Optimal Sequential Inspection Policies

Noam Goldberg[a] Jonathan Word[b] Endre Boros[c]
Paul Kantor[d]

RRR 14, October 1, 2008

[a]RUTCOR, Rutgers University, 640 Bartholomew Road, Piscataway NJ 08854-8003; ngoldberg@rutcor.rutgers.edu
[b]RUTCOR, Rutgers University, 640 Bartholomew Road, Piscataway NJ 08854-8003; jword@rutcor.rutgers.edu
[c]RUTCOR, Rutgers University, 640 Bartholomew Road, Piscataway NJ 08854-8003; boros@rutcor.rutgers.edu
[d]SCILS, Rutgers University; kantor@scils.rutgers.edu

Rutcor Research Report
RRR 14, October 1, 2008

Optimal Sequential Inspection Policies

Noam Goldberg Jonathan Word Endre Boros Paul Kantor

Abstract. We consider the problem of combining a given set of diagnostic tests into an inspection system that can classify items of interest (cases) with maximum accuracy subject to budget constraints. One motivating application is sequencing diagnostic tests for container inspection problems, where the diagnostic tests may correspond to radiation sensors, documents checks, or imaging systems. We consider mixtures of decision trees as inspection systems following the work of Boros, Fedzorah, Kantor, Saeger and Stroud [4]. We establish some properties of *efficient* inspection systems. Given an available set of elementary or compound tests (all are called "devices"), we characterize the optimal classification of cases, based on the the various "readings" or test scores given by the device. More generally, we consider the problem of optimally assigning a set of cases to a set of possible actions, based on the device "readings", while satisfying a budget constraint. The measure of performance is the fraction of all cases, in a specific class of interest, which are classified correctly. We find that this problem is a special case of a known variant of the knapsack problem, known as the Linear Multiple Choice Knapsack problem (LMCK). Exploiting the special features that we establish, we propose an algorithm that solves our special variant more efficiently than the existing LMCK algorithms. Finally, we propose a dynamic programming algorithm that enumerates all of the *efficient*, undominated, inspection policies in a two dimensional cost-detection space. Our inspection policies may sequence any arbitrary number of tests and are not restricted in the branching factor (or number of channels). Our approach directly solves the bi-criterion optimization problem of maximizing detection and minimizing cost, and thus supports sensitivity analysis over different budget or detection requirements, and the optimization of any monotone (possibly nonlinear) utility function over the efficient set.

1 Introduction

Inspection systems are mixtures of decision trees whose nodes are diagnostic tests and whose leaves are nodes representing the classes to be labeled, or assignment to actions. Particularly we are interested in assigning some items to the action of raising an alarm, when trying to identify a (single) type of threat. With more than 2 classes, the associated alarms or actions may correspond to different type of items or different severities. It is presumed that each test has an operating cost. The classification decision and subsequent action may incur an additional cost that is associated with rate of false positives (a.k.a. type I error).

The operating characteristics of tests can be expressed by a Receiver Operating Characteristic (ROC) curve [6] and by the cost parameters. These completely characterize the test. The ROC curve expresses the inspection system's detection rate as a function of the rate of false positives. From this curve, and the operating costs, and costs of false positives, we can construct a *cost-detection* curve, which is a fundamental mathematical entity of interest in our analysis.

A motivating application is the detection of nuclear and other potentially harmful material that could be smuggled into the country. One can not afford to open and manually inspect each and every container coming into the country, through ports and/or border crossings. Imperfect information about the contents of every package, vehicle or container are given by tests for the presence of nuclear material that range from different types of radiation sensors to simple document checks.

The operating goal is to detect *all* 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.

We propose an algorithm that finds an entire *efficient frontier* of inspection policies,

Acknowledgements: Funding by the 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.

thus displaying the optimal inspection strategy for each possible value of the budget. This frontier will aid decision makers in performing sensitivity analysis and evaluating cost-benefit, without restricting the analysis to any specific estimates of the prior probability and of the cost of false negatives.

As we sketch below, each inspection system can be represented as a tree. The number of branches at each node determines the complexity of the tree, and is determined by the number of different "readings" that the corresponding test may give. The problem of finding optimal inspection systems corresponding to binary decision trees has been considered by Stroud [17], and Madigan, Mittal and Roberts [13]. Stroud [17] has shown the total number T_N of possible binary decision trees involving 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 or final actions), the formula can be generalized, yielding:

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

The principal result of this paper is an efficient computation scheme that finds the optimal decision tree mixtures for all possible budget values, while avoiding a complete enumeration of all decision trees.

Boros, Fedzorah, Kantor, Saeger and Stroud [4] consider optimal decision tree inspection systems, with an arbitrary branching factor, subject to specific budget and other performance constraints. They present a large scale linear programming formulation. 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 or tests for container inspections. Future technologies being considered may require inspection systems to use a larger number of sensors. More significantly, multi-channel sensors represent a large number of conceptually distinct tests, as the 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 nuclear contraband, screening travelers at borders for drugs, screening at public events for weapons, and testing the public for diseases.

In the next section we will describe the background for our contribution and required definitions, and specifically the percise definitions of what we mean by inspection devices and policies. We proceed to characterize optimal inspection policies, in Section 4, by proving a monotonicity property of such policies. In Section 5 we model and propose an efficient algorithm for the problem of constructing optimal policies from a given device. In Section 6 we make use of this algorithm as a sub-procedure in a dynamic algorithm algorithm for enumerating all *efficient* policies. We find an upper bound for the number of efficient policies. Finally, we conclude after discussing our computational experiments (Section 8), comparing the performance of our algorithms with previous work by Boros et al. [4].

2 Background: Cases, Tests, Devices, Actions and Policies

Let us first introduce our basic terminology and notations. We consider a set U of *cases* which require some "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 specific *class* and we assume that each case in a certain class require a certain well-defined *action*. In this study we assume that cases are divided into two classes, "good" or "bad", i.e., $U = G \cup B$, and correspondingly, cases in class G should be *released* without further delays, while those in B should be subjected to a thorough manual *inspection*. We denote the corresponding set of actions by $A = \{I, R\}$.

To be able to detect the class of case, we make use of a set of *tests* (or sensors), each of which, when applied, provide (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 we denote by $c(t)$ 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 B.

Applications of tests, as well as incorrect applications are costly (e.g., inspecting manually an "innocent" good case delays trade and ties up resources, while releasing a bad case may have serious consequences later). Our main problem is to devise a plan to use the imperfect and costly tests to help us to arrive to decisions about our actions so that the overall expenses are minimized. To be able to formulate this problem precisely, we need to describe the information that we have about tests and actions in more detail.

We shall view a test $t \in \mathcal{T}$ as a *device*, filtering an input stream of cases into several channels (corresponding to the labels that it assigns). Formally, we associate to each test $t \in \mathcal{T}$, the set $\mathcal{C} = \mathcal{C}(t)$ of its *channels* (i.e., the set of possible labels the application of t can produce as a result). For each label $i \in \mathcal{C}$ we denote by $b_i \geq 0$ and $g_i \geq 0$ the fractions of "bad" and "good" containers, respectively, that are expected to receive label i in the long run. Note that index i here represents two pieces of information: the name of the test, and a specific channel associated to that test. Since every test must provide a label (channel assignment) for every case, we must have

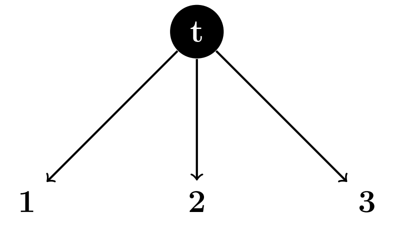
$$\sum_{i \in \mathcal{C}} b_i \;=\; \sum_{i \in \mathcal{C}} g_i \;=\; 1. \tag{2}$$

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

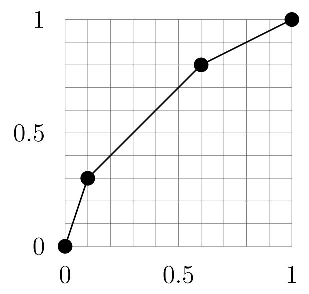
Let us note that although a label could be an arbitrary description of a category, in practice it will often be an integer score (for example the number of suspicious entries in a

[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.

shipping manifest), or a real number (such as in the case of radiation sensors). Accordingly the set $\mathcal{C}(t)$ 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 channel 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 *channels* $\mathcal{C}$. In what follows we assume that the set of channels is finite.



A: Test $\mathbf{t}$ with three channels, $\mathcal{C} = \{\mathbf{1}, \mathbf{2}, \mathbf{3}\}$, 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 $\mathbf{t}$. We have $(f_1, d_1) = (0.1, 0.3)$, $(f_2, d_2) = (0.6, 0.8)$ and $(f_3, d_3) = (1, 1)$.

Figure 1: An example for an elementary test and its ROC.

We will index channels in such a way that

$$\frac{b_1}{g_1} \geq \frac{b_2}{g_2} \geq \cdots \geq \frac{b_M}{g_M} \tag{3}$$

holds for all tests, where $M = |\mathcal{C}|$ is the number of channels. Given this sorting of the channels, it is easy to associate to this test the sequence of cumulative values, which can be represented as points in a two dimensional graph, (d_i, f_i), $i \in \mathcal{C}$, where

$$d_i = \sum_{j \leq i} b_j \qquad \text{and} \qquad f_i = \sum_{j \leq i} g_j \tag{4}$$

to completely (except for cost) represent the performance of this test. Adding the point $(d_0, f_0) = (0, 0)$, the piecewise linear function $d(f)$, determined by the points $\{(d_i, f_i) | i \in \mathcal{C} \cup \{0\}\}$, is known as the *Receiver Operating Characteristic curve* (or ROC) of the test . See Figure 1 for a simple example with three channels. Note that due to the sorting (3) and the equalities (2), the function $d(f)$ is concave, starting at $(0, 0)$ and ending at $(d_M, f_M) = (1, 1)$. When we talk about several tests, we shall refer to the ROC of test $t \in \mathcal{T}$ by $d^t(f)$, and by $d_i(t)$ and $f_i(t)$ for all $i \in \mathcal{C}(t)$ to the corresponding parameters as defined in (4).

Let us note that typically the outcome of the application of a specific test is a random variable. We assume in this study that the randomness of the test results comes essentially from differences in exogenous properties of cases, and only to a negligible extent from the 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 (except for some low level of human/computer error), we will find the same suspicious or incorrect 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, too. For example, several containers with exactly the same contraband my give different readings because of variations in other (shielding) cargo, or debris left from a previous shipment. But for a given container, all of 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 e.g., [17, 13]). The technical meaning of this assumption is that if we apply two tests: t and t' to all cases, then the fraction of e.g., "good" cases receiving label $i \in \mathcal{C}(t)$ from test t and $i' \in \mathcal{C}(t')$ 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, on 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 into several channels. Figure 2 shows such a complex system, combining three elementary tests $\mathcal{T} = \{a, b, c\}$, where a and b each have three channels, while c has only two. Such a representation, which forms a tree, 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 channels, and e.g., the device will label a case with channel label 6, if test a results in label 3, and the following test c assigns it to label 1.

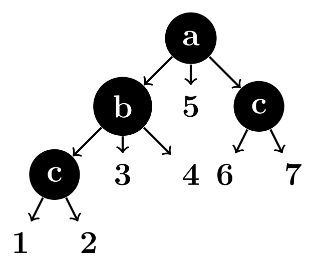


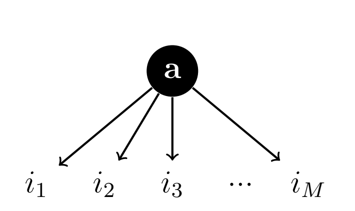
Figure 2: An example of a complex system that fuses multiple tests. The channels of each individual test (as well as the channels of the complex device) are numbered, in increasing order, from left to right.

We are now ready to give a formal representation of such complex systems, which we shall call devices. The result of "applying a device to a collection of $U = G \cup B$ "cases", of which G are good and B are harmful or "bad" is that some fraction of each will be assigned to each of the channels. The effect of the device (except for issues of cost) is completely described by a collection of fractions, corresponding to posterior probabilities of the classes, and subsets of tests describing each channel. (See e.g., Figure 3.)

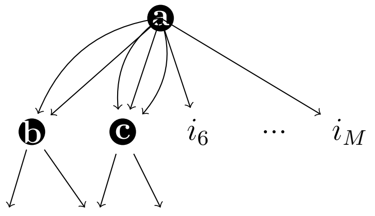
Definition 1 *A device D is defined as a set of triplets*

$$D = \{(b_i, g_i, T_i) \mid i \in \mathcal{C}\}, \tag{5}$$

where $\mathcal{C}$ is the set of (output) channels of D, the sets $T_i \subseteq \mathcal{T}$ are the subsets of elementary



A: An example of a general elementary test with M channels.



B: A compound device using the M-channel test as the root.

Figure 3: An elementary test with M channels and a compound device using the test as the root.

tests applied to cases that end up to be labeled by i, and $b_i \geq 0$ and $g_i \geq 0$ are respectively the expected fractions of the "bad" and "good" cases that are labeled by $i \in \mathcal{C}$.

When ambiguity could arise, we shall refer to these parameters of a device D explicitly as by $\mathcal{C}(D)$, $b_i(D)$, $g_i(D)$, and $T_i(D)$ for all $i \in \mathcal{C}(D)$, and we use $M(D) = |\mathcal{C}(D)|$ for the number of channels of the device D.

Furthermore, due to our assumption of stochastic independence of elementary tests, equation (2) implies that every device satisfies the following relations:

$$\begin{aligned} b_i(D) \; = \; \prod_{t \in T_i(D)} b_{i(t)}(t) \qquad \text{and} \qquad g_i(D) \; = \; \prod_{t \in T_i(D)} g_{i(t)}(t) \qquad \text{for all } i \in \mathcal{C}(D) \\ \sum_{i \in \mathcal{C}(D)} b_i(D) \; = \; \sum_{i \in \mathcal{C}(D)} g_i(D) \; = \; 1. \end{aligned} \tag{6}$$

Let us note that every channel $i \in \mathcal{C}(D)$ corresponds to a path in the corresponding decision tree, and hence to a unique channel $i(t) \in \mathcal{C}(t)$ for each tests $t \in T_i(D)$ 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

$$C(D, \pi) \; = \; \sum_{i \in \mathcal{C}} (\pi b_i + (1 - \pi) g_i) c(T_i) \tag{7}$$

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

Finally we note that it is possible to use a random mixture of devices in the following sense: Given two devices $D_1 = \{(b_i^1, g_i^1, T_i^1) | i \in \mathcal{C}(D_1)\}$, $D_2 = \{(b_i^2, g_i^2, T_i^2) | i \in \mathcal{C}(D_2)\}$, and a (probability) $0 \leq \lambda \leq 1$, we can define a new device $D_3 = \{(b_i^3, g_i^3, T_i^3) | i \in \mathcal{C}(D_3)\}$ in which, with probability λ we filter incoming cases using device D_1 and with probability $1-\lambda$ we filter them using device D_2. Both the label probabilities and the cost are linear in this parameter, and we have:

$$\mathcal{C}(D_3) \;=\; \{(1,j)|j \in \mathcal{C}(D_1)\} \cup \{(2,j)|j \in \mathcal{C}(D_2)\} \tag{8}$$

$$b_i^3 \;=\; \begin{cases} \lambda b_j^1 & \text{if } i=(1,j),\ j \in \mathcal{C}(D_1), \\ (1-\lambda) b_j^2 & \text{if } i=(2,j),\ j \in \mathcal{C}(D_2), \end{cases}$$

$$g_i^3 \;=\; \begin{cases} \lambda g_j^1 & \text{if } i=(1,j),\ j \in \mathcal{C}(D_1), \\ (1-\lambda) g_j^2 & \text{if } i=(2,j),\ j \in \mathcal{C}(D_2), \text{ and} \end{cases} \tag{9}$$

$$T_i^3 \;=\; \begin{cases} T_j^1 & \text{if } i=(1,j),\ j \in \mathcal{C}(D_1), \\ T_j^2 & \text{if } i=(2,j),\ j \in \mathcal{C}(D_2). \end{cases}$$

We shall write $D_3 = \lambda D_1 + (1-\lambda) D_2$ for such mixed devices.

An inspection *policy* is a device together with an assignment of an action to each of its channels. The cases which are assigned to a channel, will then be subjected to the corresponding action. Let us note here that labels assigned to cases and channels of a device (or test) are almost synonymous notions. The minor difference is that labels are assigned to cases at at a certain underlying physical level (e.g., the readings of a physical sensor), while channels of devices are defined at a logical level, where a channel may aggregate several labels. This possibility however does not lead to better solutions, and will not appear in this study. Therefore, we can think of labels and channels as synonyms in the sequel.

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. To each terminal action $\alpha \in A$ we associate its *cost* $C(\alpha)$ and its *detection rate* $\Delta(\alpha)$. The cost is normalized to represent the unit operating expense of executing the action. 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

$$\begin{aligned} C(R) &= 0 \qquad & C(I) &= 1 \\ \Delta(R) &= 0 \qquad & \Delta(I) &= 1 \end{aligned} \tag{10}$$

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 performance at a lower cost. Therefore we scale all of the operating cost coefficients so that $0 \le c(t) < 1$ for all (sensible) 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. Actually, we use this fact to present, in a simple way, our recursion-based solution. Before that, however, let us define formally what is meant by an inspection *policy*:

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

$$\sum_{\alpha \in A} x(i, \alpha) \; = \; 1 \qquad \textit{for all } i \in \mathcal{C}(D). \tag{11}$$

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 channel $i \in \mathcal{C}(D)$. For a budget that is insufficient for applying I to all cases, it may be optimal to choose an assignment of a channel to actions, that corresponds to such a random mixture. 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 the budget of $B = \frac{1}{2}$. In general, under a budget constraint the assignment to a mixture of actions provides a superior detection compared with a pure assignment that does not allow mixtures. In real life applications the budget may simply not be large enough to treat all cases with the action I.

Although we do not explicitly model our problem as a game, our approach recongizes that in practice it may be difficult to do so when there is a very large space of possible moves of the adversary, which are not fully known. With some partial information there may be an underlying game in our problem. The mixed policies will correspond to mixed strategies. The best possible payoff associated with a Nash equilibrium in mixed strategies must be at least as good as the best possible payoff associated with a Nash equilibrium (if it exists) in pure strategies. This is also evident in the optimization approach, i.e., applicable even if no information about player moves is available, where finding the optimal deterministic policy is an integer programming problem, while finding a mixed policy is the (linear) relaxation of that problem. Thus mixed policies may achieve a superior detection rate for each budget value compared with deterministic (pure) policies. Intuitively, mixed policies are also preferable because they are even more difficult to guess by an adversary.

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. In most applications the total cost, of a policy, depends not only on the operating cost of its associated tests, but also on an additional cost which results from false positives, i.e., the collateral damage imposed by inspection of harmless cases. We denote by E this additional expense. For example in the container inspection application, E is the expected cost of delaying traffic, or impeding commerce. The actual costs included in E will vary according to the definition of cases in a particular application. 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 the (terminal) action set $A = \{R, I\}$ and parameters defined as in (10), we can write the cost and detection of the policy as:

$$\begin{aligned}
C(P) &= \sum_{i\in\mathcal{C}(D)} \pi b_i(D)[c(T_i(D)) + x(i,I)] + (1-\pi)g_i(D)[c(T_i(D)) + x(i,I)(1+E)] \\
&= C(D) + \sum_{i\in\mathcal{C}(D)} \pi b_i(D)x(i,I) + (1-\pi)g_i(D)x(i,I)(1+E) \\
\\
\Delta(P) &= \sum_{i\in\mathcal{C}(D)} b_i(D)x(i,I).
\end{aligned} \tag{12}$$

Next we note that, in fact, terminal actions can be viewed as special cases of policies, where D is a device with one output channel 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 can view Definition 2 as a recursive structure, where the set A may involve some policies, as well as the terminal actions. Since we gain no additional information from repeating a test, we require some restrictions to the feasible action mappings. Let us denote by $T(P)$ the set of tests involved in policy P, where we define $T(I) = T(R) = \emptyset$.

We shall call a mapping $x : \mathcal{C}(D) \to [0,1]^A$ *feasible* in Definition 2 if it satisfies the further condition

$$x(i,p) \;=\; 0 \quad \text{whenever} \quad T_i(D) \cap T(p) \neq \emptyset \quad \text{for all } i \in \mathcal{C}(D) \text{ and } p \in A. \tag{13}$$

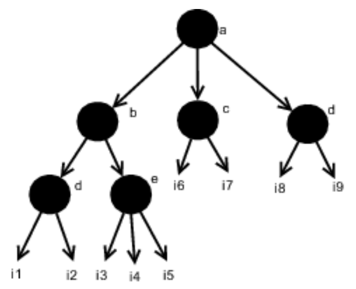
Henceforth, we consider Definition 2 extended so that the set A of actions may contain a finite set of (complex) policies as well as the terminal actions, and require that the corresponding action mapping x satisfies the conditions (13). See Figure 4 for an example of a general policy, composed of a top level device, and a set A involving both terminal actions and other policies.

2.1 Policy Mixing

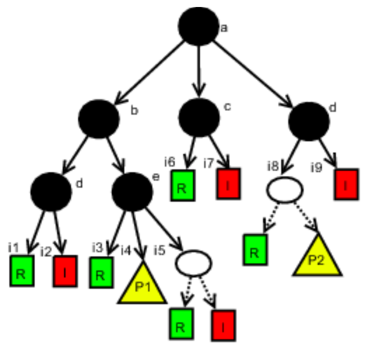
Let us next note that random mixing of policies is a meaningful operation, just as in the case of devices. Let us fix the action set for the moment. If $P_1 = (D_1, x_1)$ and $P_2 = (D_2, x_2)$ are two policies and $0 \leq \lambda \leq 1$ is a probability, then the policy $P_3 = \lambda P_1 + (1-\lambda)P_2$ can be defined as a pair (D_3, x_3), where $D_3 = \lambda D_1 + (1-\lambda)D_2$ is defined as in (8)-(9) and where

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

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



A: An example device. The set of channels is $\mathcal{C} = \{i_1, i_2, ..., i_9\}$. The set of tests associated with each channel 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 channel that mixes two or more policies.

Figure 4: An example device and a corresponding policy.

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

$$\begin{aligned} C(P_3) &= \lambda C(P_1) + (1-\lambda)C(P_2), \\ \Delta(P_3) &= \lambda \Delta(P_1) + (1-\lambda)\Delta(P_2). \end{aligned} \tag{14}$$

We say that a policy $P = (D, x)$ is a *deterministic policy*, if for all $i \in \mathcal{C}(D)$ and all $p \in A$ we have $x(i, p) \in \{0, 1\}$, i.e., no channel has an assignment to a mixture of actions. Although there are many possible policies, as long as the set of terminal actions is finite, and the number of channels 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* MIX$(\mathcal{P})$ *the set of policies obtainable from $\mathcal{P}$ by mixing, that is*

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

2.2 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 $T(P_2) \subseteq T(P_1)$, $C(P_2) \leq C(P_1)$ and $\Delta(P_2) \geq \Delta(P_1)$. If any of these inequalities are strict then we say that P_2 strictly dominates P_1.*

In other words, if P_1 is dominated by P_2, then we never need to use P_1. In any situation where we would use P_1 (even inside of a more complex policy), we can simply replace it by P_2, and this will not increase the expected cost of the total operation, will not violate the no-repeat assumption (since we assume $T(P_2) \subseteq T(P_1)$), and may even increase the resulting detection rate.

Let us observe a few more easy facts about mixtures of policies, and introduce some additional useful notions.

Claim 1 *Assume:*

i *$Q = \sum_{j=1}^{q} \lambda_j S_j$ is a mixture of policies $\mathcal{S} = \{S_j \mid j = 1, ..., q\}$, such that $\lambda_j > 0$ for all $j = 1, 2, .., q$, $q \geq 3$ and $\sum_{j=1}^{q} \lambda_j = 1$*

ii *the convex hull of the policies in the two dimensional cost-detection space, $K = \operatorname{conv}\{(C(S), \Delta(S)) \mid S \in \mathcal{S}\}$, has a nonempty interior.*

Then Q is (strictly) dominated by another mixture $P \in \mathrm{MIX}(\mathcal{S})$ of these policies.

Proof. 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 $Y = \Delta(P)$.

Since the point $(C(Q), \Delta(Q)) \in K$ belongs to the interior of K, due to $\lambda_j > 0$ being positive for $j = 1, 2, ..., q$, the point $(C(Q) - \varepsilon, \Delta(Q) + \varepsilon)$ also belongs to 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 $\mathrm{U}(\mathcal{P}) \subseteq \mathcal{P}$ subset of its un-dominated policies. Let us further denote by $\mathrm{U}^*(\mathcal{P}) \subseteq \mathrm{U}(\mathcal{P})$ the unique minimal subset of un-dominated 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 not dominated (not necessarily strictly) by any of $\mathrm{MIX}(\mathcal{P} \setminus \{P\})$.

Since random mixing of policies satisfy equations (14) and since no strictly dominated policy can be the best choice for any budget value, it follows that for any finite set of policies $\mathcal{P}$ the function

$$\Delta(B) \;=\; \max\{\Delta(P) \mid P \in \mathrm{MIX}(\mathcal{P}), \quad C(P) \leq B\}$$

is a piecewise linear concave function with break points corresponding to $\mathrm{U}^*(\mathcal{P})$. Assuming meaningful policies in $\mathcal{P}$, this curve is defined for $B \in [a, b] \subseteq [0, 1]$, where $a = \min\{C(P) \mid$

$P \in \mathcal{P}\}$ and $b = \max\{C(P) \mid P \in \mathcal{P}\}$. We can also see 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

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

when ambiguity could arise, we shall denote this curve by $\Delta_{\mathcal{P}}(B)$, and will say that the set $\mathrm{U}^*(\mathcal{P})$ and the curve $\Delta_{\mathcal{P}}(B)$ form the *extremal frontier* of the set $\mathcal{P}$. Note that when the two trivial policies (namely applying R and I, respectively) belong to $\mathcal{P}$ the extremal frontier is necessarily a nondecreasing concave curve connecting $(0, 0)$ to $(1, 1)$.

Let us note next that if $\mathcal{P}_i$, $i = 1, 2$ 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 we leave for the reader.

Let us close this section by observing that $\mathrm{U}^*(\mathcal{P})$ can efficiently be computed for any finite set $\mathcal{P}$.

Claim 2 *Assume that $\mathcal{P}$ is a finite set of policies, given in sorted order of cost, and $\mathcal{Q} = \mathrm{U}^*(\mathcal{P})$. The subset $\mathcal{Q} \subseteq \mathcal{P}$ can be determined in $O(|\mathcal{P}|)$ time.*

Proof. We claim that the following simple procedure computes $\mathrm{U}^*(\mathcal{P})$ in the claimed time.

Algorithm 1 *UpperHull*$(\mathcal{P})$

```
 1: Input: A set of policies $\mathcal{P} = \{P_1, ..., P_M\}$.
 2: Assumptions: W.l.o.g. we assume an increasing order of costs, $C(P_1) < ... < C(P_m)$
    (if the inequality is not strict, it is enough to keep the one with the largest detection
    rate).
 3: Initializations: Set $i_1 \leftarrow 1$, $i_2 \leftarrow 2$, $q \leftarrow 2$, $t \leftarrow 3$.
 4: Main Loop:
 5: while $t \leq M$ do
 6:     while $\lambda(q+1) = \frac{\Delta(P_t) - \Delta(\mathcal{Q}_q)}{C(P_t) - C(\mathcal{Q}_q)} \geq \lambda(q) = \begin{cases} \frac{\Delta(P_{i_q}) - \Delta(P_{i_{q-1}})}{C(P_{i_q}) - C(P_{i_{q-1}})} & \text{if } q \geq 2 \\ \infty & \text{if } q = 1 \end{cases}$ 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: $\mathcal{Q} = \mathrm{U}^*(\mathcal{P}) = \{P_{i_1}, P_{i_2}, ..., P_{i_q}\}$.
```

Let us next note that we execute the the main loop $O(M)$ times. Let us also observe that we must have $\lambda(1) > \lambda(2) = \frac{\Delta(P_{i_2}) - \Delta(P_{i_1})}{C(P_{i_2}) - C(P_{i_1})} > \cdots > \lambda(q) = \frac{\Delta(P_{i_q}) - \Delta(P_{i_{q-1}})}{C(P_q) - C(P_{q-1})}$. This is because if we had $\lambda(i_j) \leq \lambda(i_{j+1})$, then we would delete i_j before adding i_{j+1} in step 6 of the algorithm. Thus the points in the output set $\mathcal{Q}$ indeed form a concave curve, hence not one of them is 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. If removing $P_{i_q} = P_t$ we never access policy P_t again.

Thus we must have $\mathcal{Q} = \mathrm{U}^*(\mathcal{P})$, and the total running time is $O(M)$. $\square$

3 The Inspection Problem

The decision maker's problem is to find a policy, using 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 detection rate and minimizing cost are competing objectives, and cannot be simultaneously optimized. One approach is to minimize a single unconstrained objective function: $Z = C(P) + \pi K(1 - \Delta(P))$, 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, for a catastrophic event. However, both parameters: π and K, may be difficult to estimate in practice.

Let us observe that this objective consists of two parts of different nature: the term $\pi K(1-\Delta(P))$ is an expense in the future, and estimates the total expected damage resulting from the mistake of not detecting a "bad" case. Not only are we unable to reliably estimate the product πK, in addition, *we cannot influence this part of the product.* 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 part $C(P)$ represents a real operating cost together with the cost of false positives, which we can estimate in the long run much more reliably, and which we can influence by choosing the right policy P.

Therefore, we consider a natural reformulation of the problem:

$$\max_{P \in \mathcal{P}} \{\Delta(P) \mid C(P) \leq B\} \tag{16}$$

where B is a given budget limit for the unit operating cost of the policy, and $\mathcal{P}$ is the set of policies that can be constructed from a given set of elementary tests and terminal actions. To increase the practical impact of the analysis, we would like to solve Problem (16) for many different values of the budget B. This will permit a decision maker to evaluate the marginal gain in safety from a particular budget increase, or determine the lowest budget achieving a specified safety level, and so on.

Therefore, we view the maximum detection rate which solves Problem (16) as a function $\Delta(B)$ of the budget. We seek to determine this function for the entire range of budget values.

For the rest of this paper we assume that π is negligible in comparison with 1. [2]. This assumption further simplifies $C(P)$, and the analysis:

$$C(P) \;=\; \sum_{i \in \mathcal{C}(D)} g_i(D)[c(T_i(D)) + x(i,I)(1+E)]. \tag{17}$$

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

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

The Discrete Inspection Problem

Input: Consider a set $\mathcal{T}$ of N elementary tests and the set $A = \{R, I\}$ of terminal actions, given as in Section 2, and denote by $\mathcal{P}$ the family of all non-repeating policies using these test and terminal actions.

Task: Determine the function $\Delta(B)$, for $B \in [0,1]$, defined by the optimization problem (16).

Our main result is that, in fact $\Delta(B)$ is a piecewise linear function defined by a finite set of deterministic policies $\mathcal{P}^* = \mathrm{U}^*(\mathcal{P})$. We provide an efficient algorithm to determine these policies, and develop an upper bound on the size of $\mathcal{P}^*$.

4 Monotonicity of Optimal Policies

Our plan to solve the Discrete Inspection Problem is to build iteratively more and more complex devices from simpler ones, in an optimal way. The initial steps are the elementary actions. In a general step, we have some already generated policies and we include all of them in the set A of actions. We now want to prefix a given device $D = \{(b_i, g_i, T_i) \mid i \in \mathcal{C}\}$ to a policy with these actions in an optimal way. Let us recall, by definition that a policy $P = (D, x)$ involves a device D and an assignment $x : \mathcal{C}(D) \times A \to [0,1]$ of actions to its channels.

To guide our reasoning, we first observe certain properties of optimal policies.

Lemma 1 (Monotonicity) *Assume that $x^* : \mathcal{C}(D) \to [0,1]^A$ satisfies (13) (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 channels 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 either $x^*(i,p) = 0$ or $x^*(j,q) = 0$.*

[2]An alternative assumption, which may lead to a similar simplification, is that E is very large $E >> 1$.

In other words, if two different actions (policies) could equally be assigned to any of two channels, then in an optimal assignment we cannot assign the weaker detection rate action to the channel with a higher b/g ratio, if we have assigned a stronger action to the channel with the lower b/g ratio.
Proof. Note that the feasibility conditions give the relation, for any channel and policy, that the fraction of the bad items detected by assigning policy r to channel k, is the product $b_k\Delta(r)$, and thus we can write

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

Analogously, the cost of policy P can be written as

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

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

$$\varepsilon g_i = \varepsilon' g_j. \tag{18}$$

Let us next define a new policy P' by changing

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

and letting $x'(k,r) = x^*(k,r)$ for all other combinations of $k \in \mathcal{C}$ 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'. We have

$$C(P') - C(P) \ = \ -\varepsilon g_iC(p) + \varepsilon g_iC(q) - \varepsilon' g_jC(q) + \varepsilon' g_jC(p) \ = \ (C(p) - C(q))(\varepsilon' g_j - \varepsilon g_i) \ = \ 0$$

by (18). Furthermore we have

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

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

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

Let us note that in this optimal action assignment x there is at most one channel i (the one with the smallest value of b_i/g_i among those to which we assign action I with a positive $x(i,I)$ value) where both $x(i,I)$ and $x(i,R)$ may be nonnegative. For all other channels x is a binary assignment. Let us introduce x^α for $\alpha = 0,1$ by defining

$$x^\alpha(i,I) = \alpha,\ \ x^\alpha(i,R) = 1-\alpha,\ \ \text{and } x^\alpha(j,I) = x(j,I),\ \ x^\alpha(j,R) = x(j,R)\ \text{ for all other channels } j \neq i.$$

Then the policies $P_\alpha = (D, x^\alpha)$ for $\alpha = 0,1$ are deterministic, assuming that D is deterministic, and we have

$$(D,x) \ = \ x(i,I)P_1 \ + \ x(i,R)P_0.$$

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.,*

$$\mathcal{P}(D,A) \ = \ \{(D,x) \mid x : \mathcal{P} \times A \longrightarrow [0,1]\}.$$

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

5 Fusion of Tests

Given a new device D (which maybe a single test s) and a set of available actions $A_i \supseteq \{I,R\}$ for each of its channels $i \in \mathcal{C}(D)$, we can create a new policy P by *prefixing* device D to these subsets of the available policies. A required condition is (13), that is that $T_i \cap T(p) = \emptyset$ for assignments $x(i,p) > 0$. In the following we assume that for all $i \in \mathcal{C}(D)$ and $p \in A_i$ we have $T_i \cap T(p) = \emptyset$, and do not state it explicitly. Let us note that the sets of actions may coincide, or have a substantial overlap. For instance, in the important case when D is a single test s, we can assume without any loss of generality that $A_i = A$ for all $i \in \mathcal{C}(s)$. Introducing $\mathcal{A} = \{A_i \mid i \in \mathcal{C}(D)\}$, let us denote by $\mathcal{P}(D,\mathcal{A})$ the family of policies one can create in this way, i.e.,

$$\mathcal{P}(D,\mathcal{A}) = \left\{ (D,x) \mid x : \mathcal{C}(D) \times \left(\bigcup_{i \in \mathcal{C}(D)} A_i \right) \to [0,1], \right.$$
$$\left. \text{satisfies (11) and } x(i,p) = 0 \text{ for all } i \in \mathcal{C}(D),\ p \notin A_i \ \right\}$$

For a specific budget B, the problem of finding the policy $P \in \mathcal{P}(D,\mathcal{A})$ for which $C(P) \leq B$ and $\Delta(P)$ is the highest, can be stated as:

$$\Delta(P) = \max \sum_{i\in\mathcal{C}(D),p\in A_i} b_i \Delta(p) x(i,p) \tag{19}$$

Subject to:

$$\sum_{i\in\mathcal{C}(D),p\in A_i} g_i C(p) x(i,p) \leq B - C(D) \tag{20}$$

$$\sum_{p\in A_i} x(i,p) = 1 \qquad \text{for each } i \in \mathcal{C}(D) \tag{21}$$

$$x(i,p) \geq 0 \qquad \text{for all } i \in \mathcal{C}(D) \text{ and } p \in A_i \tag{22}$$

We shall solve test fusion for the entire range of possible budget values $C(D) \leq B \leq 1$. For this we consider policies P which could be built from device D by choosing a binary assignment $x : \mathcal{C}(D) \times \left(\bigcup_{i\in\mathcal{C}(D)} A_i\right) \longrightarrow \{0,1\}$ satisfying (11) and (13). Note that all other policies corresponding to some optimal non-binary assignment can be obtained by mixing from the ones corresponding to binary assignments, due to Corollary 2. Then we drop from this large set all those policies which are dominated by some other policies of this set, and denote by $\mathcal{Q}^* = \mathrm{U}^*(\mathcal{P}(D,\mathcal{A}))$ the remaining policies forming the extremal frontier of this test fusion problem. We know that $\mathcal{Q}^*$ defines a piecewise-linear concave $\Delta(B)$ function for $C(D) \leq B \leq 1$. Let us note that the best policies we can build from policy sets A_i, $i \in \mathcal{C}(D)$ and device D is the set $\mathcal{P}^* = \mathrm{U}^*(A \cup \mathcal{Q}^*)$.

In the Algorithm 2 itself we generate the set of $(C(P), \Delta(P))$ pairs for $P \in \mathcal{P}^*$ as well as the corresponding binary assignments $x(i,p) \in \{0,1\}$ for all $i \in \mathcal{C}(D)$ and $p \in A_i$, which uniquely defines the corresponding policies.

To see the correctness of this procedure, that is to see that $\mathcal{Q}^* = \mathrm{U}^*(\mathcal{P}(D,\mathcal{A}))$, we provide both an intuitive and a formal argument.

Our assumptions (A1), (A2), and (A3) make sure that any action $p \in A_i$ is feasible to assign to channel $i \in \mathcal{C}(D)$, that all action sets A_i, $i \in \mathcal{C}(D)$ contain the two terminal actions R and I, and finally that these action sets are sorted in their natural order of increasing costs and do not contain dominated actions (we note that this assumption is w.l.o.g. since, by Claim 2, $US(A_i)$ can be found in time $O(|A_i|)$). Due to these assumptions, we have

$$\sum_{i\in\mathcal{C}(D)} \sum_{j=1}^{M_i} \mathbf{v}^{i,j} \;=\; (1,1) \tag{23}$$

implying that the set $\mathcal{Q}^*$ corresponds to a sequence from $(C(D), 0)$ to $(1 + C(D), 1)$. In the algorithm we sum up the vectors $\mathbf{v}^{i,j}$-s in a decreasing order of slopes. Thus the main iteration we can also interpret as choosing a channel $i \in \mathcal{C}(D)$, and then for that i we replace action $Q_{i,j(i)}$ by $Q_{i,j(i)+1}$. Thus, we have after every iteration k that $j(i_k) = j_k$, and action Q_{i_k,j_k} is assigned to channel i_k in policy P_k. This further implies that $(\gamma_k, \delta_k) = (C(P_k), \Delta(P_k))$

```
Algorithm 2 $TestFusion(D, \mathcal{A})$
  Input: A device $D = \{(b_i, g_i, T_i) \mid i \in \mathcal{C}(D)\}$, and a family $\mathcal{A} = \{A_i \mid i \in \mathcal{C}(D)\}$ of policy sets, where $A_i = \{Q_{i,0}, Q_{i,1}, ..., Q_{i,M_i}\}$ for $i \in \mathcal{C}(D)$.
  Initializations:
   (I1) $\mathcal{Q}^* \leftarrow \emptyset$, $k \leftarrow 0$, $\delta_0 \leftarrow 0$, and $\gamma_0 \leftarrow C(D)$
   (I2) $C_{i,j} \leftarrow C(Q_{i,j})$ and $\Delta_{i,j} \leftarrow \Delta(Q_{i,j})$ for all $i \in \mathcal{C}(D)$ and $j = 0, ..., M_i$
   (I3) $\mathbf{v}^{i,j} \leftarrow (g_i(C_{i,j} - C_{i,j-1}), b_i(\Delta_{i,j} - \Delta_{i,j-1}))$ for all $i \in \mathcal{C}(D)$ and $j = 1, ..., M_i$
   (I4) $\lambda_{ij} \leftarrow \frac{b_i(\Delta_{i,j} - \Delta_{i,j-1})}{g_i(C_{i,j} - C_{i,j-1})}$ for all $i \in \mathcal{C}(D)$ and $j = 1, ..., M_i$
   (I5) $j(i) \leftarrow 0$ for all $i \in \mathcal{C}(D)$
   (I6) $x(i, Q_{i,0}) \leftarrow 1$, $x(i, Q_{i,j}) \leftarrow 0$ for all $i \in \mathcal{C}(D)$ and $j = 1, ..., M_i$, and set $P_0 \leftarrow (D, x)$
  Assumptions:
  (A1) $T_i \cap T(Q_{i,j}) = \emptyset$ for all $i \in \mathcal{C}(D)$ and $j = 0, ..., M_i$
  (A2) $C(D) < 1$, $0 = C_{i,0} < C_{i,1} < \cdots < C_{i,M_i} = 1$, $\Delta_{i,0} = 0$, and $\Delta_{i,M_i} = 1$ for all $i \in \mathcal{C}(D)$
  (A3) $\mathrm{U}^*(A_i) = A_i$, that is the points $\{(C_{i,j}, \Delta_{i,j}) | j = 0, ..., M_i\}$ form a concave curve, implying $\lambda_{i1} > \lambda_{i2} > \cdots > \lambda_{iM_i}$ for all $i \in \mathcal{C}(D)$
  Main Loop:
  while $\exists i \in \mathcal{C}(D)$ such that $j(i) < M_i$ do
    $\mathcal{Q}^* \leftarrow \mathcal{Q}^* \cup \{P_k\}$
    $k \leftarrow k + 1$
    $(i_k, j_k) \leftarrow \underset{i,j}{\operatorname{argmax}}\{\lambda_{ij} | i \in \mathcal{C}(D) \text{ and } j(i) < j \leq M_i\}$
    $(\delta_k, \gamma_k) \leftarrow (\delta_{k-1}, \gamma_{k-1}) + \mathbf{v}^{i_k, j_k}$
    $x(i_k, Q_{i_k, j_k - 1}) \leftarrow 0$, $x(i_k, Q_{i_k, j_k}) \leftarrow 1$, and set $j(i_k) = j_k$
    $P_k \leftarrow (D, x)$ (the policy corresponding to the current solution),
  end while
  Output: The family of policies $\mathcal{Q}^*$.
```

holds for all iterations k. Furthermore, moving along vector $\mathbf{v}^{i_k,j_k}$ can also be interpreted as considering a mixture of policies P_{k-1} and P_k, such that for every ϵ increase in the total cost value we get λ_{i_k,j_k} extra detection. Since we choose λ_{i_k,j_k} as the maximum of all possible choices, intuitively it is clear that we do the best we can in the process, to get the highest detection for a given budget.

We can also provide a formal proof for the correctness of the above procedure, by showing that procedure $TestFusion(D, \mathcal{A})$ provides us with an optimal solution to the linear programming problem (19)–(22).

Claim 3 *Assume that our budget is $B = (1-\alpha)\gamma_{k-1} + \alpha\gamma_k$ for some real $0 \leq \alpha \leq 1$. Let us run the above algorithm, and stop in the kth iteration, and consider the assignment defined by*

$$x^*(i, Q_{i,j}) \;=\; \begin{cases} 0 & \text{if } (i \neq i_k \text{ and } j \neq j(i)) \text{ or } (i = i_k \text{ and } j < j_{k-1} \text{ or } j > j_k) \\ 1 & \text{if } i \neq i_k \text{ and } j = j(i) \\ 1-\alpha & \text{if } i = i_k \text{ and } j = j_{k-1} \\ \alpha & \text{if } i = i_k \text{ and } j = j_k \end{cases}$$

Then x^* is an optimal solution of the LP (19)–(22). In other words, the policy $P^* = (D, x^*) = (1-\alpha)P_{k-1} + \alpha P_k$ is optimal in the problem

$$\max\{\Delta(P) \mid P \in \mathcal{P}(D, \mathcal{A}),\; C(P) \leq B\}.$$

Let us note that any budget value $C(D) \leq B \leq 1 + C(D)$ is of the form claimed in the above Claim for some iteration k and real $0 \leq \alpha \leq 1$, due to (23).

Proof. It is immediate to verify that x^* is a feasible solution in the LP (19)–(22), and that it defines the policy P^*. Furthermore, due to the facts that $C(P_{k-1}) = \gamma_{k-1}$ and $C(P_k) = \gamma_k$, we have equality in the budget constraint (20). 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 problem:

$$\min W = y(B - C(D)) + \sum_{i \in \mathcal{C}(D)} z_i \tag{24}$$

Subject to:

$$y g_i C(p) + z_i \geq b_i \Delta(p) \text{ for all } i \in \mathcal{C}(D) \text{ and } p \in A_i \tag{25}$$

$$y \geq 0 \tag{26}$$

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

$$z_i^* \;=\; \max_{p \in A_i} b_i \Delta(p) - y^* g_i C(p).$$

It is immediate to see that this defines a feasible solution to the above dual LP.

Since we sum up the $\mathbf{v}^{i,j}$ vectors in decreasing slope order, we have

$$z_i^* = \begin{cases} b_i\Delta_{i,j(i)} - y^* g_i C_{i,j(i)} & \text{if } i \neq i_k \\ b_{i_k}\Delta_{i_k,j_{k-1}} - y^* g_{i_k} C_{i_k,j_{k-1}} \ = \ b_{i_k}\Delta_{i_k,j_k} - y^* g_{i_k} C_{i_k,j_k} & \text{if } i = i_k. \end{cases} \tag{27}$$

Let us note that these (i,p) pairs appearing in the above formula are exactly those for which $x^*(i,p) > 0$. Thus we can rewrite $W^* = y^*(B - C(D)) + \sum_{i\in\mathcal{C}(D)} z_i^*$ as

$$\begin{aligned}
&= \ y^*(B - C(D)) + \sum_{\substack{i\in\mathcal{C}(D)\\ i\neq i_k}} z_i^* \ + \ (1-\alpha) z_{i_k}^* + \alpha z_{i_k}^* \\
&= \ y^*(B - C(D)) + \sum_{\substack{i\in\mathcal{C}(D)\\ i\neq i_k}} \left(b_i\Delta_{i,j(i)} - y^* g_i C_{i,j(i)}\right) \\
&\qquad + \ (1-\alpha)\left(b_{i_k}\Delta_{i_k,j_{k-1}} - y^* g_{i_k} C_{i_k,j_{k-1}}\right) \ + \ \alpha\left(b_{i_k}\Delta_{i_k,j_k} - y^* g_{i_k} C_{i_k,j_k}\right) \\
&= \ y^*(B - C(D)) + \sum_{\substack{i\in\mathcal{C}(D)\\ i\neq i_k}} x^*(i, Q_{i,j(i)})\left(b_i\Delta_{i,j(i)} - y^* g_i C_{i,j(i)}\right) \\
&\qquad + \ x^*(i_k, Q_{i_k,j_{k-1}})\left(b_{i_k}\Delta_{i_k,j_{k-1}} - y^* g_{i_k} C_{i_k,j_{k-1}}\right) \ + \ x^*(i_k, Q_{i_k,j_k})\left(b_{i_k}\Delta_{i_k,j_k} - y^* g_{i_k} C_{i_k,j_k}\right) \\
&= \ y^*\left(B - C(D) - \sum_{\substack{i\in\mathcal{C}(D)\\ i\neq i_k}} x^*(i, Q_{i,j(i)}) g_i C_{i,j(i)} \ - \ x^*(i_k, Q_{i_k,j_{k-1}}) g_{i_k} C_{i_k,j_{k-1}} \ - \ x^*(i_k, Q_{i_k,j_k}) g_{i_k} C_{i_k,j_k}\right) \\
&\qquad + \ \left(\sum_{\substack{i\in\mathcal{C}(D)\\ i\neq i_k}} x^*(i, Q_{i,j(i)}) b_i\Delta_{i,j(i)} \ + \ x^*(i_k, Q_{i_k,j_{k-1}}) b_{i_k}\Delta_{i_k,j_{k-1}} \ + \ x^*(i_k, Q_{i_k,j_k}) b_{i_k}\Delta_{i_k,j_k}\right)
\end{aligned}$$

Since all other components of x^* are zero, and since x^* is a solution satisfying the budget constraint (20) with equality, we have

$$\begin{aligned}
&\left(B - C(D) - \sum_{\substack{i\in\mathcal{C}(D)\\ i\neq i_k}} x^*(i, Q_{i,j(i)}) g_i C_{i,j(i)} \ - \ x^*(i_k, Q_{i_k,j_{k-1}}) g_{i_k} C_{i_k,j_{k-1}} \ - \ x^*(i_k, Q_{i_k,j_k}) g_{i_k} C_{i_k,j_k}\right) \\
&\qquad = \ B - C(D) - \sum_{\substack{i\in\mathcal{C}(D)\\ p\in A_i}} x^*(i,p) g_i C(i,p) \ = \ 0
\end{aligned}$$

Furthermore, using again that all other components of x^* are zero we have

$$\begin{aligned}
&\left(\sum_{\substack{i\in\mathcal{C}(D)\\ i\neq i_k}} x^*(i, Q_{i,j(i)}) b_i\Delta_{i,j(i)} \ + \ x^*(i_k, Q_{i_k,j_{k-1}}) b_{i_k}\Delta_{i_k,j_{k-1}} \ + \ x^*(i_k, Q_{i_k,j_k}) b_{i_k}\Delta_{i_k,j_k}\right) \\
&\qquad = \textstyle\sum_{\substack{i\in\mathcal{C}(D)\\ p\in A_i}} x^*(i,p) b_i \Delta(i,p).
\end{aligned}$$

Thus we can conclude that

$$W^* = \sum_{\substack{i \in \mathcal{C}(D) \\ p \in A_i}} x^*(i,p) b_i \Delta(i,p),$$

showing the the dual solution (y^*, z^*) has the same objective function value as the primal x^*, implying by LP duality that x^* is optimal. $\square$

Next we analyze the complexity of $TestFusion(D, \mathcal{A})$.

Claim 4 *The runtime complexity of algorithm* $TestFusion(D, \mathcal{A})$ *is* $O(\sum_{i \in \mathcal{C}(D)} |A_i| \log |\mathcal{C}(D)|)$ *if all sets* A_i *are given in sorted order.*

Proof. Since by our assumption (A3) we have all the $\lambda_{i,j}$, $j = 1, ..., M_i$ values sorted for all $i \in \mathcal{C}(D)$, we need to merge these sorted lists in order to be able to run the Main Loop. Merging $|\mathcal{C}(D)|$ ordered sets can be done in $O(\sum_{i \in \mathcal{C}(D)} |A_i| \log |\mathcal{C}(D)|)$ time, and this merging has to be done only once, at the beginning. All initializations can be done in $O(\sum_{i \in \mathcal{C}(D)} |A_i|)$ time. The main loop can be executed in $O(1)$ time, and it is executed exactly $\sum_{i \in \mathcal{C}(D)} |A_i|$ times. Thus the claim follows. $\square$

The linear programming formulation (19)–(22) is a special case of the Linear Multiple Choice Knapsack problem (LMCK) [16, 18]. 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. Each item has a value and a cost. The objective is to maximize the value of a selected set of items subject to a budget constraint. Sinha and Zoltners [16] propose a greedy algorithm and characterize the undominated solution space of the LMCK. Although Sinha and Zoltners do not make a complexity claim the run time complexity of their algorithm as stated is $O(k^2)$ where k is the total number of items. Although the algorithm [16] is designed to solve the problem LMCK with a given budget value, it may be used to find an entire efficient frontier of optimal policies for a range of costs up to the given budget value. Other algorithms that can be used to generate the entire frontier have been described in the literature in the context of solving relaxations of different variants of the integer Multiple Choice Knapsack Problem. Ibaraki, Hasewaga, Teranaka and Iwase [10] suggest a dual based algorithm for the problem when constraints (21) are inequalities. Note that this is the case in our application when $R \in A_i$ for all $i \in \mathcal{C}(D)$. The complexity of their algorithm is $O(k \lg k)$. Glover and Klingman [9] also propose an $O(k \lg k)$ algorithm that specializes the dual simplex method for the LMCK problem, and also results in a greedy algorithm. Zemel [18] generalizes the LMCK problem, to allow knapsack items that are contained in any multiple choice set, and improves on the algorithm's complexity. Zemel's [18] algorithm involves a transformation of the LMCK instance into the Continuous Knapsack Problem (CKP). The complexity of the LMCK algorithm [18] is $O(k \lg k_{max})$ where k is the total number of items, and k_{max} is the maximum number of items in a multiple choice set, when using an $O(k)$ time algorithm (Balas and Zemel) to solve the CKP problem. When the multiple choice sets are already sorted the greedy algorithm [18] can be used to solve LMCK in $O(k)$. Zemel suggests using the $O(k)$ algorithm for the CKP, for the sake of the computational complexity of the LMCK problem for a specific budget value and thus does not allow one to find the entire efficient frontier. Whether Zemel's transformation [18] can be used to find an entire efficient frontier of optimal policies depends on the CKP algorithm used. We note that it can be used to find the entire frontier when using Dantzig's greedy algorithm [5] (see also Kellerer et al. []), with a run time complexity of $O(k \lg k)$. Finally, Pisinger [15] describes a simple greedy algorithm, based on the the algorithms of Sinha and Zoltner [16] and Zemel [18], with a complexity of $O(k \lg k)$.

Although a faster, $O(k)$ algorithm has been found for the LMCK problem, independently by Dyer and Zemel (see [12]), it does not find the entire curve $\Delta(C)$ for $C \in [C(D), B]$, as do the greedy LMCK algorithms [16, 15, 9, 10].

Note that we can view each set of actions A_i available to channel $i \in \mathcal{C}(D)$ as an LMCK multiple choice set, and each pair (i, p), of a channel $i \in \mathcal{C}(D)$ and a policy $p \in A_i$, as a knapsack item with value $b_i \Delta_p$ and cost $g_i C(p)$. Thus we can apply the greedy algorithm to solve the above test fusion problem. In our case we have $k = \sum_{i \in \mathcal{C}(D)} |A_i|$ knapsack items. Therefore, the greedy algorithms that implicitly enumerate the entire efficient frontier [15, 9, 10], or the transformation of Zemel [18] into the CKP when the latter is solved by Dantzig's greedy algorithm, all have a worst case complexity of $O(\sum_{i \in \mathcal{C}(D)} |A_i| \lg \sum_{i \in \mathcal{C}(D)} |A_i|)$.

Our Algorithm 2 differs from the greedy algorithms [18, 16, 9, 10], in that it explicitly enumerates the entire curve of efficient policies. We concentrate on a special case of particular interest in our application of fusing tests; when the sets of actions A_i are already in sorted order. The algorithm makes use of the fact that our multiple choice sets are ordered by cost. Of particular interest is the worst case complexity, which is $O(\sum_{i \in \mathcal{C}(D)} |A_i| \log |\mathcal{C}(D)|)$ provided that the sets of available actions are sorted by their costs. This is a substantial improvement because, generally one finds that $|\mathcal{C}(D)| << |A_i|$ for all $i \in \mathcal{C}(D)$.

Let us make a few remarks in concluding this section. First, in the test fusion application, when the device D consists only of a single test t, then the action sets will be identical, i.e., for all channels $i \in \mathcal{C}(D)$ $A_i = A$. When the sets A_i are not given in sorted order we are able to gain an advantage in the runtime complexity compared with the straight forward application of the LMCK algorithm; in this case we only need to sort a single multiple choice set (i.e., when the entire set of actions is equal in size to a single multiple choice set). The ordering of the LMCK multiple choice set elements will be identical for all A_i where $i \in \mathcal{C}(D)$. Note that the cost coefficients of the knapsack items only differ by the scaling factor of g_i, so that we get an ordering for all of the remaining multiple choice sets for free in this case. The runtime complexity would be $O(|A| \lg |A|) = O\left(\sum_{i \in \mathcal{C}(D)} |A_i| \frac{\lg |\mathcal{C}(D)|}{|\mathcal{C}(D)|}\right)$. Finally, when we look for a "best" policy in terms of a device D and set of actions $A = \bigcup_{i \in \mathcal{C}(D)} A_i$, then we need to consider only the set $\mathrm{U}^*(\mathcal{Q}^* \cup A)$. While we may have policies $P \in \mathcal{Q}^*$ with $C(P) > 1$ (this is because we focused on proving that we get all optimal solutions to the LP, which makes sense for budget values $C(D) \leq B \leq 1 + C(D)$), those will be dominated by $\mathrm{U}^*(\mathcal{Q}^* \cup A)$. As a consequence we can speed up $TestFusion(D, \mathcal{A})$ somewhat by replacing the WHILE condition in the Main Loop by "WHILE $\gamma_k < 1$". This change will not affect our worst case time complexity, though.

6 A Dynamic Programming Algorithm

The *TestFusion* algorithm of the previous section essentially merges $|\mathcal{C}(D)|$ efficient frontiers, one for each $i \in \mathcal{C}(D)$, into a single efficient frontier of policies that use device D as the root. We are going to use now this procedure in a recursive scheme to build the frontier of policies based on larger and larger subsets of the available tests. We will apply $TestFusion(D, \mathcal{A})$ for the case when $D = t$ is a single test and when we have $A_i = A$ for all $i \in \mathcal{C}(t)$, and we denote this application as $TestFusion(t, A)$.

For a subset $T \subseteq \mathcal{T}$ of the available tests let us denote by A_T the set of un-dominated deterministic policies that use only tests from the set T and use 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 3, $T(P) = T$ for all polices for which the set T is available. This is although, in general, for $P \in A_T$ and $t \in A_T$, where $P = (D, X)$, it may be that $t \notin T_i$ for all $i \in \mathcal{C}(D)$ s.t. $b_i > 0$. I.e., the tests T are not in fact used by all policies in A_T. In other words, our domination Definition 4 pertains to the set of tests T that are available to policy P, rather than to tests that are actually used and contribute to its detection rate.

Algorithm 3 $Frontier(\mathcal{T})$

```
1: Input: A family of tests $\mathcal{T}$, and two terminal actions $I$ and $R$.
2: Initializations: $A_\emptyset \leftarrow \{I, R\}$ and $N = |\mathcal{T}|$
3: Main Loop:
4: for $k = 1, ..., N$ do
5:    for all subsets $T \subseteq \mathcal{T}$ of size $|T| = k$ do
6:       for all tests $t \in T$ do
7:          $B_{T,t} \leftarrow TestFusion\left(t, A_{T\setminus\{t\}}\right)$
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: $A_{\mathcal{T}}$.
```

Due to Claim 3 and the fact that in each main iteration for a subset of size k the input is the frontier for a subset of size $k-1$, which we have generated earlier, $Frontier(\mathcal{T})$ generates the frontier of all policies that may be obtained from the given set tests $\mathcal{T}$. For the running time of this procedure we have the following estimate.

Claim 5 *Let $M = \max_{t \in \mathcal{T}} |\mathcal{C}(t)|$, $L = \max_{T \subseteq \mathcal{T}} |A_T|$, and $N = |\mathcal{T}|$ as before. Then Frontier($\mathcal{T}$) runs in $O(2^N NM \log ML)$ time.*

Proof. $O(2^N N)$ *TestFusion* problems are needed to be solved for the 2^N possible subsets of tests. The complexity of each of the *TestFusion* applications is bounded by $O(LM \log M)$, since we maintain the action sets A_T in sorted order. By Claim 2, the complexity of *UpperHull* is bounded by $O(V)$ so that the complexity of both *UpperHull* and taking the union in step 9 are dominated by the complexity of *TestFusion* and the total runtime complexity is $O(2^N NM \log ML)$. □

It might be the case that the decision maker also needs to consider a constraint on 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 height constraint, by simply stopping when $|S|$ reaches the specified limit.

Let us finally add a some remarks about improving efficiency. Clearly, in the Main Loop in iteration k we use results for subsets of size $k-1$. Thus, in Iteration k we need only these results, that is all sets of policies A_S for $|S| < k-1$ can be deleted. This substantially reduces the memory requirement of the algorithm, though it does not change the worst case time or 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 [17]), and those can easily be extended to higher branching factors (see (1) in Section 1). We now proceed to derive a tighter upper bound on the size of the extremal frontier consisting only of the efficient deterministic policies.

Theorem 1 *Given a set of tests $\mathcal{T}$ and terminal actions $A = \{I, R\}$, there exists deterministic policies $P_0 = R$, P_1, ..., P_L such that the extremal frontier of policies obtainable from $\mathcal{T}$ and A is a piecewise linear concave functions defined by the breakpoints $(C(P_j), \Delta(P_j))$, $j = 0, 1, ..., L$, and where*

$$L \leq (|\mathcal{T}|!) \prod_{t \in \mathcal{T}} (1 + |\mathcal{C}(t)|).$$

Proof. As we observed above, we start with a deterministic policy consisting only of a single channel and no tests. Then for each *TestFusion* we create a sequence of deterministic policies. Thus, after the finite set of tests is exhausted, $A_{\mathcal{T}}$ must also consists only of deterministic policies. In each step we have only finitely many, and thus the algorithm *TestFusion* produces again finitely many, proving that $A_{\mathcal{T}}$ is finite.

To prove the cardinality claim, we observe first that $|A_\emptyset| = 2$. Thus, in this first stage, when computing fusion with single tests t, we will have at most $1 + |\mathcal{C}(t)|$ many tests (which are also policies) in $B_{\emptyset,t}$, and thus

$$|A_S| \leq \prod_{s \in S}(1 + |\mathcal{C}(s)|) \leq |S|! \prod_{s \in S}(1 + |\mathcal{C}(s)|) \tag{28}$$

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

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

follows, and thus in the merging phase we get

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

proving that (28) holds for all subsets $T \subseteq \mathcal{T}$. □

Let us remark that in practice the number of policies on the extremal frontier is much smaller than this upper bound, and thus the procedure terminates much faster than the worst case bound established in Theorem 1.

8 Computational Results

We demonstrate our computational results using randomly generated sensors and a set of sensors suggested by Saeger and Stroud as described by Boros et al. [4]. We run all computational experiments on a machine with an Intel Xeon 3.0 GHZ CPU and 6 GB of RAM.

8.1 Randomly Generated Tests

Computational results for randomly generated sensors (tests) are shown in Table 8.1, Table 8.1 and Table 8.1, for two, four and ten channel configurations, respectively. The sensor cost is selected uniformly at random from the interval $[0, 0.15]$ while the complete inspection cost is set at $c(I) = 1$. Each b_i (similarly g_i) is generated sequentially, and chosen uniformly at random from the remaining interval $[0, 1 - \sum_{i \in \mathcal{C}} b_i]$ (respectively $[0, 1 - \sum_{i \in \mathcal{C}} g_i]$). For each configuration of sensor number and channel number, 20 fold experiments are run, using different sets of randomly generated sensors.

Sensors	**Vertices**			**Runtime**		
	Max	Avg	S. Dev	Max	Avg	S. Dev
2	5	3.85	.81	.72	.055	.16
3	9	6.10	.07	.13	.06	.02
4	15	7.55	3.14	.18	.16	.01
5	28	11.30	5.67	.53	.45	.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 1: Computational results, including number of vertices of the efficient frontier and running times (in seconds), for randomly generated sensors with 2 channels.

Sensors	**Vertices**			**Runtime**		
	Max	Avg	S. Dev	Max	Avg	S. Dev
2	16	11	2.94	.02	.02	.003
3	47	27.35	10.82	.10	.08	.01
4	177	91.00	47.62	.47	.34	.07
5	364	164.30	98.79	1.88	1.21	.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	4098.6	4119.8	578.20	149.40	134.79
9	17178	9076.50	4793.70	2214.30	1139.70	621.33

Table 2: Computational results, including number of vertices of the efficient frontier and running times (in seconds), for randomly generated sensors with 4 channels.

8.2 BKFSS Gaussian Distribution Sensors

Boros, Fedzorah, Kantor Saeger and Stroud demonstrate computational results for the linear programming formulation using 4 sensors, characterized by different Gaussian distributions of "good" and "bad" cases [4]. The sensors are described in more detail in Appendix A.

Boros et al. compute the minimum cost decision tree mixture that achieves a detection rate of 81.5%. Boros et al. provide running times up to a branching factor (i.e., number of channels) of 7. With 7 channels per sensor, they find that the LP solver running time exceeds 1975 secondes. They are able to run experiments with up to 8 channels and find that the minimum cost policy with 8 channels, and overall in their experiments, has a cost of $12.06.

Sensors	Vertices			Runtime		
	Max	Avg	S. Dev	Max	Avg	S. Dev
2	67	31.3	17.48	.07	.05	.01
3	609	230.00	158.94	.93	.49	.20
4	5704	1295.70	1353.35	37.35	10.22	8.44
5	9972	3322.9	2631.29	439.66	120.97	124.89
6	36910	10172.20	7897.17	8041.53	1727.36	2021.90

Table 3: Computational results, including number of vertices of the efficient frontier and running times (in seconds), for randomly generated sensors with 10 channels.

With the same 7 channels per sensor we compute the entire efficient frontier consisting of 1747 deterministic policies (i.e., vertices) in 4.25 CPU seconds. With 8 channels per sensor we are able to compute the entire efficient frontier, consisting of 2748 vertices, in 7.91 seconds. The efficient frontier with 8 channels per sensor is shown in Figure 5.

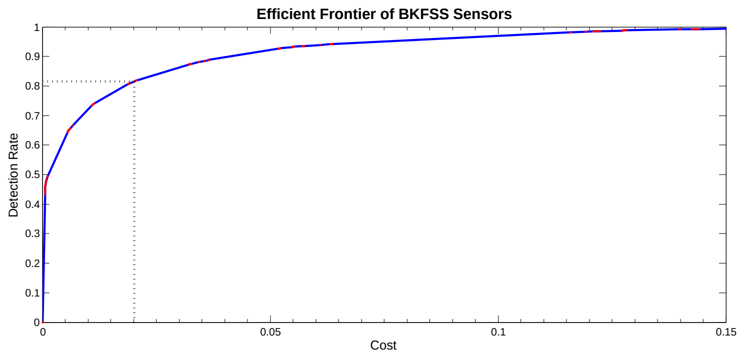


Figure 5: The entire efficient frontier computed for the BKFSS sensors using the *Frontier* dynamic programming algorithm (Algorithm 3, and the same set of seven thresholds (eight channels) used by Boros et al. [4]. The minimum cost policy achieving detection of 81.5% corresponds to the point (.02, .815) 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.)

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 channels may be used for each sensor. The resulting number of channels, number of vertices on the efficient frontier and running time as measured by CPU seconds are shown in Table 8.2. With an ϵ of .1% we find on the efficient frontier pure inspection policy with a detection rate that exceeds 81.5% and cost of \$11.87 which is lower than the least cost (mixed) policy found by Boros et al.. This inspection policy is shown in Figure 6.

ϵ	Number of channels	Vertices	Runtime
1.0%	(8,14,6,3)	1567	8.10
.9%	(8,14,6,3)	1589	8.26
.8%	(8,15,6,3)	1683	8.97
.7%	(9,16,6,3)	2004	11.11
.6%	(9,17,7,3)	2341	15.53
.5%	(10,19,7,3)	2811	22.05
.4%	(11,21,8,4)	5635	55.09
.3%	(13,24,9,4)	8710	118.10
.2%	(15,29,11,4)	13905	311.66
.1%	(21,40,15,6)	52477	3998.13

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

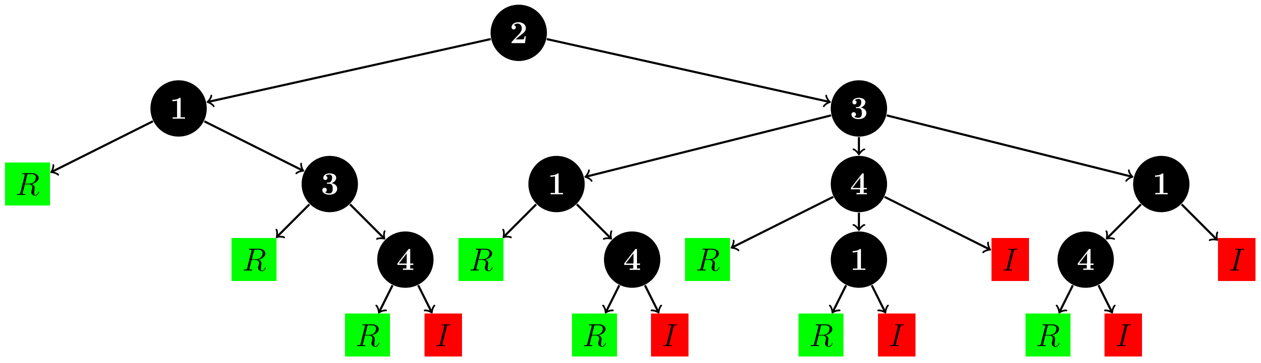


Figure 6: A pure inspection policy, using the BKFSS sensors, found using the selection of thresholds corresponding to a maximum relative error of $\epsilon = .1\%$ with respect to the continuous curve. This pure inspection policy has a cost of \$11.87 and a detection rate of 81.53%. The diagram shows a condensed representation of the decision tree; the decision rules associated with each sensor and each branch may contain conditions with respect to the previous sensor encountered along the path.

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 channels the each test (or device) labels. We have shown a monotonicity property of optimal inspection policies. We find that the monotonicity property, Lemma 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 channels of a device can be modelled 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 that solve the more general problem. When the diagnostic tests (e.g. sensors) are stochastically independent, we can use a dynamic programming algorithm, which proves to be very fast in practice for the numbers of sensors that are currently considered in the container inspection problem.

The algorithm's worst case complexity is the product of an exponential in the number of sensors, and a polynomial in the number of vertices of the efficient frontier (over all subset of sensors). We are able provide an upper bound for the number of vertices of the efficient frontier in terms of the number of channels of all tests which is tighter than previous bounds given for the total number of policies (in the case of binary decision trees) [17].

The algorithm's running time in practice is significantly faster than that of the linear programming approach [4]. This is while the algorithm provides an entire curve of efficient policies and not just the best inspection policy for a specific budget. Generating an entire efficient frontier of inspection policies supports sensitivity analysis, as well as the direct computation of any linear or nonlinear function over the efficient set. Although the number of points on the efficient frontier can grow exponentially large in the worst case, our experiments find that the size of the frontier remains manageable in practice.

References

[1] Avenhaus R., Stengel B.V., Zamir S. *Inspection games.* In R. J. Aumann and S. Hart, editors, Handbook of Game Theory, volume III. January 30 1998.

[2] Benson H.P., Lee D. *Outcome-Based Algorithm for Optimizing over the Efficient Set of Bicriteria Linear Programming Problem.* Journal of Optimization Theory and Applications, Vol. 88, No. 1, 1996, pp. 77–105.

[3] Bertsekas D.P. *Dynamic Programming and Optimal Control.* Athena Scientific, 1995.

[4] Boros E., Fedzhora L., Kantor P.B., Saeger K., Stroud P. *Large Scale LP Model For Finding Optimal Container Inspection Strategies.* Naval Research Logistics, to appear.

[5] Dantzig G. *Discrete-Variable Extremum Problems.* Operations Research, Vol. 5, No. 2, 1957, pp. 266–277.

[6] Fawcett T. *An introduction to ROC analysis.* Pattern Recognition Letters, Vol. 27 , Issue 8, 2006, pp. 861–874.

[7] Fudenberg D., Tirole J. *Game Theory.* MIT Press, 1991.

[8] Jacobson S. H., Karnani T., Kobza J. E. *Assessing the impact of deterrence on aviation checked baggage screening strategies.* International Journal of Risk Assessment and Management, Vol. 5, No. 1, 2005, pp. 1–15.

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

[10] Ibaraki T., Hasegawa T., Teranaka K., Iwase J. *The Multiple Choice Knapsack Problem.* Journal of the Operations Research Society of Japan, Vol. 21, No. 1, 1978, pp. 59–95.

[11] Kantor P., Boros E. *Deceptive detection methods for optimal security with inadequate budgets: the Screening Power Index.* RRR 26-2007, 2007. http://rutcor.rutgers.edu/pub/rrr/reports2007/26_2007.pdf.

[12] Kellerer H., Pferschy U., Pisinger D. *Knapsack Problems.* Springer, Berlin, 2004.

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

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

[15] Pisinger D. *A minimal algorithm for the Multiple-Choice Knapsack Problem.* European Journal of Operations Research, 83, 1995, pp. 394–410.

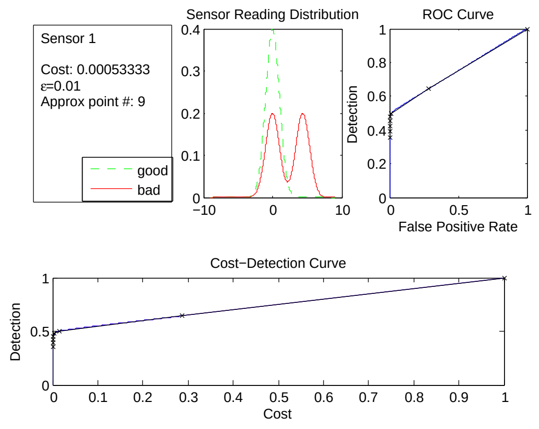
[16] Sinha P., Zoltners A. A. *The Multiple-Choice Knapsack Problem.* Operations Research, Vol. 27, No. 3, pp. 503–515.

[17] Stroud P. D. *Enumeration of increasing Boolean expressions and alternative digraph implementations for diagnostic applications.* Los Alamos National Laboratory report LAUR-03-3384, August 2003.

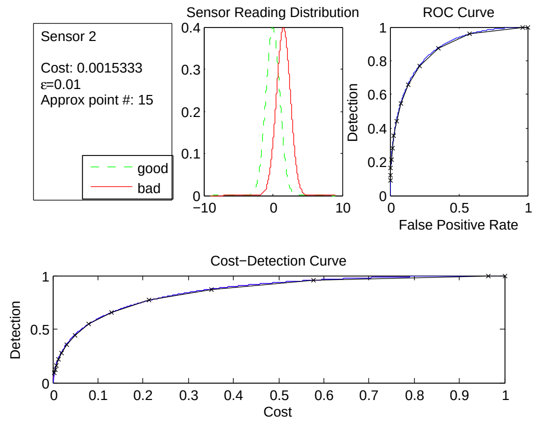
[18] Zemel E. *The Linear Multiple Choice Knapsack Problem.* Operations Research, Vol. 28, No. 6, 1980, pp. 1412–1423.

A The BFKSS Sensors

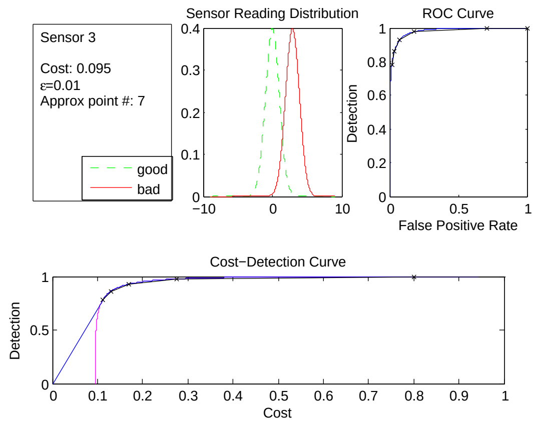
Saeger and Stroud suggest probability distributions of sensor readings for the "good" and "bad" type of shipping container populations, for four different hypothetical sensors. In Figure 7 we show the key characteristics of these sensors, which are used in the calculations reported in Section 8.2, and by Boros et al. [4]. Each sensors 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 to the *ROC Curve.*



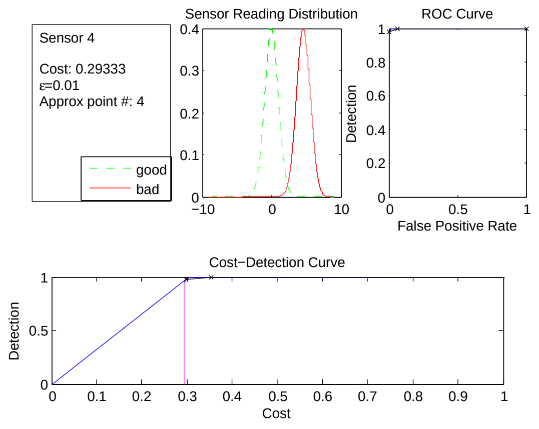
A: Sensor 1



B: Sensor 2



C: Sensor 3



D: Sensor 4

Figure 7: The 4 sensors given in [4] described by the 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 maximum relative error of 1%.

B A Practical Speed-up of the Dynamic Programming Algorithm

As mentioned in section 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 3. 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.

Claim 6 *In step 9 of Algorithm 3, 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 3.

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

For $|A_T| = |A'_T| = 1$, the claim 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 w.l.o.g. $p = (a, X)$ where a is the root test. Further assume w.l.o.g. that a is fused with policies $p_1, ..., p_l \in A'_{T\setminus\{a\}}$. Then by the inductive hypothesis $p_1, ..., p_l \in A_{T\setminus\{a\}}$. In step 7 of the algorithm, a will be considered as the root test for $TestFusion$ 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. □