Ann Oper Res (2011) 187:89–119
DOI 10.1007/s10479-010-0799-6

Optimal sequential inspection policies

Endre Boros · Noam Goldberg · Paul B. Kantor · Jonathan Word

Published online: 22 October 2010

Abstract We consider the problem of combining a given set of diagnostic tests into an inspection system to classify items of interest (cases) with maximum accuracy such that the cost of performing the tests does not exceed a given budget constraint. One motivating application is sequencing diagnostic tests for container inspection, where the diagnostic tests may correspond to radiation sensors, document checks, or imaging systems. We consider mixtures of decision trees as inspection systems following the work of Boros et al. (Nav. Res. Logist. 56:404–420, 2009). We establish some properties of *efficient* inspection systems and characterize the optimal classification of cases, based on some of their test scores. The measure of performance is the fraction of all cases in a specific class of interest, which are classified correctly. We propose a dynamic programming algorithm that constructs more

Funding by the Domestic Nuclear Detection Office (DNDO), of the Department of Homeland Security, through NSF Grants # CBET-0735910 and by Department of Homeland Security Grant/Contract #2008-DN-077-ARI003-0, the Science and Technology Directorate, Office of University Programs, by the National Science Foundation #SES 0518543 3/3 and by Office of Naval Research #DOD-DON-ONR-N00014-071-0150, # DOD-DON-ONR-N00014-07-1-0299 are all most gratefully acknowledged. The authors would like to thank DIMACS, and members of the DyDAn DHS Center of Excellence for stimulating discussions. We also thank Nir Halman and an anonymous referee for suggestions to improve the presentation.

E. Boros · J. Word
RUTCOR, Rutgers University, 640 Bartholomew Road, Piscataway, NJ 08854-8003, USA

E. Boros
e-mail: boros@rutcor.rutgers.edu

J. Word
e-mail: jword@rutcor.rutgers.edu

N. Goldberg (✉)
Faculty of Industrial Engineering and Management, Technion—Israel Institute of Technology, Haifa 32000, Israel
e-mail: noamgold@tx.technion.ac.il

P.B. Kantor
SC&I, Rutgers University, Piscataway, NJ, USA
e-mail: paul.kantor@rutgers.edu

complex policies by iteratively prefixing devices to a subset of policies and thereby enumerating all of the efficient (i.e., undominated) inspection policies in the two dimensional cost-detection space. Our inspection policies may sequence an arbitrary number of tests and are not restricted in the branching factor. Our approach directly solves the bi-criterion optimization problem of maximizing detection and minimizing cost, and thus supports sensitivity analysis over a wide range of budget and detection requirements.

Keywords Dynamic programming · Sequential inspection · Bi-objective optimization · Port security

1 Introduction

Inspection systems are mixtures of decision trees whose nodes are diagnostic tests and whose leaves represent an assignment to actions. Faced with the problem of identifying a threat, we may inspect some items of interest, and seek a system whose output is the action of raising an alarm, or performing an expensive thorough examination, for some of the given items. We are thus interested in deciding which of the items should trigger an alarm. It is presumed that each test deployed involves some operating cost. The item classification decision and subsequent action may incur an additional cost that is associated with the rate of false positives (also known as type I errors).

The operating characteristics of a test can be expressed by a Receiver Operating Characteristic (ROC) curve (Fawcett 2006) and by the cost parameters. Together, these completely characterize the test and the cost/benefit of its deployment. The ROC curve expresses the inspection system's detection rate as a function of the rate of false positives. Finally, the *cost-detection curve* , which is a fundamental mathematical entity of interest in our analysis, can be constructed by adding the operating cost to the cost of false positives of each point on the ROC curve.

A motivating application is the detection of nuclear and other potentially harmful material that could be smuggled into the country. At present, about 90% of all cargo arriving to the U.S. by sea arrives in container shipments. Each and every container requires the transmission of detailed information about its contents and origin before arriving at a seaport. Only a fraction of the containers go through more detailed testing such as X-ray and γ-ray scanning, and even fewer go through more detailed manual inspection (see Frittelli 2005 for more details). In general one can not afford to open and manually inspect each and every container coming into the country through ports or border crossings. Imperfect information about the contents of every package, vehicle or container is given by tests for the presence of nuclear material. The tests may include different types of radiation sensors as well as simple document checks. Although both basic and applied research is underway for introducing less expensive, less intrusive, and more effective technologies for scanning containers (see for example Slaughter et al. 2007), the underlying assumption remains that perfect information may not be available. In this paper, we seek to use the costly and imperfect information given by the available tests as efficiently as possible.

In principle, the operating goal is to detect 100% of a particular hazardous material being smuggled into the country. In practice, most sensors will not detect 100% of the radioactive material: physical equipment may malfunction, personnel can be negligent and adversaries may learn new ways to conceal nuclear material. Thus, policy makers face the difficult task of designing and deploying the best possible (imperfect) security inspection systems. The decision maker's problem includes both deciding the budget for inspection, allocating the budget to specific inspection policies, and implementing policies as physical inspection systems. The strategic decision of allocating the budget may entail the subjective estimation of two important parameters that are difficult to estimate: the prior probability of a dangerous item and the cost of failing to detect a dangerous item. Our approach offers an alternative for avoiding the preliminary computation of these parameters. In particular, we would like to determine the set of policies yielding the highest rate of detection for any possible cost of failing to detect a dangerous item. Our approach is advantageous to strategic decision makers who may use more information in choosing a particular budget. On the other hand, the advantage to a tactical decision maker is clearly the ability to determine the details of a multi-level inspection system with highest detection rate given a particular budget. The latter decision maker may also determine how these details (e.g., sequences of tests) may vary in response to small changes in budget allocation. To this end we propose an algorithm that finds an entire *efficient frontier* of inspection policies in the two-dimensional cost-detection space providing an optimal policy for each conceivable value of the budget.

Each inspection system can be represented as a tree as shown in Fig. 3. The number of branches at each node determines the complexity of the tree, and is determined by the maximum number of different possible actions that may follow the application of the corresponding test. The problem of finding optimal inspection systems corresponding to binary decision trees has been considered by Stroud and Saeger (2003), and Madigan et al. (2007). Stroud and Saeger (2003) have shown that the total number T_N of possible binary decision trees using N tests is given by the recursive formula $T_N = 2 + N(T_{N-1})^2$, where $T_0 = 2$. Even for binary decision trees the total number of trees becomes extremely large, so that, for example, $T_4 = 1,079,779,602$. For general decision trees with branching factor k (but two terminal actions), the formula can be generalized, yielding:

$$T_N^{(k)} = 2 + N\left(T_{N-1}^{(k)}\right)^k. \tag{1}$$

The principal result of this paper is an efficient computation scheme that finds the optimal decision tree mixtures for a range of budget values, while avoiding a complete enumeration of all decision trees. Boros et al. (2009) consider optimal decision tree inspection systems with an arbitrary branching factor, satisfying a budgetary constraint as well as other performance constraints. They present a large scale linear programming formulation for computing the optimal system. For N tests and a fixed branching factor k, the number of variables in the LP formulation is $O(N!)$, which remains exponentially large, yet is substantially smaller than the number of decision trees. The LP approach is effective for problems of sequencing as many as 5 sensors (i.e., tests) for container inspection. Future technologies being considered may require inspection systems to use a larger number of sensors. Moreover, multi-channel sensors represent a large number of conceptually distinct tests, as the sensor readings can be matched to any of several profiles of interest.

Finally, let us note that our methodology is applicable to a wide range of problems including screening for explosives, screening travelers at borders for drugs, screening at public events for weapons, and testing the public for diseases. Of particular interest in explaining and estimating the value of some of the parameters of our model in the case of the aviation industry is the work of Jacobson et al. (2005). In the next section we will describe some of the background for our contribution and the required definitions, including precise definitions of inspection devices and policies. We proceed to characterize optimal inspection policies, in Sect. 4, where we prove a monotonicity property of such policies. In Sect. 5 we propose an optimization formulation and an efficient algorithm for the problem of constructing optimal policies from a given diagnostic test and a set of available actions. In Sect. 6 we make use of this algorithm as a sub-procedure in a dynamic algorithm for enumerating all *efficient* policies. We also suggest an upper bound on the number of efficient policies. Finally, we conclude after discussing our computational experiments (Sect. 8), comparing the computational performance and detection performance of our algorithms with the previous work of Boros et al. (2009).

2 Background: cases, tests, devices, actions and policies

2.1 Definitions

Let us first introduce our basic terminology and notation. We consider a set of *cases*, some of which may require more careful inspection. In homeland security applications, cases may be containers arriving at entry ports from foreign countries, trucks arriving at border crossings, or foreign visitors lining up for immigration at airports, etc. Each case belongs to a certain *class* and we assume that cases of a given class require a certain predefined *terminal action*. In this study we assume that cases are divided into two classes, *good* or *bad*, good cases should be *released* without further delays, while those suspected to be bad are subjected to a thorough manual *inspection*.

In order to detect the class of a case, we make use of a set of *tests* (e.g., sensors), each of which, when applied, provides (imperfect) indication of the class to which the case belongs. The set of tests is denoted by $\mathcal{T}$, their number by $N = |\mathcal{T}|$, and $c(t)$ denotes the cost of applying test $t \in \mathcal{T}$ to a case.[1] A test generates a *label* (also called a score or a reading), which gives some indication about the likelihood that the case belongs to the class of bad items.

Applications of tests are costly: inspecting an innocent good case ties up resources and may delay trade, for example, while releasing a bad case may have serious consequences later. Our main problem is to devise a plan using the imperfect and costly tests in order to help us make decisions and execute actions so that the overall expenses are minimized. To be able to formulate the problem, we first need to describe the information which pertains to tests and actions in more detail.

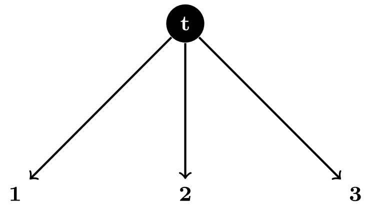
We shall view a subtree with nodes corresponding to tests, more abstractly, as a *device*, filtering an input stream of cases, each of which is assigned a label. Formally, we associate to each test $t \in \mathcal{T}$, the set $\mathcal{L} = \mathcal{L}(t)$ of possible *labels*. For each label $i \in \mathcal{L}$ we denote by $b_i \geq 0$ and $g_i \geq 0$ the probabilities of label $i \in \mathcal{L}$ given the class of a case is, respectively, bad or good; these can be estimated by the fractions of, respectively, bad or good containers that are expected to receive label i in the long run. Since every test must assign a label to every case, we have

$$\sum_{i \in \mathcal{L}} b_i = \sum_{i \in \mathcal{L}} g_i = 1. \tag{2}$$

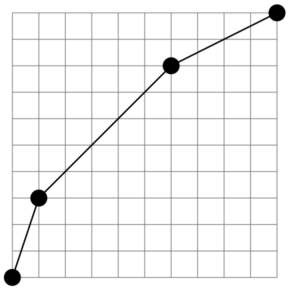
When there is any ambiguity, we will specify the test $t \in \mathcal{T}$, and refer to these parameters as $\mathcal{L}(t)$, $b_i(t)$ and $g_i(t)$, respectively.

Let us note that although a label could be an arbitrary description of a category, in practice it will often be an integer score (for example the number of suspicious entries in a shipping

[1] In order to capture fixed costs associated with test, the unit cost $c(t)$ may be the long run average cost, including the amortization of fixed costs.



(a) Test **t** with three labels, $\mathcal{L} = \{\mathbf{1}, \mathbf{2}, \mathbf{3}\}$, characterized by $(b_1, b_2, b_3) = (0.3, 0.5, 0.2)$ and $(g_1, g_2, g_3) = (0.1, 0.5, 0.4)$.



(b) The ROC describing the performance of test **t**. We have $(b_1, g_1) = (0.1, 0.3)$, $(b_1 + b_2, g_1 + g_2) = (0.6, 0.8)$ and $(\sum_{i=1}^{3} b_i, \sum_{i=1}^{3} g_i) = (1, 1)$.

Fig. 1 An example for an elementary test and its ROC

manifest), or a real number (such as in the case of single channel radiation sensors). Accordingly the set $\mathcal{L}(t)$ could be either finite and/or infinite for some of the tests $t \in \mathcal{T}$. Though much of the mathematical analysis we apply does not depend on the finiteness of the label sets, the resulting algorithms and specifically their termination does. Our approach for the case of continuous labels is to discretize the labels having a continuous range into a finite set of labels $\mathcal{L}$. In what follows we assume that the set of labels is finite.

We will index labels in such a way that

$$\frac{b_1}{g_1} \geq \frac{b_2}{g_2} \geq \cdots \geq \frac{b_L}{g_L} \tag{3}$$

holds for all tests, where $L = |\mathcal{L}|$ is the number of labels. Given this sorting of the labels, the sequence of cumulative probabilities, for each $i \in \mathcal{L}$, can be represented as points in a two dimensional graph:

$$\left(\sum_{j \leq i} b_j, \sum_{j \leq i} g_j\right),$$

in order to describe the performance of a test. Adding the point $(0, 0)$, the piecewise linear curve determined by these points is known as the ROC of the test. See Fig. 1 for a simple example with three labels. Note that due to the ordering (3) and the equalities (2), the ROC curve is concave, starting at $(0, 0)$ and ending at $(1, 1)$.

Let us note that typically the outcome of applying a particular test is a random variable. We assume in this study that the randomness of the test results essentially originates from differences in exogenous properties of cases, and only to a negligible extent from measurement errors. Consequently, we assume that repeating the same test on the same case will not result in a different labeling of the case. For example, if the test is "checking the shipping manifest", then we will find the same suspicious entries, no matter how many times we run the test for the same container. Similarly, if a radiation detector is triggered by a container, it will most likely be triggered on a second examination as well. That is to say, several containers with exactly the same contraband may give different sensor readings because of variations in other (shielding) cargo, or debris left from a previous shipment. But

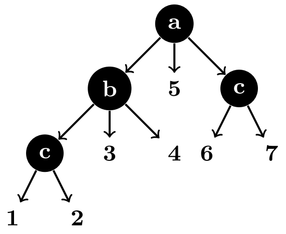
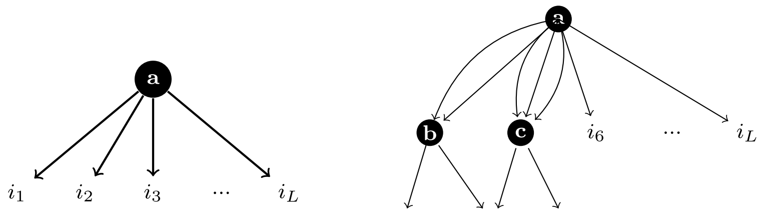


Fig. 2 An example of a complex system that fuses multiple tests. The labels of each individual test (as well as the labels of the complex device) are numbered, in increasing order, from left to right



(a) An example of a general elementary test with L labels.

(b) A compound device using the L-label test as the root.

Fig. 3 An elementary test with L labels and a compound device using the test as the root

for a given container, these factors remain the same no matter how many times we repeat the measurement.

Another important assumption in this study is the stochastic independence of the different elementary tests. This is a less self-evident condition, but, in practice, most studies in this area make this assumption (see for example Stroud and Saeger 2003; Madigan et al. 2007; Ramirez-Marquez 2008). The technical meaning of this assumption is that if we apply two tests: t and t' to all cases, then, for example, the fraction of good cases receiving label $i \in \mathcal{L}(t)$ from test t, and label $i' \in \mathcal{L}(t')$ from test t' is $g_i(t)g_{i'}(t')$. This "product rule" allows us to estimate the result of applying a sequence of tests to the stream of cases, regardless of the results of preceding tests. This assumption is important for achieving the computational speedup that we report in this paper.

Generalizing the notion of elementary tests, we consider more complex systems that fuse multiple tests, and together filter a stream of cases, assigning to each one a label. Figure 2 shows such a complex system, combining three elementary tests $\mathcal{T} = \{a, b, c\}$, where a and b each have three labels, while c has only two. The tree in the figure represents the process in which we first apply test a, and, if test a labels the case by 3 then we apply test c, etc. The resulting complex system has seven labels, and, for example, the device will label a case with label 6, if test a results in label 3, and the following test c assigns it to label 1.

We are now ready to formally define such complex systems, which we shall call devices. The result of applying a device to a set of cases, of which some are good and some are bad is that some fraction of each will be assigned to each of the labels. The effect of using a device, other than its cost, is completely described by a collection of fractions, corresponding to posterior probabilities of the labels, and subsets of tests describing each label. (See e.g., Fig. 3.)

Definition 1 A device D is defined as a set of triplets

$$D = \{(b_i, g_i, T_i) \mid i \in \mathcal{L}\}, \tag{4}$$

where $\mathcal{L}$ is the set of (output) labels of D, the sets $T_i \subseteq \mathcal{T}$ are the subsets of elementary tests applied to cases that end up to be labeled by i, and $b_i \geq 0$ and $g_i \geq 0$ are the probabilities of receiving label $i \in \mathcal{L}$, given that a case is, respectively, bad or good.

When ambiguity could arise, we shall refer to these parameters of a device D explicitly by $\mathcal{L}(D)$, $b_i(D)$, $g_i(D)$, and $T_i(D)$ for all $i \in \mathcal{L}(D)$, and we let $L(D) = |\mathcal{L}(D)|$ in order to denote the number of labels of the device D.

Furthermore, following our assumption of stochastic independence of elementary tests, and by equation (2) every device must satisfy the following relations:

$$b_i(D) = \prod_{t \in T_i(D)} b_{i(t)}(t), \quad \text{and} \quad g_i(D) = \prod_{t \in T_i(D)} g_{i(t)}(t), \quad \text{for all } i \in \mathcal{L}(D),$$
$$\text{and} \quad \sum_{i \in \mathcal{L}(D)} b_i(D) = \sum_{i \in \mathcal{L}(D)} g_i(D) = 1. \tag{5}$$

Let us note that every label $i \in \mathcal{L}(D)$ corresponds to a path in the decision tree which corresponds to D, and hence to a unique label $i(t) \in \mathcal{L}(t)$ for each test $t \in T_i(D)$ along this path.

To complete the description of a device D we must add its expected (per unit) operating cost, when applied to a given stream of cases. This can be expressed as

$$c(D) = \sum_{i \in \mathcal{L}} (\pi b_i + (1-\pi) g_i) c(T_i), \tag{6}$$

where π denotes the a priori probability that a case is bad, and $c(S) = \sum_{t \in S} c(t)$ for a subset of tests, $S \subseteq \mathcal{T}$.

An inspection *policy* is a device together with assignment of actions to its labels. The cases which are assigned to a label, will then be subjected to the corresponding action. In this paper we concentrate on two terminal actions $A = \{R, I\}$. Action R (release) corresponds to not doing any further checking. In the container inspection application, we release a container when it is considered harmless. Action I (inspect), on the other hand, is appropriate when we are "sufficiently" suspicious about the case, and execute a lengthy and detailed manual inspection; at the end of which we assume that the class of a case is determined with absolute certainty. To each terminal action $\alpha \in A$ we associate its *cost* $C(\alpha)$ and its *detection rate* $\Delta(\alpha)$. The cost[2] is normalized to in order to express the expense of executing the action per unit or case being inspected. The detection rate of I is assumed to be 1, and thus the detection rate of a policy, in general, will equal the fraction of bad cases that are assigned to the action I. We assume that action R has no cost, while we take the cost of I as our unit of measurement. Thus

$$\begin{aligned} C(R) &= 0, \qquad C(I) = 1, \\ \Delta(R) &= 0, \qquad \Delta(I) = 1. \end{aligned} \tag{7}$$

Since action I reveals all bad cases to which it is applied, no sensible policy can have a cost higher than I. Otherwise we would always replace that policy with I, and achieve better

[2]In most applications this cost depends not only on the operating cost of performing the manual inspection, but also an additional cost which results from false positives, *i.e.*, the collateral damage imposed by inspection of harmless cases.

performance at a lower cost. Therefore, we may also assume $0 \le c(t) < 1$ for all (useful) tests $t \in \mathcal{T}$.

Let us add that there could be more than these two terminal actions, and in some applications those may arise naturally. Our analysis and results generalize completely for the case of more than two terminal actions. We actually use this fact to present our recursion-based solution. Before that, however, let us formally define what is meant by an inspection *policy*:

Definition 2 A policy P is a pair $P = (D, x)$, where D is a device and $x : \mathcal{L}(D) \times A \to [0, 1]$ is a weighted mapping of the device's labels into the set of actions. More precisely, for every label $i \in \mathcal{L}(D)$ and action $\alpha \in A$ the value $x(i, \alpha)$, with $0 \le x(i, \alpha) \le 1$, is the fraction of cases labeled by i that is assigned to be processed by action α. This mapping must satisfy

$$\sum_{\alpha \in A} x(i, \alpha) = 1, \quad \text{for all } i \in \mathcal{L}(D). \tag{8}$$

This definition allows for a choice of a random mixture of assignment into actions, such as $x(i, I) = \gamma$ and $x(i, R) = 1 - \gamma$ for some label $i \in \mathcal{L}(D)$. For a budget that is insufficient for applying I to all cases it may be optimal to choose such a random mixture of assignments of labels to actions. For example for a trivial device $D = \{(1, 1, \emptyset)\}$, and $A = \{I, R\}$, the assignment $x(1, I) = x(1, R) = \frac{1}{2}$ achieves the highest possible detection rate for a budget of $B = \frac{1}{2}$. In general, under a budget constraint the assignment to a mixture of actions may only enhance the detection performance compared with a pure assignment, $x(i, \alpha) \in \{0, 1\}$, for all $i \in \mathcal{L}(D)$ and $\alpha \in A$. Moreover, in real life applications the budget may simply not be large enough to treat all cases with the action I; for an elaborate discussion of these arguments see Kantor and Boros (2010).

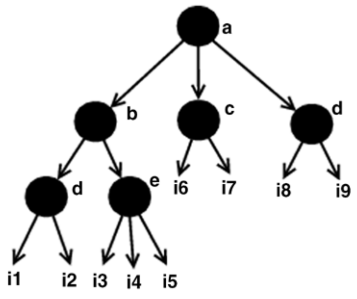
Having defined what we mean by a policy, we must also specify the cost and performance of the policy. For a policy P we can compute a unit cost $C(P)$, which is the expected cost per case of applying policy P, and a the detection rate $\Delta(P)$ which is the expected fraction of the bad cases that will be identified (that is, assigned to action I) by policy P. The impact of this cost will depend on the fraction of cases that are good and that are assigned to action I (i.e., to be inspected). With a action set $A = \{R, I\}$ and parameters defined as in (7), we can write the cost and detection of the policy as:

$$\begin{aligned} C(P) &= \sum_{i \in \mathcal{L}(D)} \pi b_i(D)[c(T_i(D)) + x(i, I)C(I)] + (1 - \pi)g_i(D)[c(T_i(D)) + x(i, I)C(I)] \\ &= c(D) + \sum_{i \in \mathcal{L}(D)} \pi b_i(D)x(i, I) + (1 - \pi)g_i(D)x(i, I), \end{aligned} \tag{9}$$

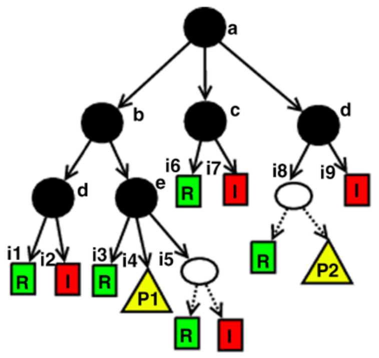
and

$$\Delta(P) = \sum_{i \in \mathcal{L}(D)} b_i(D)x(i, I)\Delta(I) = \sum_{i \in \mathcal{L}(D)} b_i(D)x(i, I). \tag{10}$$

Next we note that, in fact, terminal actions can be viewed as special cases of policies, where D is a device with one output label and an empty set of tests. Similarly, policies may be viewed as (non-terminal) actions, in the sense that they specify precisely what to do with each of the cases, although it may involve some testing, followed by a terminal action. Thus, we will view Definition 2 as a recursive structure, where the set A may involve some policies, as well as the terminal actions. Since we do not gain any additional information by repeating a test, we impose some further restrictions on the action mapping



(a) An example device. The set of labels is $\mathcal{L} = \{i_1, i_2, ..., i_9\}$. The set of tests associated with each label is simply the set of test nodes along the path, *e.g.*, $T_{i_1} = \{a, b, d\}$.



(b) The corresponding policy. Triangles represent policies. Rectangles represent terminal policies. Ellipses represent an assignment of a label that mixes two or more policies.

Fig. 4 An example device and a corresponding policy

x of policy (D, x). Let us denote by $T(P)$ the set of tests involved in policy P; we define $T(I) = T(R) = \emptyset$. We shall call a mapping x *feasible* in Definition 2 if it satisfies the further condition:

$$x(i, p) = 0 \quad \text{whenever } T_i(D) \cap T(p) \neq \emptyset, \quad \text{for all } i \in \mathcal{L}(D) \text{ and } p \in A. \tag{11}$$

Henceforth, we consider Definition 2 extended so that the set A of actions may contain a finite set of (complex) policies as well as the terminal actions, and also require the corresponding action mapping x to satisfy the conditions (11). See Fig. 4 for an example of a general policy, composed of a top level device, and a set A involving both terminal actions and other policies.

2.2 Policy mixing

In the above definition of a policy we allowed a mixed assignment of labels to actions. Although we do not explicitly model our problem as a game, our decision-analytic approach to inspection recognizes that in practice it may be difficult to reliably model the adversary's utility function as well as the potentially large set of strategies. Nevertheless, game theoretic models of inspection that do not address the aspect of sequencing the tests have been suggested earlier; see for example Maschler (1966), and a survey by Avenhaus et al. (1998), and a recent model suggested in the context of port security by Bier and Haphuriwat (2010). Hypothetically, within a game theoretic framework our mixed assignment into actions would correspond to mixed strategies. Even if no information is available about the possible moves of the adversary, the advantage of a mixed assignment into actions is also evident in the decision-analytic optimization approach, where finding the optimal deterministic policy is an integer programming problem, while finding a mixed policy is the continuous relaxation of that problem. Thus, mixed policies may achieve a superior detection rate for each budget value compared with deterministic (pure) policies. Intuitively, in practical settings, our mixed policies are also preferable because they should be more difficult to guess by an adversary.

In order to provide a more general definition of policy mixtures we introduce notion for random mixture of devices; given two devices $D_1 = \{(b_i^1, g_i^1, T_i^1) \mid i \in \mathcal{L}(D_1)\}$, $D_2 =$

$\{(b_i^2, g_i^2, T_i^2) \mid i \in \mathcal{L}(D_2)\}$, and a probability $\gamma \in [0,1]$, we can define a new device $D_3 = \{(b_i^3, g_i^3, T_i^3) \mid i \in \mathcal{L}(D_3)\}$, in which, with probability γ we screen incoming cases using device D_1, and with probability $1-\gamma$ we screen them using device D_2. Both the label probabilities and the cost are linear in γ so that

$$\mathcal{L}(D_3) = \{(1,j) \mid j \in \mathcal{L}(D_1)\} \cup \{(2,j) \mid j \in \mathcal{L}(D_2)\}, \quad \text{and} \tag{12a}$$

$$b_i^3 = \begin{cases} \gamma b_j^1 & \text{if } i = (1,j),\, j \in \mathcal{L}(D_1), \\ (1-\gamma) b_j^2 & \text{if } i = (2,j),\, j \in \mathcal{L}(D_2), \end{cases} \tag{12b}$$

$$g_i^3 = \begin{cases} \gamma g_j^1 & \text{if } i = (1,j),\, j \in \mathcal{L}(D_1), \\ (1-\gamma) g_j^2 & \text{if } i = (2,j),\, j \in \mathcal{L}(D_2), \end{cases} \quad \text{and} \tag{12c}$$

$$T_i^3 = \begin{cases} T_j^1 & \text{if } i = (1,j),\, j \in \mathcal{L}(D_1), \\ T_j^2 & \text{if } i = (2,j),\, j \in \mathcal{L}(D_2), \end{cases} \quad \text{for all } i \in \mathcal{L}(D_3). \tag{12d}$$

Now we can show that random mixing of policies is a meaningful operation, just as in the case of devices. Let us now fix the action set A. If $P_1 = (D_1, x_1)$ and $P_2 = (D_2, x_2)$ are two policies and $\gamma \in [0,1]$ is a probability of applying P_1, then the policy $P_3 = \gamma P_1 + (1-\gamma) P_2$ can be defined as a pair (D_3, x_3) where D_3 is defined by (12), and

$$x_3(i,p) = \begin{cases} \gamma x_1(j,p) & \text{if } i = (D_1, j),\, j \in \mathcal{L}(D_1), \\ (1-\gamma) x_2(j,p) & \text{if } i = (D_2, j),\, j \in \mathcal{L}(D_2), \end{cases}$$

for all policies $p \in A$ and all $i \in \mathcal{L}(D_3)$.

We can also note that for such random mixing of policies the costs and the detection rates are convex combinations given by:

$$\begin{aligned} C(P_3) &= \gamma C(P_1) + (1-\gamma) C(P_2), \\ \Delta(P_3) &= \gamma \Delta(P_1) + (1-\gamma) \Delta(P_2). \end{aligned} \tag{13}$$

We say that a policy $P = (D, x)$ is a *deterministic policy*, if for all $i \in \mathcal{L}(D)$ and all $p \in A$ we have $x(i,p) \in \{0,1\}$; by (8) such a policy assigns exactly one action to each label. Although there are many possible policies, as long as the set of terminal actions is finite, and the number of labels and tests for each device are finite, then all policies are random mixtures of a finite number of deterministic policies.

Definition 3 Given a set of policies $\mathcal{P}$, let us denote by $\text{MIX}(\mathcal{P})$ the set of policies obtainable from $\mathcal{P}$ by mixing, that is

$$\text{MIX}(\mathcal{P}) = \left\{ \sum_{P \in \mathcal{P}} \lambda_P P \,\middle|\, \lambda_P \ge 0 \text{ for all } P \in \mathcal{P}, \text{ and } \sum_{P \in \mathcal{P}} \lambda_P = 1 \right\}.$$

2.3 Policy domination

Another important basic notion in our analysis is the *domination* relation between policies.

Definition 4 Given two policies, P_1 and P_2, we say that P_2 dominates P_1, if $T(P_2) \subseteq T(P_1)$, $C(P_2) \le C(P_1)$ and $\Delta(P_2) \ge \Delta(P_1)$. If any of these inequalities are strict then we say that P_2 strictly dominates P_1.

Our definition of domination ensures that if P_1 is dominated by P_2, then we are better off using P_2 than using P_1; in any case where P_1 is used (even as a part of a more complex policy), we can simply replace it by P_2 in order to meet or exceed the same detection performance without increasing the expected cost of inspection.

Let us observe a few more easy facts about mixtures of policies. First we show that every undominated policy corresponds to some mixture of at most two deterministic policies.

Proposition 1 *Let $Q = \sum_{j=1}^{q} \lambda_j S_j$ be a mixture of policies in $\mathcal{S} = \{S_j \mid j = 1, \ldots, q\}$, such that $\lambda_j > 0$ for all $j = 1, \ldots, q$, $q \geq 3$, $\sum_{j=1}^{q} \lambda_j = 1$, and the points $(C(S_j), \Delta(S_j))$, for $j = 1, \ldots, q$, are in general position. Then Q is strictly dominated by another mixture $P \in \mathrm{MIX}(\mathcal{S})$ of the policies in $\mathcal{S}$.*

Proof Let us denote $K = \mathrm{conv}\{(C(S), \Delta(S)) \mid S \in \mathcal{S}\}$, and let us note that by the above definitions, for any point $(X, Y) \in K$ there exists a policy $P = P(X, Y) \in \mathrm{MIX}(\mathcal{S})$ such that $X = C(P)$ and $Y = \Delta(P)$. The points $(C(S_j), \Delta(S_j))$, for $j = 1, \ldots, q$, are in general position, implying that there must not exist a set of three or more of these points which are co-linear. Therefore, $\lambda_j > 0$ for $j = 1, \ldots, q$, where $q \geq 3$, implies that the point $(C(Q), \Delta(Q)) \in K$ must be in the interior of K. It follows that the point $(C(Q) - \varepsilon, \Delta(Q) + \varepsilon)$ is also in K, for a suitably small $\varepsilon > 0$. Thus, the policy $P = P(C(Q) - \varepsilon, \Delta(Q) + \varepsilon) \in \mathrm{MIX}(\mathcal{S})$ strictly dominates Q. □

Definition 5 Given a set $\mathcal{P}$ of policies, let us denote by $\mathrm{U}(\mathcal{P}) \subseteq \mathcal{P}$ a subset of its undominated policies. Let us further denote by $\mathrm{U}^*(\mathcal{P}) \subseteq \mathrm{U}(\mathcal{P})$ the unique minimal subset of undominated policies for which we have

$$\mathrm{MIX}(\mathrm{U}^*(\mathcal{P})) = \mathrm{MIX}(\mathrm{U}(\mathcal{P})).$$

It is easy to see that $P \in \mathrm{U}^*(\mathcal{P})$ if and only if $P \in \mathcal{P}$ and it is nondominated (not necessarily strictly) by any of the policies in $\mathrm{MIX}(\mathcal{P} \setminus \{P\})$.

Since any random mixture of policies, P_3, satisfies (13), for some pair of policies P_1 and P_2, and since no strictly dominated policy can be an optimal choice for any budget value, it follows that for any finite set of policies $\mathcal{P}$ the function

$$d_{\mathcal{P}}(B) = \max_{P \in \mathrm{MIX}(\mathcal{P})} \{\Delta(P) \mid C(P) \leq B\} \tag{14}$$

is a piecewise linear concave function with break points corresponding to elements of $\mathrm{U}^*(\mathcal{P})$; assuming meaningful policies in $\mathcal{P}$, it includes only policies P whose cost $C(P) \leq C(I) = 1$, and the curve is defined for all $B \in [a, b] \subseteq [0, 1]$, where

$$a = \min\{C(P) \mid P \in \mathcal{P}\}, \quad \text{and} \quad b = \max\{C(P) \mid P \in \mathcal{P}\}.$$

We may also observe that this curve is the two dimensional cost-detection projection of the set $\mathrm{U}(\mathrm{MIX}(\mathrm{U}^*(\mathcal{P})))$, or in other words

$$\{(B, d(B)) \mid a \leq B \leq b\} = \{(C(P), \Delta(P)) \mid P \in \mathrm{U}(\mathrm{MIX}(\mathrm{U}^*(\mathcal{P})))\}.$$

When $\mathcal{P}$ is clear from the context we shall simply denote the curve by $d(\cdot)$. We say that the set $\mathrm{U}^*(\mathcal{P})$ and the curve $d(\cdot)$ form the *extremal frontier* of the set $\mathcal{P}$. Note that when the two terminal actions, R and I, belong to $\mathcal{P}$, the extremal frontier is necessarily a nondecreasing concave curve connecting, respectively, $(0,0)$ to $(1,1)$. Let us also note next that if $\mathcal{P}_i$, for $i=1,2$, are two finite sets of policies, then we have

$$\mathrm{U}^*(\mathcal{P}_1 \cup \mathcal{P}_2) = \mathrm{U}^*(\mathrm{U}^*(\mathcal{P}_1) \cup \mathrm{U}^*(\mathcal{P}_2)), \tag{15}$$

an easy fact, the verification of which is left for the reader.

Finally, let us conclude this section by observing, in the following proposition, that $\mathrm{U}^*(\mathcal{P})$ can be efficiently computed for any finite set $\mathcal{P}$.

Proposition 2 *Assume that $\mathcal{P}$ is a finite set of policies given in sorted order of cost. Given $\mathcal{P}$ as input, the subset $\mathrm{U}^*(\mathcal{P}) \subseteq \mathcal{P}$ can then be determined in $O(|\mathcal{P}|)$ time.*

The proof of Proposition 2 is given in Appendix B.

3 The Inspection problem

The decision maker's problem is to find a policy that uses the available tests and terminal actions, which provides a high level of safety (i.e., has high detection rate), and has the smallest possible cost. Maximizing the detection rate and minimizing cost are competing objectives that cannot be simultaneously optimized. One approach is to minimize a single unconstrained objective function: $C(P) + \pi K(1 - \Delta(P))$, where K is the expected cost of missing a bad case, and as before, π is the a priori probability of bad cases. Such a cost K may include damage to infrastructure, the economy, and must quantify lives lost in a catastrophic event. However, both parameters π and K may be difficult to estimate in practice. Let us observe that not only we are unable to reliably estimate the product πK in the objective, but also this quantity cannot be affected by our decision making. What we *can* influence with our choice of a policy is the factor $(1 - \Delta(P))$, and the higher the detection rate we achieve with policy P, the smaller the expected future damage. In contrast, the other term, $C(P)$, represents a real operating cost together with the cost of false positives. Typically, this cost can be reliably estimated in the long run, and it can be influenced by the "right" choice of policy P.

Before we proceed, we make an assumption that π is negligible in comparison with 1.[3] This assumption further simplifies the analysis so that the cost of a policy P given in (9) can be rewritten as

$$C(P) = \sum_{i \in \mathcal{L}(D)} g_i(D)[c(T_i(D)) + x(i, I)]. \tag{16}$$

We note that this assumption is not necessary for our analysis, but the algebra of the proofs becomes much simpler.

Henceforth, we consider the natural formulation of the problem of finding the policies with maximum detection as given by (14), where B is a given budget limit for the total unit operating cost of the policy (i.e., including also the cost of false positives), and $\mathcal{P}$ is the set of all policies that can be constructed from a given set of elementary tests and terminal actions. To enhance the practical impact of the analysis, we would like to parametrically solve (14) for a range of different values of the budget B. This will permit a decision maker

[3] An alternative assumption, which may lead to a similar simplification, is that the cost of false positives is very large compared with the operating cost of the different tests.

to evaluate the marginal gain in safety from a particular budget increase, or determine the lowest budget value needed to achieve a specified level of safety. The solution technique which we introduce in the following will allow us to determine the function $d(\cdot)$ for the entire range of budget values [0, 1].

We are now ready to formulate our main problem and to state our main results.

The inspection problem

Input: Consider a set $\mathcal{T}$ of N elementary tests and the set of terminal actions $A = \{R, I\}$.

Problem: Denote by $\mathcal{P}$ the family of all feasible policies using the given tests and terminal actions. Determine the function $d_{\mathcal{P}}(B)$, for $B \in [0, 1]$, as defined by the optimization problem (14).

Our results show that the function $d(B)$ is a piecewise linear function defined by the finite set of deterministic policies $\mathrm{U}^*(\mathcal{P})$. Furthermore, we provide an algorithm to determine these policies, and develop an upper bound on the size of $\mathrm{U}^*(\mathcal{P})$.

4 Monotonicity of optimal policies

We are about to solve the INSPECTION PROBLEM by iteratively constructing more complex devices and policies from simpler ones, in an optimal way. In the initial step we have the elementary actions I and R. In the general step, we have already generated some policies that we include in the set of available actions. Let us recall the definition of a policy $P = (D, x)$ which involves a device D and an assignment of actions to its labels, given by x. Using the extended set of actions (including actions which, in turn, correspond to policies themselves), we would like to assign some of these actions to the labels $\mathcal{L}(D)$ in order to construct an optimal policy. To guide our reasoning, we first observe certain properties of optimal policies.

Theorem 1 (Monotonicity) *Assume that $x^* : \mathcal{L}(D) \times A \to [0, 1]$ satisfies (11) (no-repeats) and yields an optimal policy $P = (D, x^*)$ that has the largest $\Delta(P)$ among all policies of the form $P = (D, x)$ for which $C(P) \le B$. Then, for any pair of labels i and j for which $b_i/g_i > b_j/g_j$ and pairs of policies $p, q \in A$ for which $(T(p) \cup T(q)) \cap (T_i \cup T_j) = \emptyset$ and $\Delta(p) < \Delta(q)$, we must have $x^*(i, p) = 0$ or $x^*(j, q) = 0$.*

Theorem 1 implies that if two different actions (or policies) can be assigned to either of two labels, then an optimal assignment will not assign the weaker detection rate action to the label with a higher discriminatory power (where the discriminatory power of label i is indicated by its b_i/g_i ratio), if the stronger action is assigned to the label with the lower discriminatory power.

Proof Note that the feasibility condition implies that the fraction of the bad items detected by assigning policy $r \in A$ to label $k \in \mathcal{L}(D)$ is the product $b_k \Delta(r)$, and thus

$$\Delta(P) = \sum_{r \in \mathcal{P}} \sum_{k \in \mathcal{L}} x^*(k, r) b_k \Delta(r).$$

Analogously, the cost of policy P can be written as

$$C(P) = c(D) + \sum_{r \in \mathcal{P}} \sum_{k \in \mathcal{L}} x^*(k, r) g_k C(r).$$

Now, assume to the contrary that both $x^*(i,p) > 0$ and $x^*(j,q) > 0$, and let us also choose two suitably small positive parameters ε and ε' which satisfy

$$\varepsilon g_i = \varepsilon' g_j. \tag{17}$$

Let us next define a new policy P' by setting

$$x'(i,p) = x^*(i,p) - \varepsilon, \qquad x'(i,q) = x^*(i,q) + \varepsilon,$$
$$x'(j,p) = x^*(j,p) + \varepsilon' \quad \text{and} \quad x'(j,q) = x^*(j,q) - \varepsilon',$$

and letting $x'(k,r) = x^*(k,r)$ for all other combinations of $k \in \mathcal{L}$ and $r \in A$. By our assumption and by the choice of ε and ε', these values are nonnegative, and hence they define a new policy $P' = (D, x')$. We have

$$\begin{aligned} C(P') - C(P) &= -\varepsilon g_i C(p) + \varepsilon g_i C(q) - \varepsilon' g_j C(q) + \varepsilon' g_j C(p) \\ &= (C(p) - C(q))(\varepsilon' g_j - \varepsilon g_i) \\ &= 0 \end{aligned}$$

by (17). Furthermore, we have

$$\begin{aligned} \Delta(P') - \Delta(P) &= -\varepsilon b_i \Delta(p) + \varepsilon b_i \Delta(q) - \varepsilon' b_j \Delta(q) + \varepsilon' b_j \Delta(p) \\ &= (\varepsilon g_i)\frac{b_i}{g_i}(\Delta(q) - \Delta(p)) - (\varepsilon' g_j)\frac{b_j}{g_j}(\Delta(q) - \Delta(p)) \\ &= (\varepsilon g_i)\left(\frac{b_i}{g_i} - \frac{b_j}{g_j}\right)(\Delta(q) - \Delta(p)) \\ &> 0. \end{aligned}$$

Therefore, P' strictly dominates P, and hence P could not have been optimal. This contradicts the choice of x^*, and thus establishes our proposition. □

Corollary 1 *Given a device $D = \{(b_i, g_i, T_i) \mid i \in \mathcal{L}(D)\}$ and the terminal set of actions $A = \{I, R\}$, the policy (D, x) which has maximum detection, and whose cost does not exceed the given budget B, has an action assignment x that can be determined greedily by assigning I to labels $i \in \mathcal{L}(D)$ in a decreasing order of b_i/g_i, until the budget is exhausted, and then assigning R to all of the remaining labels in $\mathcal{L}(D)$.*

Let us note that this optimal action assignment x has at most one label i' (the one with the smallest value of b_i/g_i among those labels to which we assign action I with a positive $x(i,I)$ value) where both $x(i',I)$ and $x(i',R)$ may be nonzero. For all other labels $i \in \mathcal{L}$, $x(i,\cdot)$ is either zero or one. Let us introduce x^γ for $\gamma \in [0,1]$ by defining

$$x^\gamma(i',I) = \gamma, \qquad x^\gamma(i',R) = 1 - \gamma, \quad \text{and}$$
$$x^\gamma(j,I) = x(j,I), \qquad x^\gamma(j,R) = x(j,R) \quad \text{for all other labels } j \neq i'.$$

Then the policies $P_\gamma = (D, x^\gamma)$ with $\gamma \in \{0,1\}$ are deterministic, and we have

$$(D, x^\gamma) = \gamma P_1 + (1-\gamma) P_0,$$

for all optimal action assignments x^γ and the corresponding policies (D, x^γ). This proves the following corollary.

Corollary 2 *Given a device D and a set of actions A, let us denote by $\mathcal{P}(D, A)$ the set of possible policies, i.e.,*

$$\mathcal{P}(D, A) = \{(D, x) \mid x \textit{ satisfies } (8) \textit{ and } (11)\}.$$

Then for every device D and for $A = \{I, R\}$ we have that $\mathrm{U}^*(\mathcal{P}(D, A))$ is finite and consists only of deterministic policies.

5 Prefixing a test: optimal assignment of actions to test labels

A main building block in our solution technique is the solution of the *Test Prefix* problem; how to assign the available actions (or policies) to the labels of a test in a way that would maximize the detection subject to a budget constraint. We formulate this problem as an LP and suggest a greedy algorithm as a solution technique. By solving this problem parametrically for all conceivable budget values, one obtains an extremal frontier of all policies that use the given test at the root and apply a subset of the available actions.

In general, given any device t and a subset of *available actions* $A \supseteq \{I, R\}$, we can create a new policy P by *prefixing* t to a subset of the available actions A. To simplify the presentation we will assume that A is available for assignment to every label $i \in \mathcal{L}(t)$. This assumption applies to the particular case which we apply in Sect. 6; prefixing a device consisting of a single test to a set of available and undominated policies. For a specific budget value B, the problem of finding the policy with highest detection rate, using device t and actions A, and such that its total cost is at most B, is given by:

$$d_{\mathcal{P}(t,A)}(B) = \max \sum_{\substack{i\in\mathcal{L}(t),\\ \alpha\in A}} b_i \Delta(\alpha) x(i, \alpha) \tag{18a}$$

$$\text{s.t.:} \quad \sum_{\substack{i\in\mathcal{L}(t),\\ \alpha\in A}} g_i C(\alpha) x(i, \alpha) \le B - c(t), \tag{18b}$$

$$\begin{aligned} &\sum_{\alpha\in A} x(i, \alpha) = 1 \quad && \text{for each } i \in \mathcal{L}(t), \\ &x(i, \alpha) \ge 0 && \text{for all } i \in \mathcal{L}(t) \text{ and } \alpha \in A. \end{aligned} \tag{18c}$$

We now consider the deterministic policies P which are optimal for some value of B and can be constructed from t by choosing a binary assignment $x : \mathcal{L}(t) \times A \to \{0, 1\}$ satisfying both (8) and (11). All other feasible undominated policies corresponding to a non-binary assignment can be obtained by mixing from the ones corresponding to binary assignments following Corollary 2. In Algorithm 1 we generate the binary assignments $x(i, p) \in \{0, 1\}$, for all $i \in \mathcal{L}(t)$ and $p \in A$, in order to generate the corresponding set of deterministic policies $\mathcal{Q}^*$. We will show that these policies uniquely define the set of undominated policies which can be constructed from the device t and actions A, i.e., $\mathcal{Q}^* = \mathrm{U}^*(\mathcal{P}(t, A))$. In order to prove $\mathcal{Q}^* = \mathrm{U}^*(\mathcal{P}(t, A))$ we show that $\mathcal{Q}^*$ defines the piecewise-linear concave function $d(B)$, for $B \in [c(t), 1 + c(t)]$.

Algorithm 1 TestPrefix(t, A)

```
Input: A device $t = \{(b_i, g_i, T_i) \mid i \in \mathcal{L}(t)\}$, and a set of available actions $A = \{Q_0, Q_1, \dots, Q_M\}$.
Initializations:

(I1) $\mathcal{Q}^* \leftarrow \emptyset, k \leftarrow 0$
(I2) $C_j \leftarrow C(Q_j)$ and $\Delta_j \leftarrow \Delta(Q_j)$ for all $i \in \mathcal{L}(t)$ and $j = 0, \dots, M$
(I3) $\lambda_{ij} \leftarrow \frac{b_i(\Delta_j - \Delta_{j-1})}{g_i(C_j - C_{j-1})}$ for all $i \in \mathcal{L}(t)$ and $j = 1, \dots, M$
(I4) $j(i) \leftarrow 0$ for all $i \in \mathcal{L}(t)$
(I5) $x(i, Q_0) \leftarrow 1, x(i, Q_j) \leftarrow 0$ for all $i \in \mathcal{L}(t)$ and $j = 1, \dots, M$, and set $P_0 \leftarrow (t, x)$

Assumptions:

(A1) $T_i \cap T(Q_j) = \emptyset$ for all $i \in \mathcal{L}(t)$ and $j = 0, \dots, M$
(A2) $0 = C_0 < C_1 < \cdots < C_M = 1$, $\Delta_0 = 0$, and $\Delta_M = 1$ for all $i \in \mathcal{L}(t)$
(A3) $\mathrm{U}^*(A) = A$, that is the points $\{(C_j, \Delta_j) \mid j = 0, \dots, M\}$ form a concave curve,
     implying $\lambda_{i1} \geq \lambda_{i2} \geq \cdots \geq \lambda_{iM}$ for all $i \in \mathcal{L}(t)$

Main Loop:
while $\exists i \in \mathcal{L}(t)$ such that $j(i) < M$ do
  $\mathcal{Q}^* \leftarrow \mathcal{Q}^* \cup \{P_k\}$
  $k \leftarrow k + 1$
  $(i_k, j_k) \leftarrow \arg\max_{i,j}\{\lambda_{ij} \mid i \in \mathcal{L}(t) \text{ and } j(i) < j \leq M\}$
  $x(i_k, Q_{j_k - 1}) \leftarrow 0, x(i_k, Q_{j_k}) \leftarrow 1$, and set $j(i_k) = j_k$
  $P_k \leftarrow (t, x)$
  {Comment: note that at each iteration we must have
  $C(P_k) = C(P_{k-1}) + g_{i_k}(C_{j_k} - C_{j_k - 1})$
  $\Delta(P_k) = \Delta(P_{k-1}) + b_{i_k}(\Delta_{j_k} - \Delta_{j_k - 1})$}
end while
Output: The set of policies $\mathcal{Q}^*$.
```

To see the correctness of Algorithm (1), that is to see that $\mathcal{Q}^* = \mathrm{U}^*(\mathcal{P}(t, A))$, we provide both an intuitive and a formal argument. Our assumptions (A1), (A2), and (A3) make sure that it is feasible to assign any action $p \in A$ to any label $i \in \mathcal{L}(t)$, that the action set A contains the two terminal actions R and I, the action set A is ordered in an increasing order of costs, and finally A does not contain dominated actions (note that otherwise by, Proposition 2, the undominated set $\mathrm{U}^*(A)$ can be easily found in time $O(|A|)$).

Following these assumptions, we have

$$\sum_{i \in \mathcal{L}(t)} \sum_{j=1}^{M} (g_i(C_j - C_{j-1}), b_i(\Delta_j - \Delta_{j-1})) = (1, 1) \tag{19}$$

implying that the set $\mathcal{Q}^*$ corresponds to a sequence of points from $(c(t), 0)$ to $(1 + c(t), 1)$. In the algorithm we sum up the vectors

$$\big(g_{i_k}(C_{j_k} - C_{j_k - 1}), b_{i_k}(\Delta_{j_k} - \Delta_{j_k - 1})\big)$$

in a decreasing order of slopes. In every iteration of the main loop we choose a label $i \in \mathcal{L}(t)$, for which we replace the assigned action $Q_{j(i)}$ by $Q_{j(i)+1}$. Thus, at the end of the kth iteration we have $j(i_k) = j_k$, and action Q_{j_k} is assigned to label i_k in policy P_k. This further implies that

$$(C(P_k), \Delta(P_k)) = \big(C(P_{k-1}) + g_{i_k}(C_{j_k} - C_{j_k-1}), \Delta(P_{k-1}) + b_{i_k}(\Delta_{j_k} - \Delta_{j_k-1})\big)$$

holds for all iterations k. Furthermore, in the kth iteration we move along the vector

$$\big(g_{i_k}(C_{j_k} - C_{j_k-1}), b_{i_k}(\Delta_{j_k} - \Delta_{j_k-1})\big);$$

this can be interpreted as considering a mixture of policies P_{k-1} and P_k, such that for every small increase in the total cost of $\epsilon > 0$, we get an incremental $\lambda_{i_k j_k}\epsilon$ detection. Since at each iteration we choose the maximum possible $\lambda_{i_k j_k}$, intuitively it is clear that we gain the highest level of detection for a given budget. We can also provide a formal proof for the correctness of Algorithm 1 by showing that it yields an optimal solution to the linear programming problem (18). First, let us note that given a budget value of $B = c(t)$, for all $i \in \mathcal{L}(t)$, we have $P_0 = (t, x)$ with $x(i, Q_0) = 1$, for all $i \in \mathcal{L}(t)$, which is the only policy that satisfies the budget constraint. Proposition 3 will demonstrate the correctness of the algorithm for all other budget values $B \in (c(t), 1 + c(t)]$. Let us note that following from (19), for any budget value $B \in (c(t), 1 + c(t)]$ we have $B = (1 - \gamma)C(P_{k-1}) + \gamma C(P_k)$, for some iteration $k \geq 1$ and $\gamma \in (0, 1]$.

Proposition 3 *Assume that our budget is $B = (1 - \gamma)C(P_{k-1}) + \gamma C(P_k)$ for some $\gamma \in (0, 1]$. Let us run Algorithm 1, stop in the kth iteration and consider the assignment defined by*

$$x^*(i, Q_j) = \begin{cases} 1 & j = j(i), \\ 1 - \gamma & j = j_{k-1}, \\ \gamma & j = j_k, \\ 0 & \text{otherwise}. \end{cases}$$

Then x^* is an optimal solution of the LP (18).

Proof It is immediate to verify that x^* is a feasible solution in the LP (18), and that it defines the policy P^*. Furthermore, by construction, we have equality in the budget constraint (18b). According to the theory of linear programming, to prove that x^* optimal, it is enough to show a dual feasible solution, which has the same objective value. To this end, let us first consider the dual LP:

$$W^* = \min(B - c(t))y + \sum_{i \in \mathcal{L}(t)} z_i$$

$$\text{s.t.:} \quad g_i C(p)y + z_i \geq b_i \Delta(p) \quad \text{for all } i \in \mathcal{L}(t) \text{ and } p \in A,$$

$$y \geq 0,$$

Let us now fix $y^* = \lambda_{i_k j_k}$, and let

$$z_i^* = \max_{j=0,\dots,M} \left\{ b_i \Delta(Q_j) - g_i C(Q_j) y^* \right\}. \tag{20}$$

It is immediate to see that (y^*, z^*) is a feasible solution to the dual LP. Furthermore, letting $\lambda_{i0} = +\infty$, the maximum in (20) is attained by

$$j(i) = \max\{j = 0, \dots, M \mid \lambda_{ij} \geq y^*\}.$$

Now since

$$\lambda_{i_1 j_1} \geq \cdots \geq \lambda_{i_k j_k} \geq \max_{i,j}\{\lambda_{ij} \mid i \in \mathcal{L}(t) \text{ and } j(i) < j \leq M\},$$

we must have

$$z_i^* = \begin{cases} b_i \Delta_{j(i)} - g_i C_{j(i)} y^* & \text{if } i \neq i_k, \\ b_{i_k} \Delta_{j_{k-1}} - g_{i_k} C_{j_{k-1}} y^* = b_{i_k} \Delta_{j_k} - g_{i_k} C_{j_k} y^* & \text{if } i = i_k. \end{cases} \tag{21}$$

Thus we can rewrite $W^* = (B - c(t))y^* + \sum_{i \in \mathcal{L}(t)} z_i^*$ as

$$\begin{aligned} W^* &= (B - c(t))y^* + \sum_{i \in \mathcal{L}(t) \setminus \{i_k\}} z_i^* + (1-\gamma) z_{i_k}^* + \gamma z_{i_k}^* \\ &= (B - c(t))y^* + \sum_{i \in \mathcal{L}(t) \setminus \{i_k\}} \left(b_i \Delta_{j(i)} - g_i C_{j(i)} y^*\right) \\ &\quad + (1-\gamma)\left(b_{i_k} \Delta_{j_{k-1}} - g_{i_k} C_{j_{k-1}} y^*\right) + \gamma \left(b_{i_k} \Delta_{j_k} - g_{i_k} C_{j_k} y^*\right) \end{aligned}$$

Let us note that Algorithm 1 ensures that for all $(i, p) \in \mathcal{L}(t) \times A$, $x^*(i, p) > 0$ if an only if

$$(i, p) \in \{(i, j(i)) \mid i \in \mathcal{L}(t) \setminus \{i_k\}\} \cup \{i_k, j_{k-1}\} \cup \{i_k, j_k\}.$$

Therefore,

$$\begin{aligned} W^* &= (B - c(t))y^* + \sum_{i \in \mathcal{L}(t) \setminus \{i_k\}} \left(b_i \Delta_{j(i)} - g_i C_{j(i)} y^*\right) x^*(i, Q_{j(i)}) \\ &\quad + \left(b_{i_k} \Delta_{j_{k-1}} - g_{i_k} C_{j_{k-1}} y^*\right) x^*(i_k, Q_{j_{k-1}}) + \left(b_{i_k} \Delta_{j_k} - g_{i_k} C_{j_k} y^*\right) x^*(i_k, Q_{j_k}) \\ &= \Bigg(B - c(t) - \sum_{i \in \mathcal{L}(t) \setminus \{i_k\}} g_i C_{j(i)} x^*(i, Q_{j(i)}) - g_{i_k} C_{j_{k-1}} x^*(i_k, Q_{j_{k-1}}) \\ &\quad - g_{i_k} C_{j_k} x^*(i_k, Q_{j_k})\Bigg) y^* \\ &\quad + \Bigg(\sum_{i \in \mathcal{L}(t) \setminus \{i_k\}} b_i \Delta_{j(i)} x^*(i, Q_{j(i)}) + b_{i_k} \Delta_{j_{k-1}} x^*(i_k, Q_{j_{k-1}}) + b_{i_k} \Delta_{j_k} x^*(i_k, Q_{j_k})\Bigg). \end{aligned}$$

Since x^* is a solution satisfying the budget constraint (18b) with equality, we have

$$\begin{aligned} &\Bigg(B - c(t) - \sum_{i \in \mathcal{L}(t) \setminus \{i_k\}} g_i C_{j(i)} x^*(i, Q_{j(i)}) - g_{i_k} C_{j_{k-1}} x^*(i_k, Q_{j_{k-1}}) - g_{i_k} C_{j_k} x^*(i_k, Q_{j_k})\Bigg) \\ &= B - c(t) - \sum_{\substack{i \in \mathcal{L}(t), \\ p \in A}} g_i C(i, p) x^*(i, p) = 0. \end{aligned}$$

Thus,

$$W^* = \left(\sum_{i \in \mathcal{L}(t) \setminus \{i_k\}} b_i \Delta_{j(i)} x^*(i, Q_{j(i)}) + b_{i_k} \Delta_{j_{k-1}} x^*(i_k, Q_{j_{k-1}}) + b_{i_k} \Delta_{j_k} x^*(i_k, Q_{j_k}) \right)$$
$$= \sum_{\substack{i \in \mathcal{L}(t), \\ p \in A}} b_i \Delta(i, p) x^*(i, p),$$

and we have shown that the dual solution (y^*, z^*) has the same objective function value as the primal solution x^*, implying that x^* is optimal by LP duality. □

Next we analyze the runtime complexity of TestPrefix(t, A).

Proposition 4 *The runtime complexity of Algorithm 1 is* $O(|\mathcal{L}(t)||A| \log |\mathcal{L}(t)|)$.

Proof By our assumption (A3) we have all the λ_{ij} values, for $j = 1, \dots, M$, sorted for each $i \in \mathcal{L}(t)$. We need to merge these ordered sets to be able to run the main loop. Merging $|\mathcal{L}(t)|$ ordered sets can be done in $O(|\mathcal{L}(t)||A| \log |\mathcal{L}(t)|)$ time, which has to be done only once, prior to running the main loop. All other initializations can be done in $O(|\mathcal{L}(t)||A|)$ time. The proposition follows since an iteration of the main loop requires $O(1)$ time and is executed exactly $|\mathcal{L}(t)||A|$ times. □

The LP formulation (18) turns out to be a special case of the Linear Multiple Choice Knapsack problem (LMCK); see for example (Sinha and Zoltners 1979). In that problem, we are given a collection of mutually disjoint sets of items, called multiple-choice sets. A convex combination of items must be selected from each set in order to maximize the profit subject to a budgetary constraint. We can view the set of potential action assignments to a particular label as an LMCK multiple-choice set, and each pair of a label and action, as a knapsack item with the corresponding incremental detection and cost values. A naive modeling of our problem as an LMCK instance would involve a number of knapsack items that is equal to the number of available actions times the number of test labels. While our algorithm might resemble a host of greedy algorithms developed for LMCK, the algorithm is different than those algorithms in explicitly enumerating the entire extremal frontier of policies, and in utilizing a computational improvement which applies to our special case with multiple-choice sets that originate from the same single set of available actions.

Sinha and Zoltners (1979) propose a greedy algorithm and characterize the solution space of LMCK; although their algorithm was designed to solve LMCK for a given budget value, it could be used to parametrically solve the problem for a range of costs up to the given budget value in a similar fashion as Algorithm 1. Other algorithms that can be used to generate an entire frontier of optimal policies have been described in the literature in the context of relaxing the integer Multiple Choice Knapsack problem. Ibaraki et al. (1978) suggest a dual based algorithm for a particular special case of LMCK with equality constraints replaced by inequalities.[4] The complexity of their algorithm is $O(k \log k)$, where k is the total number of items, and it can also be extended to find an entire frontier of optimal policies. Glover and Klingman (1979) also propose an $O(k \log k)$ algorithm that specializes the dual simplex

[4]Note that having an element with zero cost and zero profit in each multiple-choice set corresponds to a slack variable which allows one to solve the inequality version as a standard LMCK problem.

method for the LMCK problem, and also results in a greedy algorithm. Zemel (1980) generalizes LMCK to allow also knapsack items that are not contained in any multiple-choice set; his algorithm involves a transformation of the LMCK instance into the Continuous sKnapsack Problem (CKP). The complexity of Zemel's algorithm is $O(k \log k_{\max})$ where $k_{\max}$ is the maximum number of items in a multiple-choice set, using an $O(k)$ time algorithm to solve the CKP problem. By using instead Dantzig's $O(k \log k)$ greedy algorithm (Dantzig 1957) one could also use Zemel's transformation to generate the entire frontier of optimal policies. Finally, Pisinger (1995) also describes a greedy algorithm based on Sinha and Zoltners (1979), which could enumerate the entire frontier of optimal policies, with a runtime complexity of $O(k \log k)$. The linear time LMCK algorithms of Dyer (1984) and Zemel (1984), although more efficient for a particular budget value, do not suit our purpose of parametrically solving the problem for all conceivable budget values. The reader may refer to Kellerer et al. (2004) for more details about LMCK and other related knapsack variants.

Let us now consider the LMCK solution techniques for solving (18): consider each pair $(i, p) \in \mathcal{L}(t) \times A$ to be a knapsack item with value $b_i \Delta_p$, and cost $g_i C(p)$, and also consider the set of possible assignments for a particular $i \in \mathcal{L}(t)$ as a multiple-choice set. Then by naively applying any LMCK greedy algorithm to solve (18) with the resulting $k = |\mathcal{L}(t)||A|$ knapsack items one obtains a worst case complexity of $O(|\mathcal{L}(t)||A| \log(|\mathcal{L}(t)||A|))$. The runtime complexity of Algorithm 1, on the other hand, is $O(|\mathcal{L}(t)||A| \log |\mathcal{L}(t)|)$, which is a substantial improvement because one generally finds that $|\mathcal{L}(t)| \ll |A|$.

Note that if set A is not given in sorted order we only need to sort the policies $p \in A$ in increasing order of the costs $C(p)$. The costs of the elements of the ith LMCK multiple-choice set are simply given by the products of g_i and $C(p)$, for each $p \in A$, (i.e., the costs of elements corresponding to the same $p \in A$ differ only by scalar multiplication). Even if we considered an unordered set A as input, then taking into account the runtime complexity of sorting the set A, the total runtime complexity of TestPrefix(t, A) with sorting would be $O(|\mathcal{L}(t)||A| \log |\mathcal{L}(t)| + |A| \log |A|)$, which remains preferable to $O(|\mathcal{L}(t)||A| \log(|\mathcal{L}(t)||A|))$. Finally, if seeking the best policies which can be constructed from the device t and set of policies A, then we only need to consider the set $\mathrm{U}^*(\mathcal{Q}^* \cup A)$. While we may have policies $P \in \mathcal{Q}^*$ with $C(P) > 1$, these will be dominated by $I \in \mathrm{U}^*(\mathcal{Q}^* \cup A)$. As a consequence we can speed up TestPrefix(t, A) somewhat by replacing the WHILE condition in the Main Loop by "WHILE $C(P_k) \leq 1$". This change, however, will not affect our worst case time complexity.

6 Putting the pieces together: solving the inspection problem via dynamic programming

The *TestPrefix* algorithm of the previous section essentially merges $|\mathcal{L}(t)|$ efficient frontiers, one for each $i \in \mathcal{L}(t)$, into a single extremal frontier of policies that use device t as the root. We are now going to use this procedure in a recursive scheme to build the frontier of policies, based on iteratively enriching the subsets of the available actions with more complex policies. We will apply *TestPrefix*$(t, \mathcal{A})$ for the case when t is a device consisting of a single test (and thus we maintain the assumption that the set of actions A is available for all $i \in \mathcal{L}(t)$).

For a subset $T \subseteq \mathcal{T}$ of the available tests let us denote by A_T the set of undominated deterministic policies that use only tests in the set T as well as the two terminal actions I and R. By definition, we have $A_\emptyset = \{I, R\}$. Note that we have previously denoted the set of tests involved in policy P as $T(P)$. Given a set of tests T in Algorithm 2, $T(P) = T$ for all polices for which the set T is available. This is although in general for $P = (D, x) \in A_T$,

Algorithm 2 *Frontier*$(\mathcal{T})$

```
1: **Input**: A family of tests $\mathcal{T}$, and two terminal actions $I$ and $R$.
2: **Initializations**: $A_\emptyset \leftarrow \{I, R\}$
3: **Main Loop**:
4: **for** $k = 1, \ldots, N$ **do**
5:     **for** all subsets $T \subseteq \mathcal{T}$ of size $|T| = k$ **do**
6:         **for** all tests $t \in T$ **do**
7:             $B_{T,t} \leftarrow \mathit{TestPrefix}(t, A_{T\setminus\{t\}})$
8:         **end for**
9:         $A_T \leftarrow \mathrm{U}^*\left(\bigcup_{t\in T}(B_{T,t} \cup A_{T\setminus\{t\}})\right)$
10:    **end for**
11: **end for**
12: **Output**: $A_{\mathcal{T}}$.
```

and $t \in T$, it may be that $t \notin T_i$ for all $i \in \mathcal{L}(D)$ with $b_i > 0$. That is, the tests T are not in fact used by all policies in A_T.

Following from Proposition 3, and the fact that in each iteration of the main loop, for all subsets of k tests, the input is a frontier corresponding to a subset of $k - 1$ tests which we have generated earlier, *Frontier*$(\mathcal{T})$ eventually generates the frontier of all policies that may be obtained from the given set of tests $\mathcal{T}$. Let $N = |\mathcal{T}|$, $L = \max_{t\in\mathcal{T}}\{|\mathcal{L}(t)|\}$, and $M = \max_{T\subseteq\mathcal{T}}\{|A_T|\}$. Then the running time of this procedure can be bounded in the worst case, as we establish in the following proposition.

Proposition 5 *Frontier*$(\mathcal{T})$ *runs in* $O(2^N NL \log LM)$ *time.*

Proof $O(2^N N)$ *TestPrefix* problems are solved for all 2^N possible subsets of tests. Since we maintain the action sets A_T in sorted order, the runtime complexity of each of the *TestPrefix* applications is bounded by $O(ML \log L)$. By Proposition 2, the complexity of *UpperHull* is bounded by $O(M)$ so that the complexity of both *UpperHull* and taking the union in step 9 are dominated by the complexity of *TestPrefix* and the total runtime complexity is $O(2^N NL \log LM)$. □

Finally, we conclude the section with a few remarks about an extension and a practical improvement of the algorithm. First, it might be the case that a decision maker would also like to constrain the number of tests that can be conducted in any sequence. For example this may be necessary in order to control delay, or may be due to some physical layout limitations. The dynamic programming algorithm can be easily extended to solve the inspection problem subject to a sequence length constraint, by simply stopping when $k = |T|$ reaches the specified limit. Regarding improving the practical efficiency of the algorithm, clearly, in the kth Main Loop iteration we use the previously computed subsets of size $k - 1$. Thus, all policy sets A_T for $|T| < k - 1$ can be deleted in the kth iteration. This substantially reduces the memory requirement of the algorithm, though it may not change the worst case time and space complexities.

7 An upper bound on the size of the extremal frontier

There are simple bounds available for the number of binary decision trees (see e.g., Stroud and Saeger 2003), and these bounds can be easily extended for trees with larger branching factors (see (1) and the discussion in Sect. 1). We now proceed to derive a tighter upper bound on the size of the extremal frontier consisting only of the undominated deterministic policies.

Theorem 2 *Given a set of tests $\mathcal{T}$, there exist deterministic policies $P_0 = R, P_1, \dots, P_M$ such that the extremal frontier of policies, obtainable from $\mathcal{T}$ and the terminal actions s $\{I, R\}$, is a piecewise linear concave function defined by the breakpoints $(C(P_j), \Delta(P_j))$, $j = 0, 1, \dots, M$, and where*

$$M \le |\mathcal{T}|! \prod_{t\in\mathcal{T}} (1 + |\mathcal{L}(t)|) \le N!(1+L)^N.$$

Proof As we observed above, Algorithm 2 starts with deterministic policies consisting only of a single label and no tests. Then in step 7 each invocation of *TestPrefix* generates a set of deterministic policies.

To prove the upper bound on the cardinality of the set of policies, we observe first that $|A_\emptyset| = 2$. Thus, in this first stage, when invoking *TestPrefix*$(t, A_\emptyset)$, we will have at most $1 + |\mathcal{L}(t)|$ many actions in $B_{\emptyset,t}$, and thus

$$|A_S| \le \prod_{s\in S} (1 + |\mathcal{L}(s)|) \le |S|! \prod_{s\in S} (1 + |\mathcal{L}(s)|)$$

follows for all subsets S of tests having size $|S| = 1$. In a general step, the root test $t \in T$ has $|\mathcal{L}(t)|$ labels and for each label we have $|A_{T\setminus\{t\}}|$ possible actions. Thus,

$$|B_{T,t}| \le 1 + |\mathcal{L}(t)||A_{T\setminus\{t\}}| \le (1 + |\mathcal{L}(t)|)|A_{T\setminus\{t\}}|$$

follows, and consequently when these sets are merged in step 9 we get

$$|A_T| \le \sum_{t\in T} (1 + |\mathcal{L}(t)|)|A_{T\setminus\{t\}}| = |T|! \prod_{t\in T} (1 + |\mathcal{L}(t)|)$$

for all subsets $T \subseteq \mathcal{T}$. □

Let us remark that in practice the number of policies corresponding to the breakpoints of the extremal frontier is much smaller than this upper bound, and thus Algorithm 2 terminates much faster than the running time implied by the worst case upper bounds of Theorem 2 and Propostion 5.

8 Computational results

We demonstrate our computational results using randomly generated sensors and a set of continuous Gaussian sensor models suggested by Stroud and Saeger (2003) and Boros et al. (2009). We ran all computational experiments, using a MATLAB implementation of Algorithms 2 and 1, on a machine with an Intel Xeon 3.0 GHZ CPU and 6 GB of RAM. Using both the randomly generated sensors and the previously studied sensor models we show that we are able to solve larger instances of the inspection problem. We are also able to demonstrate improved detection performance when using finer discrete approximations of the continuous sensor models suggested by Stroud and Saeger (2003) and Boros et al. (2009).

8.1 Randomly generated tests

Computational results for randomly generated sensors (tests) are shown in Table 1, Table 2 and Table 3, for two, four and ten label configurations, respectively. The sensor cost is se-

Table 1 Computational results, including number of vertices of the efficient frontier and running times (in seconds), for randomly generated sensors with 2 labels. *Each row* corresponds to the data of 20 repeated experiments with the same indicated configuration

Sensors	Vertices			Runtime		
	Max	Avg	S. Dev	Max	Avg	S. Dev
2	5	3.85	0.81	0.72	0.055	0.16
3	9	6.10	0.07	0.13	0.06	0.02
4	15	7.55	3.14	0.18	0.16	0.01
5	28	11.30	5.67	0.53	0.45	0.04
6	43	16.20	9.25	1.46	1.18	0.13
7	73	25.55	15.99	4.11	3.12	0.39
8	212	48.35	44.19	15.43	9.23	1.99
9	323	53.25	68.55	39.91	20.99	6.23
15	1801	337.20	433.90	7807.71	3563.68	1522.28

Table 2 Computational results, including number of vertices of the efficient frontier and running times (in seconds), for randomly generated sensors with 4 labels. *Each row* corresponds to the data of 20 repeated experiments with the same indicated configuration

Sensors	Vertices			Runtime		
	Max	Avg	S. Dev	Max	Avg	S. Dev
2	16	11.00	2.94	0.02	0.02	0.003
3	47	27.35	10.82	0.10	0.08	0.01
4	177	91.00	47.62	0.47	0.34	0.07
5	364	164.30	98.79	1.88	1.21	0.34
6	1127	438.55	268.67	8.33	5.22	1.58
7	4848	1385.20	1330.73	67.33	26.27	16.49
8	16416	4098.60	4119.8	578.20	149.40	134.79
9	17178	9076.50	4793.70	2214.30	1139.70	621.33

lected uniformly at random from the interval $[0, 0.15]$ while the complete inspection cost $C(I) = 1$. Each b_i (similarly g_i) is generated sequentially, and chosen uniformly at random from the remaining interval $[0, 1 - \sum_{i\in\mathcal{L}} b_i]$ (respectively from $[0, 1 - \sum_{i\in\mathcal{L}} g_i]$). For each configuration of sensor number and label number, we ran 20 repeated experiments, using different instances of randomly generated sensors.

8.2 BKFSS Gaussian distribution sensors

Boros et al. (2009) demonstrate computational results for their linear programming formulation using 4 sensors that are characterized by different Gaussian distributions of good and bad cases. These sensors are described in more detail in Appendix A. The authors compute minimum cost decision tree mixtures achieving a detection rate of at least 81.5%. They provide running times up to a branching factor (i.e., the maximum number of labels per test) of 7. With 7 labels per sensor, they find that the LP solver running time exceeds 1975 seconds. They are able to run experiments with up to 8 labels and find that the minimum cost policy with 8 labels, and overall in their experiments, has a cost of \$12.06. With the same 7 labels

Table 3 Computational results, including number of vertices of the efficient frontier and running times (in seconds), for randomly generated sensors with 10 labels. *Each row* corresponds to the data of 20 repeated experiments with the same indicated configuration

Sensors	Vertices			Runtime		
	Max	Avg	S. Dev	Max	Avg	S. Dev
2	67	31.30	17.48	0.07	0.05	0.01
3	609	230.00	158.94	0.93	0.49	0.20
4	5704	1295.70	1353.35	37.35	10.22	8.44
5	9972	3322.90	2631.29	439.66	120.97	124.89
6	36910	10172.20	7897.17	8041.53	1727.36	2021.90

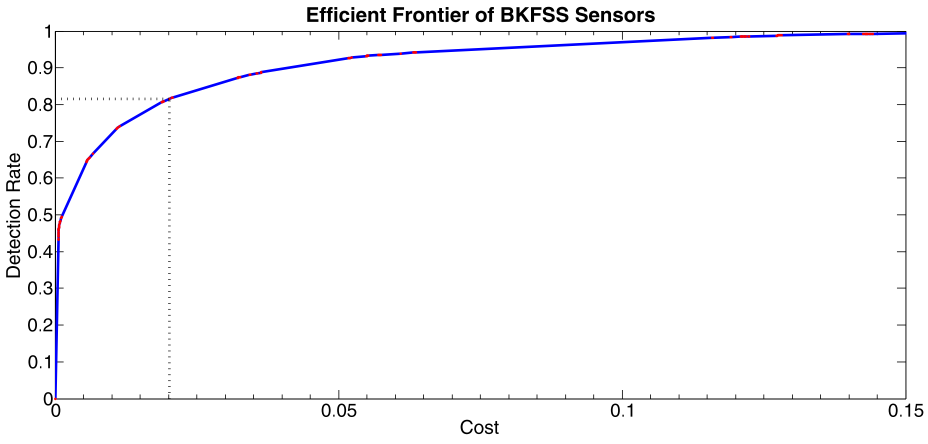


Fig. 5 The entire efficient frontier computed for the BKFSS sensors using the *Frontier* dynamic programming algorithm (Algorithm 2), and the same set of seven thresholds (eight labels) used by Boros et al. (2009). The minimum cost policy achieving detection of 81.5% corresponds to the point (0.02, 0.815) on the *curve* (since the cost of inspection $60 is normalized as the unit of measurement, the policy's cost is, in fact $12.06)

per sensor we compute the entire extremal frontier consisting of 1747 deterministic policies (i.e., vertices) in 4.25 CPU seconds. With 8 labels per sensor we are able to compute the entire efficient frontier, consisting of 2748 vertices, in 7.91 seconds. Although we are unable to replicate the experiment of Boros et al. (2009) on an the same computing platform, it is quite clear that we are able to solve larger problems, within an order of magnitude faster than the running times reported in Boros et al. (2009).[5] This is while we enumerate the entire frontier of efficient policies compared with the output of the LP formulation in Boros et al. (2009) which consists of only a single optimal policy for a fixed value of the budget. The efficient frontier with 8 labels per sensor is shown in Fig. 5.

We have also run experiments using a different discretization scheme of the continuous ROC curve. We discretize the curve by choosing the break-points so that the maximum relative error never exceeds a given error parameter ϵ. Following this scheme a different number of labels may be used for each sensor. The resulting number of labels, number of

[5]Note that the type of LP solver used to compute the formulation in Boros et al. (2009) may also affect their running times to a large extent. Such improvement can only be negligible compared with respect to the order of magnitude improvement which we were able to achieve.

Table 4 Computational results, including number of vertices of the efficient frontier and running times (seconds), for the BKFSS sensors

ϵ	Number of labels	Vertices	Runtime
1.0%	(8,14,6,3)	1567	8.10
0.9%	(8,14,6,3)	1589	8.26
0.8%	(8,15,6,3)	1683	8.97
0.7%	(9,16,6,3)	2004	11.11
0.6%	(9,17,7,3)	2341	15.53
0.5%	(10,19,7,3)	2811	22.05
0.4%	(11,21,8,4)	5635	55.09
0.3%	(13,24,9,4)	8710	118.10
0.2%	(15,29,11,4)	13905	311.66
0.1%	(21,40,15,6)	52477	3998.13

This table shows that we loose very little in precision and significantly save on computation time if the input ROC curves are approximated by piecewise linear functions with only a few pieces. Applying an analogous idea to the cost-detection curves would suggest that a controlled approximation, applied at every step of the dynamic program, might yield computational improvements while not losing much accuracy. This line of research has been initiated while this paper was in press. The reader may find updates at the project's website: http://snsrtree.rutgers.edu

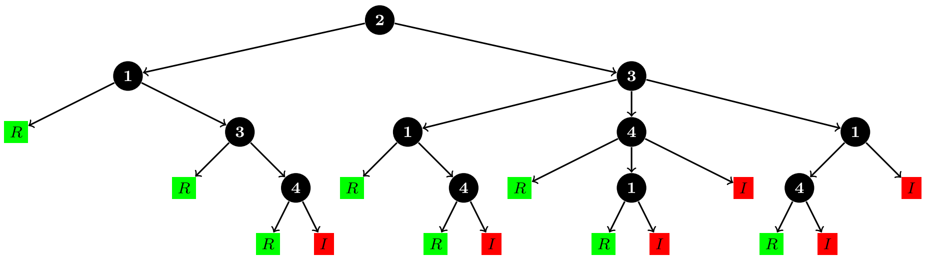


Fig. 6 A pure inspection policy, using the BKFSS sensors, found using the selection of thresholds corresponding to a maximum relative error of $\epsilon = 0.1\%$ with respect to the continuous curve. This pure inspection policy has a cost of \$11.87 and a detection rate of 81.53%. The *diagram* shows a condensed representation of the decision tree; the decision rules associated with each branch may contain conditions with respect to the previous sensors encountered along the path

vertices on the efficient frontier and running time as measured by CPU seconds are shown in Table 4. With an ϵ of 0.1% we find an undominated pure inspection policy with a detection rate that exceeds 81.5% and cost of \$11.87, which is lower than the cost of the least cost policy found in Boros et al. (2009). This inspection policy is shown in Fig. 6.

9 Conclusions and future research

We have considered inspection systems as mixtures of decision trees, with no *a priori* constraint on branching factor, or number of test labels. We have shown a monotonicity property of optimal inspection policies. We find that the monotonicity property, Theorem 1, and more specifically Corollary 1, is constructive in finding optimal inspection policies in the special case of assigning only terminal actions to a given device.

In general we find that optimally assigning actions to the labels of a device can be modeled as a Linear Multiple Choice Knapsack problem. In the special case of interest we are able to solve the corresponding variation of the knapsack problem faster than previous algorithms, which solve the more general problem. For diagnostic tests (e.g., sensors) that are stochastically independent, we propose a dynamic programming algorithm, which proves to be very fast in practice for the number of sensors that is currently being considered in container inspection applications.

The dynamic programming algorithm's worst case complexity is the product of an exponential in the number of sensors, and a polynomial in the maximum number of vertices of the extremal frontier (over all subsets of sensors). We are able to provide an upper bound for the number of vertices of the extremal frontier in terms of the input of the problem that is tighter than previous bounds given for the total number of policies (in the case of binary decision trees) (Stroud and Saeger 2003).

The algorithm's running time in practice is significantly faster than that of the linear programming approach (Boros et al. 2009). This is true even though our algorithm provides *an entire curve* of efficient (undominated) policies and not just the best inspection policy given a single specific budget. Generating an entire extremal frontier of inspection policies supports sensitivity analysis, as well as the direct computation of any linear or nonlinear utility function over the efficient set (for more details and an alternative approach of optimizing over the efficient set of a bicriteria optimization problem we refer the reader to Benson and Lee (1996)). Although the number of points on the extremal frontier can grow exponentially large in the worst case, our experiments find that the size of the frontier remains manageable in practice.

Appendix A: The BFKSS sensors

In Stroud and Saeger (2003) probability distributions of sensor readings are suggested for the good and bad type of shipping container populations, for four different hypothetical sensors. In Fig. 7 we show the key characteristics of these sensors, which are used in the calculations reported in Sect. 8.2, and by Boros et al. (2009).

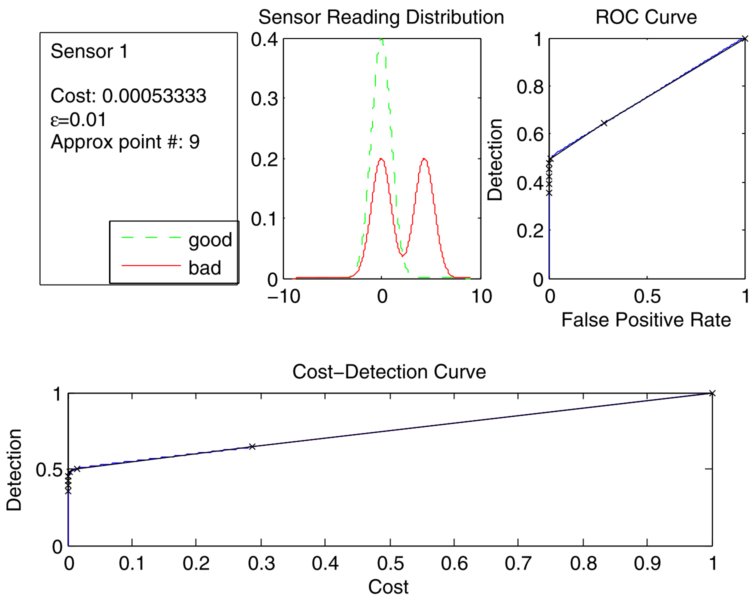
The sensor model (Stroud and Saeger 2003; Boros et al. 2009) assumes for each sensor that the good and bad containers have readings distributed normally with variance 1. Readings of good containers have a mean of zero for all sensors. For the first sensor half of the containers have a mean of 4.37 and the other half has a mean of zero (which is the same as the good containers). For the remaining three sensors the mean of bad containers is 1.53, 2.9 and 4.6. The costs of the four sensors are 0.32, 0.92, 57 and 176, respectively. The cost of manual inspection is assumed to be \$600.

Each sensor is described initially by two conditional probability distributions in the "signal space". We convert them (numerically) to ROC curves, and then, using the specified cost, to cost-detection curves. In choosing the breakpoints to replace these continuous curves by a finite number of linear components, we consider the relative error of the piecewise linear approximation of the continuous ROC curve.

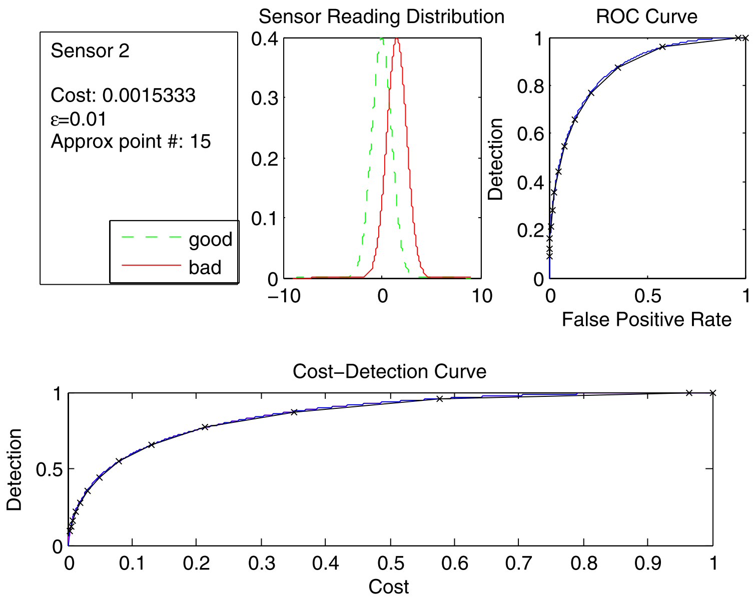
Appendix B: Proof of Proposition 2

Proof of Proposition 2 We show that the following simple procedure computes $\mathrm{U}^*(\mathcal{P})$ in the claimed running time. We first note that we execute the main loop $O(M)$ times. Let us also observe that we have

$$\lambda(1) > \lambda(2) = \frac{\Delta(P_{i_2}) - \Delta(P_{i_1})}{C(P_{i_2}) - C(P_{i_1})} > \cdots > \lambda(q) = \frac{\Delta(P_{i_q}) - \Delta(P_{i_{q-1}})}{C(P_q) - C(P_{q-1})}.$$

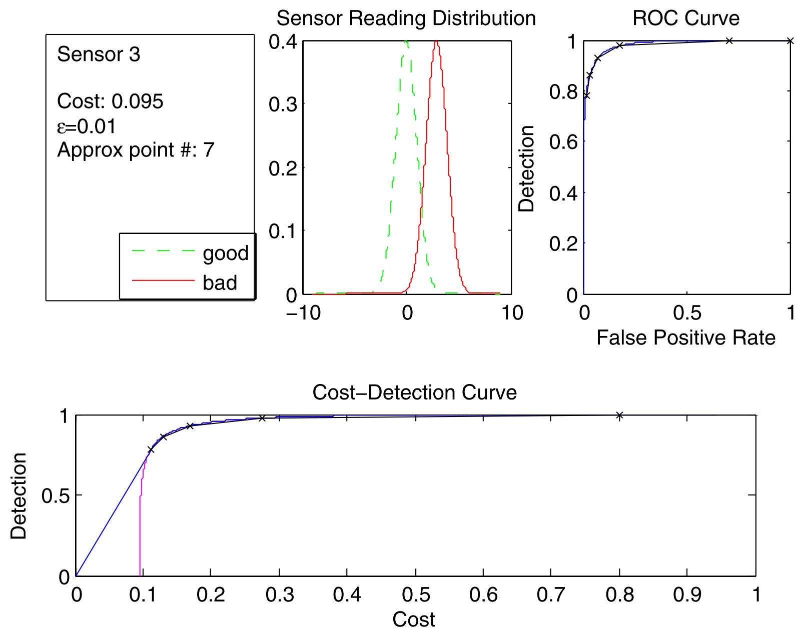


(a) Sensor 1

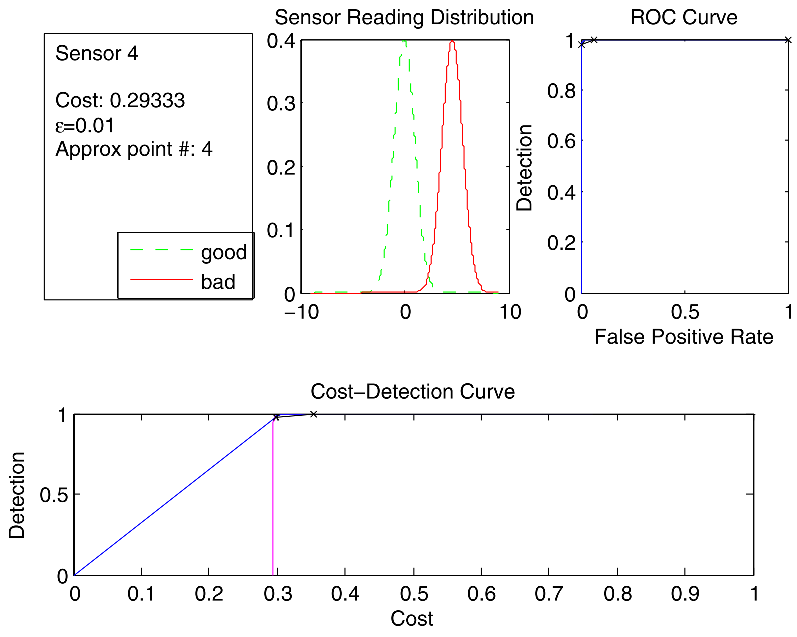


(b) Sensor 2

Fig. 7 The 4 sensors given in Boros et al. (2009) are described by the bad and good probability distributions in the signal space, ROC curve and cost-detection curve. A discretization of the cost-detection curve is given by a piecewise linear function approximating the curve, with a prespecified maximum relative error of 1%



(c) Sensor 3



(d) Sensor 4

Fig. 7 (*Continued*)

Algorithm 3 UpperHull($\mathcal{P}$)

```
 1: Input: a set of policies $\mathcal{P} = \{P_1, \dots, P_M\}$.
 2: Assumptions: an increasing order of costs, $C(P_1) < \cdots < C(P_M)$ (whenever inequality
    is not strict, it suffices to keep the policy with the highest detection rate).
 3: Initializations: Set $i_1 \leftarrow 1$, $i_2 \leftarrow 2$, $q \leftarrow 2$, $t \leftarrow 3$.
 4: Main Loop:
 5: while $t \le M$ do
 6:     while $\lambda(q+1) = \frac{\Delta(P_t)-\Delta(P_{i_q})}{C(P_t)-C(P_{i_q})} \ge \lambda(q) = \begin{cases} \frac{\Delta(P_{i_q})-\Delta(P_{i_{q-1}})}{C(P_{i_q})-C(P_{i_{q-1}})} & \text{if } q \ge 2 \\ \infty & \text{if } q = 1 \end{cases}$ do
 7:         $q \leftarrow q - 1$
 8:     end while
 9:     $q \leftarrow q + 1$
10:     $i_q \leftarrow t$
11:     $t \leftarrow t + 1$.
12: end while
13: Output: $\{P_{i_1}, P_{i_2}, \dots, P_{i_q}\}$.
```

Otherwise, if we had $\lambda(i_j) \le \lambda(i_{j+1})$, then we would delete i_j before adding i_{j+1} in step 6 of the algorithm. Thus, the points corresponding to the policies of the output set $\{P_{i_1}, P_{i_2}, \dots, P_{i_q}\}$ indeed form a concave curve. Hence none of these policies are dominated by a mixture of the others. Finally, in every step that a policy P_{i_q} is removed in step 6, it is dominated by a mixture of policies P_t and P_{q-1}. We compute a slope $\frac{\Delta(P_t)-\Delta(P_{i_q})}{C(P_t)-C(P_{i_q})}$ at most twice for each element $P_t \in \mathcal{P}$. Once when adding a policy, by assigning t to i_q, and possibly another time when removing i_q in step 6. Now, if $P_{i_q} = P_t$ is removed in step 6 we never access this policy again. Thus, when the algorithm terminates we must have $\{P_{i_1}, \dots, P_{i_q}\} = \mathrm{U}^*(\mathcal{P})$, and the total running time is $O(M)$. □

Appendix C: A Practical speed-up of the dynamic programming algorithm

As mentioned in Sect. 6 it is possible to gain a practical speed up by not taking the union with the previous frontier at each iteration in each stage of the dynamic programming Algorithm 2. We prove the sufficiency of taking the union with $\{(0,0),(1,1)\}$ instead of taking the union with the entire frontier $A_{T\setminus\{t\}}$ in each iteration of the algorithm.

Proposition 6 *In step 9 of Algorithm 2, for all $T \subseteq \mathcal{T}$,*

$$A_T = \mathrm{U}^*\left(\bigcup_{t\in T} B_{T,t} \cup A_{T\setminus\{t\}}\right) = \mathrm{U}^*\left(\bigcup_{t\in T} B_{T,t} \cup \{0,1\}\right) = A'_T$$

where A_T is the correct extremal frontier, using only tests in T, and which is computed by Algorithm 2.

Proof We prove $A_T = A'_T$ by induction.

For $|A_T| = |A'_T| = 1$, the proposition is trivially true since $A_\emptyset = \{0,1\}$.

In the inductive hypothesis we assume, $A_{T\setminus\{t\}} = A'_{T\setminus\{t\}}$ for all $t \in T$. Note that it is sufficient to prove $A_T \subseteq A'_T$; A_T is the true extremal frontier containing all undominated policies, so that applying $\mathrm{U}^*(\cdot)$ (*UpperHull*) ensures that $A_T \not\subset A'_T$.

To prove $A_T \subseteq A'_T$, assume $p \in A_T$ but $p \notin A'_T$. $p \notin A'_T$ implies $p \notin \{(0,0),(1,1)\}$, so $|T(p)| \geq 1$ and we can write without loss of generality $p = (a, X)$ where a is the root test. Further assume without loss of generality that a is fused with policies $p_1, \ldots, p_l \in A'_{T\setminus\{a\}}$. Then by the inductive hypothesis $p_1, \ldots, p_l \in A_{T\setminus\{a\}}$. In step 7 of the algorithm, a will be considered as the root test for *TestPrefix* with the set of actions $A'_{T\setminus\{a\}}$, and thus $p \in B_{T,a}$. But $p \notin A'_T$ implies that p is dominated by another (mixed) policy q in step 9 of the algorithm. This is a contradiction with the correctness of A_T. □

References

Avenhaus, R., Stengel, B. V., & Zamir, S. (1998). Inspection games. In R. J. Aumann & S. Hart (Eds.), *Handbook of game theory*, *Vol. III*.

Benson, H. P., & Lee, D. (1996). Outcome-based algorithm for optimizing over the efficient set of bicriteria linear programming problem. *Journal of Optimization Theory and Applications*, *88*(1), 77–105.

Bier, V. M., & Haphuriwat, N. (2010, to appear). Analytical method to identify the number of containers to inspect at US ports to deter terrorist attack. *Annals of Operations Research*.

Boros, E., Fedzhora, L., Kantor, P. B., Saeger, K., & Stroud, P. (2009). Large scale LP model for finding optimal container inspection strategies. *Naval Research Logistics*, *56*(5), 404–420.

Dantzig, G. (1957). Discrete-variable extremum problems. *Operations Research*, *5*(2), 266–277.

Dyer, M. E. (1984). An $O(n)$ algorithm for the multiple-choice knapsack linear program. *Mathematical Programming*, *29*, 57–63.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, *27*(8), 861–874.

Frittelli, J. (2005). Port and maritime security: background and issues for congress. *CRS Report RL31733*, US Library of Congress.

Jacobson, S. H., Karnani, T., & Kobza, J. E. (2005). Assessing the impact of deterrence on aviation checked baggage screening strategies. *International Journal of Risk Assessment and Management*, *5*(1), 1–15.

Glover, F., & Klingman, D. (1979). A $O(n \log n)$ algorithm for LP knapsacks with GUB constraints. *Mathematical Programming*, *17*, 345–361.

Ibaraki, T., Hasegawa, T., Teranaka, K., & Iwase, J. (1978). The multiple choice knapsack problem. *Journal of the Operations Research Society of Japan*, *21*(1), 59–95.

Kantor, P.B., & Boros, E. (2010). Deceptive detection methods for optimal security with inadequate budgets: the Testing Power Index. *Risk Analysis*, *30*(4), 663–673.

Kellerer, H., Pferschy, U., & Pisinger, D. (2004). *Knapsack problems*. Berlin: Springer.

Madigan, D., Mittal, S., & Roberts, F. (2007). Sequential decisions-making algorithms for port-of-entry inspection: Overcoming computational challenges. In *Proceedings IEEE Intelligence and Security Informatics*, New Brunswick, NJ (pp. 1–7).

Maschler, M. (1966). A price leadership method for solving the inspector's non-constant-sum game. *Naval Research Logistics Quarterly*, *13*, 11–33.

Pisinger, D. (1995). A minimal algorithm for the multiple-choice Knapsack problem. *European Journal of Operations Research*, *83*, 394–410.

Ramirez-Marquez, J. (2008). Port-of-entry safety via the reliability optimization of container inspection strategy through an evolutionary approach. *Reliability Engineering & System Safety*, *93*(11), 1698–1709.

Sinha, P., & Zoltners, A. A. (1979). The multiple-choice Knapsack problem. *Operations Research*, *27*(3), 503–515.

Slaughter, D. R., Accatino, M. R., Bernstein, A., Biltoft, P., Church, J. A., Descalle, M. A., Hall, J. M., Manatt, D. R., Mauger, G. J., Moore, T. L., Norman, E. B., Petersen, D. C., Pruet, J. A., & Prussin, S. G. (2007). The nuclear car wash: A system to detect nuclear weapons in commercial cargo shipments. *Nuclear Instruments and Methods in Physics Research Section A*, *579*(1), 349–352.

Stroud, P. D., & Saeger, K. J. (2003). Enumeration of increasing Boolean expressions and alternative digraph implementations for diagnostic applications. In H. Chu, J. Ferrer, T. Nguyen, & Y. Yu (Eds.), *Proceedings of computer communication and control technologies*, *Vol. IV* (pp. 328–333). Orlando: International Institute of Informatics and Systematics.

Zemel, E. (1980). The linear multiple choice Knapsack problem. *Operations Research*, *28*(6), 1412–1423.

Zemel, E. (1984). An $O(n)$ algorithm for the linear multiple choice knapsack problem and related problems. *Information Processing Letters*, *18*, 123–128.