inspection strategy of the service centers, with the minimum expected cost, such that one can decide whether all customers are able to receive the service with at least a certain quality level. First, this problem is transformed to a binary one where components take only values 0 or 1. A dynamic programming formulation is given (essentially the same as in [16]). Since this algorithm takes exponential time in the worst case, generalization of heuristics in [16] are proposed for this problem.

Yet another variation is considered by Cox in [13], where misclassification is possible at some cost. In this model, each attribute takes a value from a discrete set (independent of each other) and the number of classes is arbitrary. Also, at any stage in the inspection policy the strategy is allowed to make a wrong decision about its class and the cost of a wrong decision is given by a loss function. In this case, the cost of misclassification should be included in the total cost of a strategy. Heuristic algorithms are proposed. The first one uses recursive partitioning. At any stage, the expected net value of information is calculated for each uninspected attribute given the values of already inspected attributes. The next attribute to be inspected is the one with the maximum information if it is positive. Otherwise, the algorithm classifies by minimizing the total expected loss. This calculation involves the inclusion–exclusion algorithm and takes too much time. Alternatively, a generalization of the optimistic sort algorithm in [16] is proposed. The next component to be inspected is chosen by using the same methods, but in addition the expected net value of the information for this attribute is calculated. If this is positive, it is inspected next; otherwise classification occurs by minimizing the expected total loss. Computational results are given for the special case of linear threshold systems, with an infinite cost of misclassification (these are the same experiments mentioned in [16]).

A similar model was considered in [15]. The main difference with our main model is the cost function that is being minimized. In this paper, algorithms for constructing optimal adaptive strategies in order to maximize the average revenue per unit time are presented, for the situation in which there is a multi-component reliability system that earns revenue while it is working and inspections, as well as replacements and repairs, are costly (opportunity costs due to lost revenue are also considered as well as the costs and times required for inspections). It is assumed that each component has a constant failure rate. Optimal algorithms are given for series and parallel systems. Heuristic algorithms are proposed for k-out-of-n and general systems. Also, characterizations of optimality are obtained for series systems in the case of discounted return over an infinite planning horizon.

References

[1] B. Alidaee, Optimal ordering policy of a sequential model, J. Optim. Theory Appl. 83 (1994) 199–205.
[2] Y. Ben-Dov, A branch and bound algorithm for minimizing the expected cost of testing coherent systems, Eur. J. Oper. Res. 7 (1981) 284–289.
[3] Y. Ben-Dov, Optimal testing procedures for special structures of coherent systems, Manage. Sci. 27 (12) (1981) 1410–1420.
[4] J.C. Bioch, T. Ibaraki, Complexity of identification and dualization of positive Boolean functions, Inform. and Comput. 123 (1995) 51–75.
[5] E. Boros, K. Elbassioni, V. Gurvich, L. Khachiyan, An incremental rnc algorithm for generating all maximal independent sets in hypergraphs of bounded dimension, DIMACS Research Report 2000-21, Rutgers University, DIMACS, 2000.
[6] E. Boros, T. Ünlüyurt, Diagnosing double regular systems, Ann. Math. Artif. Intell. 26 (1999) 171–191.
[7] E. Boros, T. Ünlüyurt, Sequential testing of series-parallel systems of small depth, in: Laguna, Velarde (Eds.), Kluwer, Boston, Computing Tools for the New Millennium, 2000, pp. 39–74.
[8] Y. Breitbart, A. Reiter, A branch-and-bound algorithm to obtain an optimal evaluation tree for monotonic Boolean functions, Acta Inform. 4 (1975) 311–319.
[9] F. Cascio, L. Console, M. Guagliumi, M. Osella, A. Panati, S. Sottano D.T. Dupré, Generating on-board diagnostics of dynamic automotive systems based on qualitative models, AI Comm. 12 (1/2) (1999) 33–43.

[10] M. Chang, W. Shi, W.K. Fuchs, Optimal diagnosis procedures for k-out-of-n structures, IEEE Trans. Comput. 39 (4) (1990) 559–564.
[11] S.Y. Chiu, L.A. Cox Jr., X. Sun, Optimal sequential inspections of reliability systems subject to parallel chain precedence constraints, 1997 (personal communication).
[12] P. Chung, Y. Wang, I.N. Hajj, Logic design error diagnosis and correction, IEEE Trans. Very Large Scale Integration (VLSI) Systems 2 (3) (1994) 320–332.
[13] L.A. Cox Jr., Incorporating statistical information into expert classification systems to reduce classification costs, Ann. Math. Artif. Intell. 2 (1990) 93–108.
[14] L.A. Cox, S.Y. Chiu, X. Sun, Least-cost failure diagnosis in uncertain reliability systems, Reliab. Engrg. System Safety 54 (1996) 203–216.
[15] L.A. Cox Jr., Y. Qiu, Optimal inspection and repair of renewable coherent systems with independent components and constant failure rates, Naval Res. Logist. 41 (1994) 771–788.
[16] L.A. Cox Jr., Y. Qiu, W. Kuehner, Heuristic least-cost computation of discrete classification functions with uncertain argument values, Ann. Oper. Res. 21 (1989) 1–21.
[17] D. Dubois, M.P. Wellman, B. D'Ambrosio, P. Smets (Eds.), Guess-and-verify heuristics for reducing uncertainties in expert classification systems, uncertainty in artificial intelligence, Proceedings of the Eighth Conference, Morgan Kaufman, Los Altos, CA, 1992.
[18] S.O. Duffuaa, A. Raouf, An optimal sequence in multicharacteristics inspection, J. Optim. Theory Appl. 67 (1) (1990) 79–87.
[19] T. Eiter, B. Gottlob, Identifying the minimal transversals of a hypergraph and related problems, SIAM J. Comput. 24 (6) (1995) 1278–1304.
[20] M. Fredman, L. Khachiyan, On the complexity of dualization of monotone disjunctive normal forms, J. Algorithms 21 (3) (1996) 618–628.
[21] G. Friedrich, G. Gottlob, W. Nejdl, Generating efficient diagnostic procedures from model-based knowledge using logic programming techniques (special issue on logic programming in intelligent decision and control systems), Comput. Math. Appl. 20 (9/10) (1990) 57–72.
[22] G. Friedrich, W. Nejdl, Generating efficient diagnostic decision tree procedures for model-based reasoning systems, in: International Workshop on Model-Based Diagnosis, Paris, July 1989.
[23] S. Ganapathy, V. Rajaraman, Information theory applied to conversion of decision tables to computer programs, Comm. ACM 16 (9) (1973) 532–539.
[24] M.R. Garey, Optimal binary identification procedures, SIAM J. Appl. Math. 23 (2) (1972) 173–186.
[25] M.R. Garey, Simple binary identification problems, IEEE Trans. Comput. 21 (1972) 588–590.
[26] M.R. Garey, Optimal task sequencing with precedence constraints, Discrete Math. 4 (1973) 37–56.
[27] M.R. Garey, R.L. Graham, Performance bounds on the splitting algorithm for binary testing, Acta Inform. 3 (1974) 347–355.
[28] D. Geiger, J.A. Barnett, Optimal satisficing tree searches, in: AAAI91, Anaheim, CA, Morgan Kaufmann, Los Altos, CA, 1991, pp. 441–445.
[29] R. Greiner, Finding optimal derivation strategies in redundant knowledge bases, Artificial Intelligence 50 (1990) 95–115.
[30] E. Gudes, A. Hoffman, A note on "an optimal evaluation of Boolean expressions in an online query system", Short Commun. Artif. Intell./Language Process. 22 (10) (1979) 550–553.
[31] V. Gurvich, L. Khachiyan, On generating the irredundant conjunctive and disjunctive normal forms of monotone Boolean functions, Discrete Appl. Math. 96–97 (1–3) (1999) 363–373.
[32] J.Y. Halpern, Evaluating Boolean function with random variables, Internat. J. Systems Sci. 5 (6) (1974) 545–553.
[33] M.Z. Hanani, An optimal evaluation of Boolean expressions in an online query system, Short Commun. Artif. Intell./Language Process. 20 (5) (1977) 344–347.
[34] M. Huth, M. Ryan, Logic in Computer Science: Modelling and Reasoning about Systems, Cambridge University Press, Cambridge, 1999.
[35] L. Hyafil, R.L. Rivest, Constructing optimal binary decision trees is NP-complete, Inform. Process. Lett. 5 (1) (1976) 15–17.
[36] P. Jedrzejowicz, Minimizing the average cost of testing coherent systems: complexity and approximate algorithms, IEEE Trans. Reliab. R-32 (1) (1983) 66–70.
[37] W.B. Joyce, Organizations of unsuccessful R&D projects, IEEE Trans. Engrg. Manage. EM-18 (2) (1971) 57–65.
[38] J.B. Kadane, Quiz show problems, J. Math. Anal. Appl. 27 (1969) 609–623.
[39] J. Kleer, B.C. Williams, Diagnosing multiple faults, Artificial Intelligence 32 (1987) 97–130.
[40] R. Kowalski, Search strategies for theorem proving, in: B. Meltzer, D. Mitchie (Eds.), Machine Intelligence, Vol. 5, Edinburgh University Press, Edinburgh, 1969, pp. 181–201.
[41] H.L. Lee, On the optimality of a simplified multicharacteristic component inspection model, IIE Trans. 20 (4) (1988) 392–398.
[42] K. Makino, T. Ibaraki, The maximum latency and identification of positive Boolean functions, in: D.Z. Du, X.S. Zhang (Eds.), ISAAC'94 Algorithms and Computation, Lecture Notes in Computer Science, Vol. 834, Springer, Berlin, Heidelberg, 1994, pp. 324–332.
[43] K.L. McMillan, Symbolic Model Checking, Kluwer, Dordrecht, 1993, ISBN 0-7923-9380-5.
[44] L.G. Mitten, An analytic solution to the least cost testing sequence problem, J. Indust. Engrg. (January–February 1960) 17.
[45] M. Miyakawa, Optimum decision trees—an optimal variable theorem and its applications, Acta Inform. 22 (5) (1985) 475–498.
[46] M. Miyakawa, Criteria for selecting a variable in the construction of efficient trees, IEEE Trans. Comput. 38 (1) (1989) 130–141.
[47] C.L. Monma, Sequencing with general precedence constraints, Discrete Appl. Math. 3 (1981) 137–150.

[48] C.L. Monma, J.B. Sidney, Sequencing with series-parallel precedence constraints, Math. Oper. Res. 4 (3) (1979) 215–224.
[49] B.M.E. Moret, Decision trees and diagrams, Comput. Surveys 14 (4) (1982) 593–623.
[50] S. Muroga, Threshold Logic and Its Applications, Wiley, New York, 1971.
[51] K.S. Natarajan, Optimizing depth-first search of AND-OR trees, Technical Report, IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, 1986.
[52] N.J. Nilsson, Problem-solving Methods in Artificial Intelligence, McGraw-Hill, New York, 1971.
[53] H.W. Price, Least-cost testing sequence, J. Indust. Engrg. July-August (1959) 278–279.
[54] Y. Qiu, L.A. Cox Jr., Heuristic testing procedures for general coherent systems, Eur. J. Oper. Res. 69 (1993) 65–74.
[55] A. Raouf, J.K. Jain, P.T. Sathe, A cost-minimization model for multicharacteristic component inspection, IIE Trans. 15 (3) (1983) 187–194.
[56] J.G. Rau, Minimizing a function of permutations of n integers, Technical Notes, 1969, pp. 237–240.
[57] L.T. Reinwald, R.M. Soland, Conversion of limited-entry decision tables to optimal computer programs, i: minimum average processing time, J. Assoc. Comput. Mach. 13 (3) (1966) 339–358.
[58] R. Reiter, A theory of diagnosis from first principles, Artificial Intelligence 32 (1987) 57–95.
[59] H.A. Simon, J.B. Kadane, Optimal problem-solving search: all-or-none solutions, Artificial Intelligence 6 (1975) 235–247.
[60] D.E. Smith, Controlling backward inference, Artificial Intelligence 39 (1989) 145–208.
[61] X. Sun, S.Y. Chiu, L.A. Cox, A hill-climbing approach for optimizing classification trees (personal communication).
[62] J. Wang, Identifying key missing data for inference under uncertainty, Internat. J. Approx. Reason. 10 (1994) 287–309.
[63] J. Wang, E. Triantaphyllou, A cost effective question-asking strategy for horn clause systems, Ann. Math. Artif. Intell. 17 (1996) 359–379.
[64] J. Wang, J. Vanda Vate, Question asking strategies for horn clause systems, Ann. Math. Artif. Intell. 1 (1990) 359–370.
[65] I. Wegener, Branching Programs and Binary Decision Diagrams: Theory and Applications, SIAM Monographs on Discrete Mathematics and Applications, SIAM, Philadelphia, PA, 2000.
[66] R.O. Winder, Threshold logic, Ph.D. Dissertation, Department of Mathematics, Princeton University, 1962.
[67] A. Yener, C. Rose, Highly mobile users and paging: optimal polling strategies, IEEE Trans. Veh. Tech. 47 (4) (1998) 1251–1257.