Journal of Combinatorial Optimization, 10, 261–282, 2005

Testing Systems of Identical Components

TONGUÇ ÜNLÜYURT tonguc@sabanciuniv.edu
Sabanci University, Faculty of Engineering and Natural Sciences, Orhanlı, Tuzla, 34956, İstanbul, Turkey

Received January 18, 2005; Revised June 1, 2005; Accepted June 17, 2005

Abstract. We consider the problem of testing sequentially the components of a multi-component reliability system in order to figure out the state of the system via costly tests. In particular, systems with identical components are considered. The notion of lexicographically large binary decision trees is introduced and a heuristic algorithm based on that notion is proposed. The performance of the heuristic algorithm is demonstrated by computational results, for various classes of functions. In particular, in all 200 random cases where the underlying function is a threshold function, the proposed heuristic produces optimal solutions.

Keywords: sequential testing, boolean functions, binary decision trees

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 (apriori) probability that it is working correctly, are known in advance. A special case that have attracted the attention of researchers is when the system is composed of identical components (i.e. inspecting any component costs the same and the probability of functioning is the same for all components). 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. (Cox et al., 1989, 1996), manufacturing (testing machines before shipping, or testing for replacement in technical service centers, see e.g. (Duffuaa and Raouf, 1990)), design of screening procedures (see (e.g. (Garey, 1973); (Halpern, 1974)), electrical engineering (wafer probe testing, see (Chang et al., 1990)), artificial intelligence (finding optimal derivation strategies in knowledge bases, see e.g. (Greiner, 1990); best-value or satisficing search algorithms, see e.g. (Simon and Kadane, 1975); (Smith, 1989), optimal evaluation of Boolean functions, see e.g. (Hanani, 1977); (Gudes and Hoffman, 1979)).

In this paper, first we define the problem and provide background information. We introduce the notion of lexicographically large trees. Then we provide results that suggest that lexicographically large trees are good candidate solutions with respect to expected cost. Finally, a heuristic to find lexicographically large binary decision trees is proposed and experimental results are given for some special cases when the optimal solution can be obtained. It turns out that the proposed heuristic performs very well on those instances.

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 $f(\mathbf{x}) \geq f(\mathbf{y})$ whenever $\mathbf{x} \geq \mathbf{y}$. 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.

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* $\mathcal{S}$ 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 $N = \{u_1, u_2, u_3\}$ with system function

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

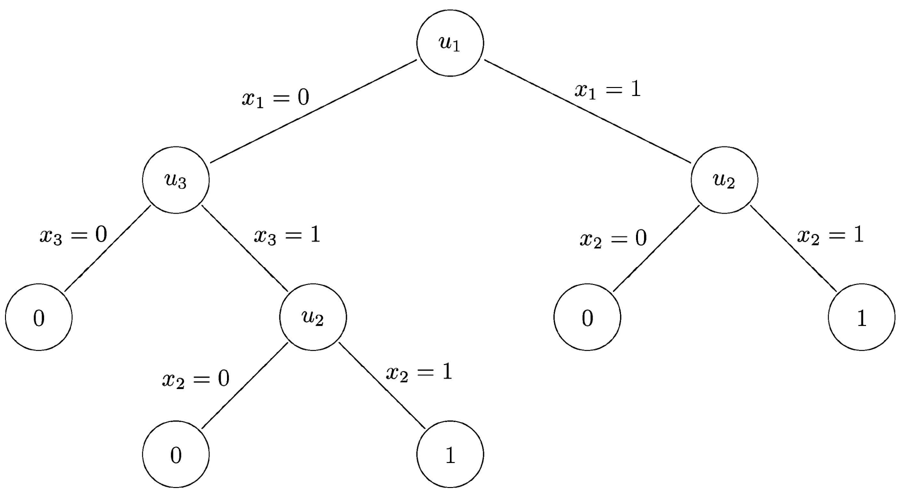

Figure 1. An inspection strategy for the system of Example 1.

Then, figure 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 costly. Let us denote by c_i the cost of inspecting component i.

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 figure 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 apriori distribution of the states of the components is known. We shall assume in this paper that component i works with probability p_i and fails with probability $q_i = 1 - p_i$. We shall also assume that the components work or fail independently of each other.

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 $\mathcal{S}$ 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 figure 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.

2.1. Classes of monotone system (boolean) functions

Let us 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.

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

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

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

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

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

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

the family of its prime implicants, and let

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

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

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

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

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

respectively.

We now describe certain classes of Monotone System (Boolean) Functions for which the *Sequential Diagnosis* problem is tractable. We will use these classes of systems to evaluate the performance of the heuristic algorithm that will be proposed later.

2.1.1. Simple series and parallel 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 i^{th} 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.

2.1.2. k-out-of-n and threshold systems. Let us first define the well known class of *threshold functions*, that arise many 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

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

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. One can easily see that a k-out-of-n function is a threshold function for which all weights are equal.

2.1.3. 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* consisting 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 $\Gamma(N_1, f_1)$

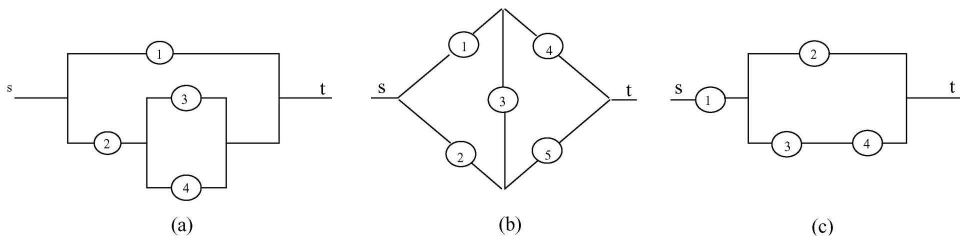

Figure 2. Examples for an SPS (a), and a non series-parallel network (b), dual SPS (c).

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 figure 2(a) works if u_1 is working, or if u_2 and at least one of u_3 and u_4 are working. 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 figure 2, 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 $\{u_4\}$.

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 figure 2 (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.

2.2. *Strategies and binary decision trees*

As we noted earlier, inspection strategies can naturally be represented as *binary decision trees* (BDT), i.e. as rooted trees, in which every node has two or zero successors. Such a tree $\mathcal{T}$ is representing the system function f in the following way. Nodes of $\mathcal{T}$ with two successors are labelled by the components of the system. The two arcs leading to the successors from a node represent the two states—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 $\overline{x}_i$. To every node a of the tree $\mathcal{T}$ let us associate the set $P(a) \subseteq \{x_1, \overline{x}_1, x_2, \overline{x}_2, \ldots, x_n, \overline{x}_n\}$ consisting of the labels along the arcs on the unique path connecting the root of $\mathcal{T}$ to node a. Nodes of $\mathcal{T}$ 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 $\mathcal{T}$ 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 $\mathcal{T}$. For a node a of $\mathcal{T}$ let us denote by $B(a)$ the set of those binary vectors for which the above path contains node a. Since $\mathcal{T}$ 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 denote the leaf nodes of the tree by $L(\mathcal{T})$.

Let us denote by $\mathbb{T}(f)$ the family of all binary decision trees corresponding to reasonable strategies which represent the Boolean function f.

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

Let us denote by $C(\mathcal{S}, \mathbf{x})$ the cost of inspection by strategy $\mathcal{S}$ 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

$$C(\mathcal{S}, \mathbf{x}) = \mathrm{Cost}(\mathcal{S}, a) = \left(\sum_{i: x_i \in P(a) \text{ or } \overline{x}_i \in P(a)} c_i \right). \tag{9}$$

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

$$\mathrm{Prob}(\mathcal{S}, a) = \left(\prod_{i: x_i \in P(a)} p_i \right) \left(\prod_{i: \overline{x}_i \in P(a)} q_i \right) \tag{10}$$

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

$$\mathrm{Prob}(\mathbf{x}) = \left(\prod_{i: x_i = 1} p_i \right) \left(\prod_{i: x_i = 0} q_i \right). \tag{11}$$

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

$$E(\mathcal{S}) = \sum_{a \in L(\mathcal{S})} \mathrm{Cost}(\mathcal{S}, a) \mathrm{Prob}(\mathcal{S}, a) \tag{12}$$

or equivalently as

$$E(\mathcal{S}) = \sum_{\mathbf{x} \in T(f) \cup F(f)} C(\mathcal{S}, \mathbf{x}) \mathrm{Prob}(\mathbf{x}). \tag{13}$$

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

$$\min_{\mathcal{S} \in \mathbb{T}(f)} E(\mathcal{S}). \tag{14}$$

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

$$\min_{\mathcal{S}\in\mathbb{T}(f)} \max_{\mathbf{a}\in L(\mathcal{S})} C(\mathcal{S}, \mathbf{a}). \tag{15}$$

3. Testing systems with identical components

As mentioned before, an interesting special case of the general problem occurs when the components of the system are identical. In other words, $c_i = c$ and $p_i = p$ for all i. Let us note that in this case the expected cost of a strategy is the average number of inspected components in the long run. In some studies it is further assumed that $p = 0.5$. It might seem that this will make the problem easier, but in fact this case can become even more difficult in the sense that it is difficult to extract information on the functionality of the components, by using the probability and cost information. For instance, one cannot talk about cheap tests in this case. There are some results in the literature for some special cases of functions. For instance in Boros and Ünlüyurt (2000), optimal polynomial time algorithms are provided for 3-level Series-Parallel systems that consist of identical components. Similar systems are also considered in Arseneau (1996). The results in Boros and Ünlüyurt (1999) can be extended to obtain solutions for threshold functions with identical components. This extends the results in Chang et al. (1990) for k-out-of-n systems. In this section, we are going to study the case in which the system function is any arbitrary monotone Boolean function. First, we review the results in the literature and state conditions for certain system functions that are satisfied by an optimal strategy. Then, we are going to introduce the notion of lexicographically large BDTs and propose a heuristic algorithm based on this notion.

3.1. Review and some extensions

In the literature, some heuristic algorithms have been proposed. The basic idea in almost all related articles is to inspect a variable that appears in short implicants and implicants of the dual. For instance, the following heuristic, proposed in Breitbart and Reiter (1975) for the case $p = 0.5$, inspects next a component which appears the most times in the minimal length primal and dual prime implicants, with some tie breaking rule to ensure a unique solution. It is also shown that if the primal *DNF* has linear terms, then every optimal algorithm inspects these linear variables first.

We can generalize the latter result for any value of p in the following way:

Lemma 1. *If f or f^d has a linear implicant, then there exists an optimal algorithm that starts inspection with that variable.*

Proof: Suppose $f = x \bigvee g(y_1, y_2, \ldots, y_n)$ and let us consider an optimal strategy for f which does not inspect x first. Observe that x should be on all paths that lead to false leaves. Then, we have the following two cases.

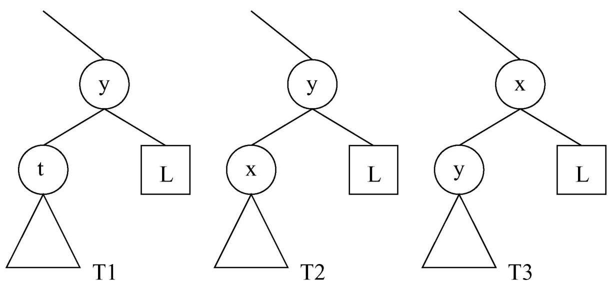

Figure 3. Functions with linear implicants.

Case 1 x is also on all paths that lead to true leaves. Then, Prob(x is inspected) $= 1$ and one can construct another BDT which does not have higher cost by inspecting x first (i.e. the root of the strategy tree is indexed by x.)

Case 2 There exists a true path along which x is not inspected. Let L be a leaf that is at the deepest level among all such leaves. Let us consider the subtree rooted at t, the twin of L. (see $T1$ in figure 3.) Since x is on all paths for this subtree, we can construct another subtree that does not have higher cost and such that $T2$ inspects x at the root of this subtree. From $T2$, we can obtain $T3$ by swapping x and y (see figure 3), which is is also optimal since it has the same expected cost as $T2$ due to the identical nature of the components. We are now back to case 1, since if there are still true leaves for which the paths from the root to this leaf do not include x, we can do the same operation, and we will eventually obtain an optimal tree that starts by inspecting x.

Finally, the case when f^d has a linear implicant follows by duality. □

In a reasonable tree, associated with each leaf there exists an implicant or implicant of the dual that causes to end up at that leaf. Related to this, we claim the following result:

Lemma 2. *There exists an optimal BDT such that the last two variables on the path from the root to a leaf associated with a prime implicant* (*implicant of the dual*) *belong to that prime implicant* (*prime implicant of the dual*), *assuming that there are at least two variables for that implicant.*

Proof: Without loss of generality, consider a true leaf in an optimal tree. Obviously, the last variable on the path belongs to the corresponding prime implicant which, by assumption, has at least two variables. Then, the corresponding function for the parent of the last variable has a linear term, and by Lemma 1, there exists an optimal tree which begins inspection by that variable. □

In Breitbart and Gal (1978), the class of algorithms which inspect next only variables appearing in a minimal length conjunction is considered (for the case $p = 0.5$.) If $\mathcal{A}$ denotes this class of algorithms, (Breitbart and Gal, 1978) shows that $b_n \geq \frac{n}{\log^2 n}$, $w_n \geq \frac{n}{\log n}$

and $R_w(f) \leq \frac{n}{\log n}$ for a (large) class of monotone Boolean functions, where $R_w(f) = \max_{A \in \mathcal{A}} \frac{E(f,A)}{E(f)}$, $R_b(f) = \min_{A \in \mathcal{A}} \frac{E(f,A)}{E(f)}$, $w_n = \max_{f \in M} R_w(f)$, $b_n = \max_{f \in M} R_b(f)$ and M is the set of all monotone functions. Here $E(f)$ denotes the value of the expected cost of the optimal BDT and $E(f, A)$ denotes the value of the expected cost of the BDT obtained by an algorithm from $\mathcal{A}$. All these results indicate that inspecting from short implicants is a promising strategy. An exact branch and bound approach is provided in Breitbart and Reiter (1975) that works for small problems.

The BDT with the minimum number of nodes has been proposed as a candidate for a good solution in the literature. (Let us recall that minimizing the number of nodes, itself could be another type of reasonable objective in a diagnosis problem.) Yet, the problem of finding such a BDT is hard. Further, the BDT with fewer number of leaves does not always have smaller cost (for an example see (Ünlüyurt, 1999)).

4. Lexicographically large BDTs

From this point on, in this section, we shall assume that $c_i = 1$ and $p_i = 0.5$ for all $i = 1, 2, \ldots, n$. We will now introduce the notion of *lexicographically large BDTs*. Let us associate with a strategy an n-vector, $l(\mathcal{S})$, whose i^{th} component is the number of leaves at depth i in the corresponding BDT.

Let **a** and **b** be n dimensional vectors. Then **a** is *lexicographically larger* than **b**, denoted by $\mathbf{a} \succ \mathbf{b}$, if there exists an index k such that $a_i = b_i$ for $i \leq k - 1$ and $a_k > b_k$. Let us also note that $\succ$ is a complete ordering of any subset of vectors.

We will say that strategy $\mathcal{S}$, (or its corresponding BDT) is *lexicographically larger* than strategy $\mathcal{S}'$ (or its corresponding BDT) if $l(\mathcal{S}) \succ l(\mathcal{S}')$. (Note that this definition does not require the strategies to correspond to the same Boolean function.)

Let us first note that if the primal or the dual has a linear term, then obviously the lexicographically largest strategy starts inspection with the variable of the linear term, and we have already shown the existence of an optimal strategy which starts inspection with that variable (even when $p \neq 0.5$).

Suppose $l(\mathcal{S}) = (a_1, a_2, \ldots, a_n)$ is a vector corresponding to some reasonable strategy for some function. Such a vector should satisfy certain conditions. First of all,

$$\sum_{i=1}^{n} a_i 2^{n-i} = 2^n \tag{16}$$

must hold, since 2^{-i} is the probability of reaching a leaf at depth i, and the sum of all the probabilities of reaching some leaf should total to 1. In addition we have,

$$\sum_{i=1}^{n} a_i \geq n + 1, \tag{17}$$

since each variable shows up at least once at an internal node, and the number of leaves is one more than the number of internal nodes in a binary tree.

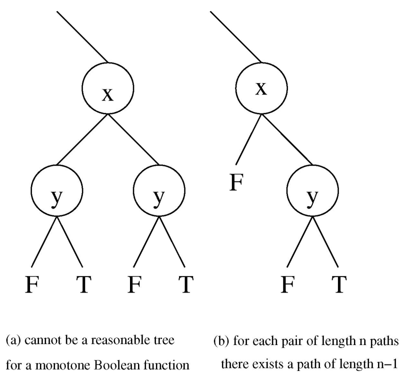
x
y
y
F
T
F
T
x
F
y
F
T
(a) cannot be a reasonable tree for a monotone Boolean function
(b) for each pair of length n paths there exists a path of length n−1

Figure 4. Paths of length $n - 1$.

Further, the following must also hold:

Lemma 3. *For any reasonable BDT, $a_{n-1} \geq \frac{a_n}{2}$.*

Proof: Suppose that in our BDT there exists a path that has length n. Let us consider the subtree that consists of the last two components that are inspected on this path. Since this subtree should be a reasonable tree, it is not possible to have, with respect to the original tree, 4 paths of length n. In this subtree, we could have 2 paths of length n and one path of length $n - 1$, as can be seen on figure 4. Hence, the result follows. □

Let us call a vector $(a_1, a_2, \ldots, a_n)$ *valid* if it satisfies (16), (17) and the condition stated in Lemma 3. The total expected cost for a BDT with $l(\mathcal{S}) = (a_1, a_2, \ldots, a_n)$ can be written as

$$C(\mathbf{a}) = \sum_{i=1}^{n} a_i i 2^{n-i}. \tag{18}$$

In the remainder of this section, we state results that support the idea that lexicographically large BDTs should not have high expected inspection cost. We consider the set of all valid vectors and we shall, in some sense, try to compare the total cost of *lexicographically large valid* vectors with other valid vectors.

Note that a valid vector may not correspond to a reasonable BDT for a particular function f. In particular, if we have a pair of valid vectors, they may not represent two reasonable strategies for the same function. Further, it is possible that a valid vector does not correspond to any reasonable BDT for any monotone function. For instance, when we try construct a BDT corresponding to $a = (1, 0, 2, 4)$ that is supposed to have only 4 variables, we see that we need at least 6 variables to do that.

4.1. Lexicographically large trees with small number of leaves

In this section, we consider the vectors that satisfy (16) and show that the vector that is lexicographically largest among the vectors with the same number of leaves, has the minimum cost among those vectors. In other words, we show that $C(\mathbf{a}) - C(\mathbf{b}) \leq 0$ if

$$\begin{aligned} &\sum_{i=1}^{n} a_i 2^{(n-i)} = 2^n, \\ &\sum_{i=1}^{n} b_i 2^{(n-i)} = 2^n, \\ &\qquad \sum_{i=1}^{n} a_i = \sum_{i=1}^{n} b_i = L, \quad \text{and} \end{aligned} \tag{19}$$

and $\mathbf{a}$ is the lexicographically largest vector among all vectors with L leaves. We will then explore what happens as the number of leaves changes. We need a number of technical lemmas to reach our results.

We first define operation L^+ on a vector $\mathbf{a}$, satisfying (16), by defining, $\mathbf{a}' = L^+(\mathbf{a})$ for some indices i and j satisfying the conditions below, as

$a'_{i-1} := a_{i-1} + 1$, $a'_i := a_i - 2$, where $a_i \geq 2$,
$a'_j := a_j - 1$, $a'_{j+1} := a_{j+1} + 2$, where $a_j \geq 1$ and $j \geq i$.

In what follows, we will not refer to the specific indices when L^+ is applied, whenever they can be figured out from the context. Suppose $\mathbf{a}$ is a feasible vector for which L^+ can be applied. If $\mathbf{a} = (a_1, \ldots, a_{i-1}, a_i, \ldots, a_j, a_{j+1}, \ldots, a_n)$, then $L^+(\mathbf{a}) = (a_1, a_2, \ldots, a_{i-1} + 1, a_i - 2, \ldots, a_j - 1, a_{j+1} + 2, \ldots, a_n)$.

Proposition 2. *Suppose it is possible to perform* L^+ *on* $\mathbf{a}$ *and we obtain valid* $\mathbf{a}'$ *after the operation. Then,*
(a) $\mathbf{a}' \succ \mathbf{a}$,
(b) $\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} a'_i$,
(c) *If* $\mathbf{a}$ *satisfies* (16), *so does* $\mathbf{a}'$, *and*
(d) $C(\mathbf{a}) \geq C(\mathbf{a}')$.

Proof:

(a) The first index that $\mathbf{a}$ and $\mathbf{a}'$ differ in is $i - 1$, and $a_{i-1} < a'_{i-1}$.
(b) To obtain $\mathbf{a}'$, we increase a_{i-1} by 1 and a_{j+1} by 2, whereas we decrease a_i by 2 and a_j by 1. Hence the net change in the sum is 0.

(c) We need to show that if $\sum_{i=1}^{n} 2^{(n-i)}a_i = 2^n$, then $\sum_{i=1}^{n} a_i' = 2^n$. This is true since

$$\begin{aligned}\sum_{i=1}^{n} a_i' 2^{(n-i)} &= \sum_{i=1}^{n} a_i 2^{(n-i)} + 2^{n-i+1} - 2\big(2^{(n-i)}\big) - 2^{(n-j)} + 2\big(2^{(n-j-1)}\big)\\ &= \sum_{i=1}^{n} a_i 2^{(n-i)}\\ &= 2^n. \end{aligned}\tag{20}$$

(d)

$$\begin{aligned}C(\mathbf{a}) - C(\mathbf{a}') &= \sum_{j=1}^{n} j2^{(n-j)}(a_j - a_j')\\ &= -(i-1)2^{(n-i+1)} + i2\big(2^{(n-i)}\big) + j2^{(n-j)} - 2(j+1)2^{(n-j-1)}\\ &= 2^{(n-i+1)} - 2^{(n-j)}\\ &\geq 0, \text{ since } j \geq i. \end{aligned}\tag{21}$$

□

Proposition 3. *If L^+ cannot be performed on a feasible vector* $\mathbf{a}$, *then either of the following is true*:

(a) *There exists an $i < n$ for which $a_i = 2$ (in which case $a_j = 0$ for $n > j > i$ and $a_j \leq 1$ for $j < i$.), or*
(b) *$a_i \leq 1$ for $n > i \geq 1$.*

Proof: Let us first suppose that that there exists $i < n$ for which $a_i = 2$. If $i = 1$, we necessarily have $\mathbf{a} = (2, 0, 0, \ldots, 0)$; hence, the first statement is true. If $n > i > 1$ and if there exists $j > i$ where $a_j > 0$, then we can apply L^+ for the indices i and j, which contradicts the assumption. Hence, $a_j = 0$ for $j > i$. Again, if $n > i > 1$ and if we have $i' < i$ such that $a_i' = 2$, we can apply L^+ for indices i' and i, which is again a contradiction. Hence, $a_j \leq 1$ for $j \leq i$. Note there can be no $i < n$ such that $a_i \geq 3$, since then we could apply L^+ for $i = j$. Now suppose that there is no $i < n$ with $a_i = 2$, then $a_i = 0$ or 1 for $i < n$. □

Lemma 4. *Suppose* $\mathbf{a}$ *is a feasible vector with L leaves. Then L^+ cannot be performed on* $\mathbf{a}$ *if and only if* $\mathbf{a}$ *is the lexicographically largest feasible vector with L leaves.*

Proof: First suppose $\mathbf{a}$ is the lexicographically largest feasible vector with L leaves, then L^+ cannot be performed on $\mathbf{a}$ since the resulting vector $\mathbf{a}'$ would be a feasible vector, and $\mathbf{a}' \succ \mathbf{a}$. Conversely, suppose there exists a vector $\mathbf{b}$ which is lexicographically larger than $\mathbf{a}$, yet L^+ cannot be performed on $\mathbf{a}$. First, note that for each i we can subtract $\min\{a_i, b_i\}$

from a_i and b_i, to get two vectors $\mathbf{a}^*$ and $\mathbf{b}^*$ such that $a_i^* b_i^* = 0$. These new vectors are no longer feasible since $\sum_{i=1}^n a_i^*(b_i^*)2^{(n-i)} \neq 2^n$, yet they satisfy the following:

$$\begin{aligned} \sum_{i=1}^n a_i^* 2^{(n-i)} &= \sum_{i=1}^n b_i^* 2^{(n-i)} \\ \sum_{i=1}^n a_i^* &= \sum_{i=1}^n b_i^* \\ a_i^* b_i^* &= 0 \text{ for all } i \\ \mathbf{b}^* &\succ \mathbf{a}^*. \end{aligned} \tag{22}$$

Let k be the smallest index in which $\mathbf{a}^*$ and $\mathbf{b}^*$ differ. Note that $a_i^* = b_i^* = 0$ for $i < k$. First, consider the case in which there is an index $j < n$ such that $a_j^* = 2$. Then, by Proposition 3 we have,

$$\sum_{i=1}^n a_i^* 2^{(n-i)} \leq 2^{n-k} + a_n \tag{23}$$

since

$$\begin{aligned} &a_i^* = 0 \text{ for } i \leq k \\ &a_i^* = 1 \text{ for } k < i < j \\ &a_j^* = 2 \\ &a_i^* = 0 \text{ for } j < i < n. \end{aligned} \tag{24}$$

On the other hand, we have

$$\begin{aligned} \sum_{i=1}^n b_i^* 2^{(n-i)} &\geq 2^{(n-k)} + D - 1 \\ &\geq 2^{(n-j)} + a_n^* + 1, \end{aligned} \tag{25}$$

since $b_k^* \geq 1$ for $k \leq j$ and $\sum_{i=1}^n b_i^* = D$, where $\sum_{i=1}^n a_i^* = \sum_{i=1}^n b_i^* = D \geq a_n^* + 2$. Inequalities (23) and (25) give us a contradiction since we should have

$$\sum_{i=1}^n a_i^* 2^{(n-i)} = \sum_{i=1}^n b_i^* 2^{(n-i)}.$$

Now let us consider the case when there is no index i with $a_i^* = 2$. Then, by Proposition 3, we have the following inequality:

$$\sum_{i=1}^{n} a_i^* 2^{(n-i)} \leq 2^{(n-k)-2} + a_n^*, \tag{26}$$

where $\sum_{i=1}^{n} a_i^* = \sum_{i=1}^{n} b_i^* = D \geq a_n^*$. As in the argument before we also have the following:

$$\begin{aligned} \sum_{i=1}^{n} b_i^* 2^{(n-i)} &\geq 2^{(n-k)} + D - 1 \\ &\geq 2^{(n-k)} + a_n^* - 1, \end{aligned} \tag{27}$$

which again leads to a contradiction. This proves that if L^+ cannot be applied on **a**, then **a** is the lexicographically largest among vectors with L leaves. □

Lemma 5. *The vector* **a** *is minimum cost vector among the vectors with L leaves if and only if it is the lexicographically largest among the vectors with L leaves.*

Proof: If **a** has the minimum cost, then L^+ cannot be applied on **a** since L^+ decreases the cost. Hence, **a** is lexicographically largest. Conversely, suppose **a** is lexicographically largest but not minimum cost, and $\mathbf{a} \succ \mathbf{b}$ is the minimum cost vector. But then L^+ can be applied on **b** to get a vector with less cost by Proposition 2 and Lemma 4. Hence, we have a contradiction. □

Remark 1. Lemma 5 does not say that for a particular function, the tree corresponding to the lexicographically largest vector has minimum cost. It may be the case that the lexicographically largest vector may not be achieved by any function. If it is achieved then it is the optimal one.

Next we show how the cost of the lexicographically largest vectors change when the number of leaves increases. In some sense, we are combining the two criteria, minimum number of leaves and lexicographically largest. We need a lemma first.

Lemma 6. *Suppose* **a** *is the lexicographically largest feasible vector with l leaves. Let $i = \min_{j<n}\{j : a_j > 0\}$. Let* $\mathbf{a}'$ *be such that $a'_j = a_j$ when $j \neq i$ or $i + 1$, $a'_i := a_i - 1$, $a'_{i+1} := a_{i+1} + 2$. Then* $\mathbf{a}'$ *is the lexicographically largest feasible vector with $l + 1$ leaves.*

Proof: Let us first observe that $\mathbf{a}'$ is a feasible vector with $l + 1$ leaves. Suppose under the conditions of the lemma, it is not the lexicographically largest vector with $l + 1$ leaves. Then there exists a feasible vector **b** with $l + 1$ leaves such that $\mathbf{b} \succ \mathbf{a}'$. Let k be the first index in which they differ, i.e. $b_k > a'_k$. Consider the following cases.

(1) If $k \le i$ and $b_k = 2$, then $\mathbf{b}' = (b_1, b_2, \ldots, b_{k-1}+1, b_k - 2, \ldots, b_n)$ is a feasible vector with l leaves and $\mathbf{b}' \succ \mathbf{a}$, which contradicts the assumption that $\mathbf{a}$ is the lexicographically largest vector with l leaves. Note that if $b_k = 1$, then $a_k = 0$. Also, we have $l+1 \ge 2+a_n$. Then we have the following inequality:

$$\sum_{i=k}^{n} b_k 2^{(n-k)} \ge 2^{(n-k)} + l. \tag{28}$$

Note that since $\mathbf{a}$ is the lexicographically largest vector with l leaves, L^+ cannot be applied on $\mathbf{a}$, and it satisfies the properties of Proposition 3. Thus we have the following:

$$\begin{aligned}\sum_{j=k}^{n} a_j 2^{(n-j)} &\le 2^{(n-k)} + a_n \\ &\le 2^{(n-k)} + l - 1.\end{aligned} \tag{29}$$

Now we have a contradiction since $\mathbf{a}'$ and $\mathbf{b}$ cannot be both feasible.

(2) If $k = i + 1$, then we have $b_{i+1} > a_{i+1} + 2$, but this cannot happen with the lexicographically largest vector by Proposition 3 and Lemma 4.

(3) If $k > i + 1$, we have the following relations:

$$\begin{aligned}\sum_{j=k}^{n} b_j 2^{(n-j)} &\ge 2^{(n-k)} + a_n - 1 \\ \sum_{j=k}^{n} a_j 2^{(n-j)} &= a_n,\end{aligned} \tag{30}$$

which again shows that $\mathbf{a}'$ and $\mathbf{b}$ cannot be both feasible.

Since we covered all cases and each case leads to a contradiction, we conclude that $\mathbf{a}'$ is the lexicographically largest vector among vectors with $l + 1$ leaves. □

Lemma 7. *The cost of the lexicographically largest vector with l leaves is not more than the cost of the lexicographically largest vector with l' leaves, where $l' > l$.*

Proof: By Lemma 6, we know how to obtain the lexicographically largest vector with $l + 1$ leaves, from the lexicographically largest vector with l leaves. To prove the lemma, it suffices to note that that operation increases the cost. If we use the notation of Lemma 6, we have the following:

$$\begin{aligned} C(\mathbf{a}') - C(\mathbf{a}) &= (i+1)2^{(n-i)} - i2^{(n-i)} \\ &= 2^{(n-i)} \\ &\geq 0. \end{aligned} \tag{31}$$

□

5. A Heuristic to find lexicographically large *BDT*s

The results of the previous section intuitively support the idea that the lexicographically large trees are candidates for “good” solutions. In this section, we propose a heuristic to find lexicographically large BDTs.

It is possible that the lexicographically largest tree for a particular function does not necessarily have the minimum number of leaves and that lexicographically largest tree may have less cost than another tree, which has fewer leaves (see (Ünlüyurt, 1999) for such examples).

We have not found any reference about finding the lexicographically largest tree for a function. Yet, one can make the following easy observation.

Proposition 4. *If the first component inspected is not in any of the shortest prime implicates or shortest prime implicants, then that tree cannot be lexicographically largest.*

For each leaf of a reasonable BDT of a monotone Boolean function, there exists a unique prime implicant or implicant of the dual whose variables are all set to 1 or 0 respectively. Furthermore, for each prime implicant or implicant of the dual of the function, there exists such a corresponding leaf. This means, for instance, if we have k prime implicants and implicants of the dual of length h, in any of the reasonable binary decision tree there should be at least k leaves at depth h or more.

Let us associate an n-vector l^f for a monotone Boolean function such that l_i^f is the total number of prime implicants and prime implicates of f of length i. Let l^* be the lexicographically largest vector among the vectors for all reasonable BDTs for f. Obviously, we have that $l^f \succ l^*$, due to the discussion above.

Let us now use this idea to find a variable to inspect next, that will likely lead to lexicographically large BDTs. Let l_i^* be the lexicographically largest vector among the vectors for all reasonable trees whose root is indexed by component i. First, observe the following proposition that follows from the discussion above of the relation between prime implicants and leaves of reasonable BDTs.

Proposition 5. *The vector* $l^{f|x_i=0} + l^{f|x_i=1} + 2\mathbf{e} \succ l_i^*$*, where* $\mathbf{e}$ *is the vector of all* 1*s.*

In other words, $l^{f|x_i=0} + l^{f|x_i=1} + 2\mathbf{e}$ is an upper bound on the vector we can obtain for any tree if we start inspection with x_i. Depending on this observation and all the results of the previous section, we propose the following heuristic for finding a minimum cost strategy.

```
LEXICO ($f$)

Input: A monotone boolean function $f$ and $f^d$ in $DNF$.
Output: A subset $I \subseteq \{1, 2, \ldots, n\}$ containing the indices of the inspected components,
    together with the values $x_i$, $i \in I$ of these components at the current state $\mathbf{x}$, and the
    value $f(\mathbf{x})$.
Step 1: Calculate for all relevant variables the vector, $l^{f|x_i=0} + l^{f|x_i=1}$.
Step 2: Choose $x_i$ for which the corresponding vector computed in Step 1 is the lex-
    icographically largest. Inspect $x_i$ and let $\alpha$ denote its value. Set $I := I \bigcup \{i\}$,
    $f := f|x_i = \alpha$. If the value of $f$ is not determined yet, return to Step 1.
```

Note that in order to compute the total expected cost of a strategy, we build the whole BDT obtained using the heuristic, which can be exponentially larger than the input. The input in this case is the DNFs of the primal and dual, which can be exponentially larger than the number of components n. In an actual problem, this blow up in sizes can be overcome by just specifying a rule to choose the next component to inspect unless the state of the system can be determined uniquely. For this reason, we are interested in the quality of the solution rather than the running time. In particular, LEXICO can determine the next component to inspect in polynomial time in the size of the input. We compare LEXICO to the exact optimal solution for cases where optimal solutions can be obtained.

The second heuristic we consider has been proposed in various papers both for the general case and for the case of identical components. We describe this, INTERSECTION SORT heuristic in detail below.

```
INTERSECTION SORT($f$)

Input: A monotone Boolean function $f$ and $f^d$ in $DNF$.
Output: A subset $I \subseteq \{1, 2, \ldots, n\}$ containing the indices of the inspected components,
    together with the values $x_i$, $i \in I$ of these components at the current state $\mathbf{x}$, and the
    value $f(\mathbf{x})$.
Step 1: Pick a shortest prime implicant of $f$, say $P^*$.
Step 2: Pick a shortest prime implicant of $f^d$, say $Q^*$.
Step 3: Pick a variable in $P^* \cap Q^*$ to inspect next, say $x_h$.
Step 4: $I = I \cup \{h\}$. Update the prime implicants of $f$ and $f^d$ by assigning the appro-
    priate value to $x_h$. If the value of $f$ cannot be determined uniquely, return to step
    1.
Step 5: Output $I$ and the correct value of $f$.
```

Let us note that there are generalizations of INTERSECTION SORT for general costs and probabilities in the literature. (See for instance (Boros and Ünlüyurt, 2000; Cox et al., 1989; Ben-Dov, 1981)). This generalization essentially picks a best prime implicant and a best prime implicant of the dual and inspects a variable in the intersection.

Note that the algorithm does not discriminate between the prime implicants with the same length. One of these prime implicants is chosen in step 1 and step 2. In addition, if there is more than one element in the intersection in step 3, the algorithm picks any one of these components.

First, we compare the performance of the heuristic algorithm LEXICO for 3-level deep SPSs with identical components. ($p = 0.5$) For such systems optimal algorithms are provided in Boros and Ünlüyurt (2000). We also compute the expected cost of the strategy induced by the INTERSECTION SORT algorithm in each case. The parameters we used to create random instances are explained in Table 1. The global problem is parallel in each case.

The results are presented in Table 2. Each row corresponds to 20 random instances with the indicated parameters. Number of variables vary for each instance with respect to the parameters.

Similar results are shown in Table 3. For this table, we created random instances of *3-level*

Table 1. Notation for experiments.

NS	number of subsystems
NSS	maximum number of subsubsystems
NP	maximum size of parallel sub-subsubsystems
N	number of problems (out of 20) that could be solved without memory problem.
MIN VARS	minimum number of variables out of the N problems
MAX VARS	maximum number of variables out of the N problems
OPT	number of times heuristic was optimal
MAX	maximum percentage error of heuristic
AVG	average percentage error of heuristic

Table 2. LEXICO and INTERSECTION SORT versus Optimal Solution for 3-level deep SPSs.

						LEXICO			INTERSECTION SORT		
NS	NSS	NP	*N*	MIN VARS	MAX VARS	OPT	MAX	AVG	OPT	MAX	AVG
2	4	3	20	6	22	17	3.57	0.26	16	7.02	0.83
3	3	3	20	11	24	18	0.76	0.05	11	4.94	1.1
3	4	3	20	11	32	17	1.32	0.11	13	6.05	0.96
3	4	4	20	15	35	16	1.48	0.17	13	11.16	1.7
4	3	3	19	15	29	19	0	0	10	4.45	1.01
4	4	3	17	13	35	15	1.42	0.14	8	7.72	1.42
5	5	3	13	20	30	12	0.18	0.01	4	3.89	1.18

Table 3. LEXICO and INTERSECTION SORT versus Optimal Solution for 3-level deep SPSs.

	LEXICO			INTERSECTION SORT		
NUM OF VARS	OPT	AVG	MAX	OPT	AVG	MAX
16	20	0	0	14	0.89	6.07
18	20	0	0	10	1.93	10.86
20	19	0.06	1.21	11	1.7	9.18
22	19	0.06	1.11	9	1.94	8.94
24	18	0.04	0.74	8	1.46	6.38
26	18	0.01	0.16	11	0.94	6.36
28	20	0	0	10	1.7	10.09
30	17	0.01	0.1	8	1.63	9.43
32	18	0.001	0.01	9	1.22	8.14

deep SPSs consisting of certain number of variables shown in column 1. These 3-level deep SPSs consist of at most 5 *subsystems*. Each subsystem consists of at most 5 *subsubsystems* and finally the parallel subsystems consist of up to 7 components. Other columns are labeled in the same manner as Table 2.

LEXICO performed very well in the case of 3-level deep SPSs with identical components and $p = 0.5$. Another class of functions for which optimal strategies can be obtained for identical components, is the class of threshold functions (for details see (Boros and Ünlüyurt, 1999)). We experimented on two sets of randomly generated functions with 12, 15 and 18 variables. For one of the sets, the weights are drawn uniformly from the interval [2,20], for the second set from [2,10]. For both sets of randomly generated problem instances, we tried 5 threshold values. Namely, if $T = \sum_{i=1}^{n} w_i$ where n is the number of variables, then we tried 0.1T, 0.25T, 0.5T, 0.75T, and 0.9T for the threshold values. So in total, we had 200 random instances. We don't need to display the results in a table since in all of these 200 instances LEXICO found the optimal solution. Let us note that for the case when weights are drawn from [1,10], $t = 0.5T$ and the number of variables is 18, the number of prime implicants of f and f^d are both over 20,000.

6. Conclusion and future directions

In this study, sequential testing of systems that consist of identical components is considered. It turns out that this problem has various quite unrelated application areas. The notion of lexicographically large BDTs is introduced and it is shown that lexicographically large BDTs tend to have less cost. A heuristic is proposed to find lexicographically large BDTs and the computational results show that strategies produced by this heuristic are quite good.

One question that may be investigated further is to characterize subclasses of problems when LEXICO is optimal. Another direction to explore is to consider entropy based heuristic algorithms. Also, apart from the notion of lexicographically large BDTs, one can observe that optimal algorithms are provided for quite special cases. For instance, no results are known for general SPSs with identical components. Another extension of this problem is when there are precedence constraints between the tests. For certain type of precedence constraints and only for the simple series system, optimal algorithms are provided in Garey (1973) for general costs and probabilities. There is a complete review on such issues in Ünlüyurt (2004).

Acknowledgment

I would like to thank Prof. Endre Boros for his guidance and comments for this paper.

References

L. Arseneau, "Optimal testing strategies for s,t-series parallel systems," Master's thesis, Combinatorics and Optimization, University of Waterloo, 1996.

Y. Ben-Dov, "Optimal testing procedures for special structures of coherent systems," *Management Science*, vol. 27, no. 12, pp. 1410–1420, 1981.

E. Boros and T. Ünlüyurt, "Diagnosing double regular systems," *Annals of Mathematics and Artificial Intelligence*, vol. 26, pp. 171–191, 1999.

E. Boros and T. Ünlüyurt, "Sequential testing of series-parallel systems of small depth," in M. Laguna and J.L.G. Velarde (eds.), *OR Computing Tools for the New Millennium*, Kluwer, 2000, pp. 39–74.

Y. Breitbart and A. Reiter, "A branch-and-bound algorithm to obtain an optimal evaluation tree for monotonic boolean functions," *Acta Informatica*, vol. 4, pp. 311–319, 1975.

Y. Breitbart and S. Gal, "Analysis of algorithms for the evaluation of monotonic Boolean functions," *IEEE Transactions on Computers*, vol. 27, no. 11, pp. 1083–1087, 1978.

Y. Breitbart and A. Reiter, "Algorithms for fast evaluation of boolean expressions," *Acta Informatica*, vol. 4, pp. 107–116, 1975.

M. Chang, W. Shi, and W.K. Fuchs, "Optimal diagnosis procedures for k-out-of-n structures," *IEEE Transactions on Computers*, vol. 39, no. 4, pp. 559–564, 1990.

L.A. Cox, S.Y. Chiu, and X. Sun, "Least-cost failure diagnosis in uncertain reliability systems," *Reliability Engineering and System Safety*, vol. 54, pp. 203–216, 1996.

S.O. Duffuaa and A. Raouf, "An optimal sequence in multicharacteristics inspection," *Journal of Optimization Theory and Applications*, vol. 67, no. 1, pp. 79–87, 1990.

M.R. Garey, "Optimal task sequencing with precedence constraints," *Discrete Mathematics*, vol. 4, pp. 37–56, 1973.

R. Greiner, "Finding optimal derivation strategies in redundant knowledge bases," *Artificial Intelligence*, vol. 50, pp. 95–115, 1990.

E. Gudes and A. Hoffman, "A note on an optimal evaluation of boolean expressions in an online query system," *Short Communications, Artificial Intelligence/Language Processing*, vol. 22, no. 10, pp. 550–553, 1979.

J.Y. Halpern, "Evaluating Boolean function with random variables," *International Journal of Systems Science*, vol. 5, no. 6, pp. 545–553, 1974.

M.Z. Hanani, "An optimal evaluation of boolean expressions in an online query system," *Short Communications, Artificial Intelligence/Language Processing*, vol. 20, no. 5, pp. 344–347, 1977.

L.A. Cox Jr., Y. Qiu, and W. Kuehner, "Heuristic least-cost computation of discrete classification functions with uncertain argument values," *Annals of Operations Research*, vol. 21, pp. 1–21, 1989.

H.A. Simon and J.B. Kadane, “Optimal problem-solving search: All-or-none solutions,” *Artificial Intelligence*, vol. 6, pp. 235–247, 1975.

D.E. Smith, “Controlling backward inference,” *Artificial Intelligence*, vol. 39, pp. 145–208, 1989.

T. Ünlüyurt, Sequential Testing of Complex Systems. PhD thesis, Rutgers University, 1999.

T. Ünlüyurt, “Sequential testing of complex systems: A review,” *Discrete Applied Mathematics*, vol. 142, nos. (1–3), pp. 189–205, 2004.