Sequential Decision Making Algorithms for Port of Entry Inspection: Overcoming Computational Challenges

David Madigan, Sushil Mittal, Fred Roberts [1]
Rutgers University
Piscataway, NJ 08854 USA

***Abstract*- Following work of Stroud and Saeger [2] and Anand et al. [1], we formulate a port of entry inspection sequencing task as a problem of finding an optimal binary decision tree for an appropriate Boolean decision function. We report on new algorithms that are more efficient computationally than those presented by Stroud and Saeger and Anand et al. We achieve these efficiencies through a combination of specific numerical methods for finding optimal thresholds for sensor functions and a novel binary decision tree search algorithm that operates on a space of potentially acceptable binary decision trees.**

I. INTRODUCTION

As a stream of containers arrives at a port, a decision maker has to decide how to inspect them, which to subject to further inspection, which to allow to pass through with only minimal levels of inspection, etc. Stroud and Saeger [2] looked at this as a sequential decision making problem and formulated it in an important special case as a problem of finding an optimal binary decision tree for an appropriate binary decision function. Anand et al. [1] reported on experimental analysis of the Stroud-Saeger method that led to the conclusion that the optimal inspection strategy is remarkably insensitive to variations in the parameters needed to apply the method.

Finding algorithms for sequential diagnosis that minimize the total "cost" of the inspection procedure, including the cost of false positives and false negatives, presents serious computational challenges that stand in the way of practical implementation. To make the problem precise, we imagine a stream of containers arriving at the port with the goal of classifying each of them into one of several categories. In the simplest case, these are "ok" (0) or "suspicious" (1). There are several possible tests that can be performed and an inspection scheme specifies which test to perform next based on outcomes of previous tests. We can think of the containers as having certain attributes, such as: Does the container's ship's manifest set off an "alarm"? Is the neutron or Gamma emission count above threshold? Does a radiograph image come up positive? Does an induced fission test come up positive? We will think in the abstract of having a sensor to test for each attribute.

In the simplest case, the attributes can be described as being in one of two states, either 0 ("absent") or 1 ("present"), and we can think of a container as corresponding to a binary attribute string such as 011001. Classification then corresponds to a binary decision function F that assigns each binary string to a category. If the category must be 0 or 1, as we shall assume, F is a *Boolean decision function* (*BDF*). Stroud and Saeger consider the problem of finding an optimal *binary decision tree* (*BDT*) for calculating F. In the BDT, the interior nodes correspond to sensors and the leaf nodes correspond to categories. Two arcs exit from each sensor node, labeled left and right. By convention, the left arc corresponds to a sensor outcome of 0 and the right arc corresponds to a sensor outcome of 1. Even if the Boolean function F is fixed, the problem of finding the "optimal" BDT for it is hard (NP-complete). One can try to solve it by brute force enumeration. However, even if the number of attributes, n, is as small as 4, this is not practical. In present-day practice at busy US ports, we understand that n is of the order of 3 to 5, but this number is likely to grow as sensor technology becomes more advanced. Even under special assumptions, Stroud and Saeger were unable to produce feasible methods for finding optimal BDTs beyond the case $n = 4$. They ranked all trees formed from 3 or 4 sensors according to increasing tree costs using a measure of cost we describe in Section III. Anand et al. [1] described extensive sensitivity analysis showing that the Stroud-Saeger results were remarkably insensitive to wide-ranging changes in values of underlying parameters.

The purpose of this paper is to describe computational approaches to this problem that are more efficient than those developed to date. We describe approaches to the computation of sensor thresholds that seek to minimize the cost of inspection. We also modify the special assumptions of Stroud and Saeger to allow search through a larger number of possible BDFs, and introduce an algorithm for searching through the space of allowable BDTs that avoids searching through the Boolean decision functions entirely. We describe

[1] All three authors were supported by ONR grant number N00014-05-1-0237 and NSF grant number NSFSES 05-18543 to Rutgers University.

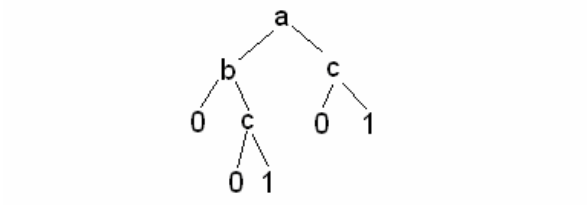

Figure 1. A binary decision tree τ with 3 sensors. The individual sensors classify good and bad containers towards left and right respectively.

experiments that parallel those of Stroud and Saeger's work.

II. Complete, Monotone Boolean Functions

The special assumptions Stroud and Saeger make in order to make computation more feasible are to limit consideration to so-called complete and monotone Boolean functions. A Boolean function F is *monotone* if, given two strings $x_1x_2\ldots x_n$, $y_1y_2\ldots y_n$ with $x_i \geq y_i$ for all i, $F(x_1x_2\ldots x_n) \geq F(y_1y_2\ldots y_n)$. F is *incomplete* if it can be calculated by finding at most n-1 attributes and knowing the value of the input string on those attributes. Stroud and Saeger enumerate all complete, monotone Boolean functions and then calculate the least expensive corresponding BDTs under assumptions about various costs associated with the trees. Their method is practical for n up to 4, but not n = 5. The problem is exacerbated by the number of BDFs. For example, for n = 4, there are 114 complete, monotone Boolean functions and 11,808 distinct corresponding BDTs. By comparison, for unrestricted Boolean functions on four variables, there exist 1,079,779,602 BDTs! For n = 5, there are 6,894 complete, monotone Boolean functions and 263,515,920 corresponding BDTs. For the unrestricted case, the number of BDTs is approximately 5 x 10^{18} [2].

III. Cost of a BDT

Following Anand et al. [1] and Stroud and Saeger [2], we assume the cost of a binary decision tree comprises two components: (i) the cost of utilization of the tree and (ii) the cost of misclassification. The cost of utilization of a tree is computed probabilistically by performing a summation over the cost of each sensor in the tree times the fraction of containers inspected by that particular sensor. We compute the cost of misclassification for a tree by adding the probabilities of false positive and false negative misclassifications by the tree and multiplying by their respective costs. Costs (i) and (ii) both depend on the distribution of the containers and the probabilities of misclassification of the individual sensors. For example, consider the decision tree τ in Fig. 1 with 3 sensors. The overall cost function to be optimized can be written as

$$\begin{aligned} f(\tau) = {} & P_0(C_a + P_{a00}C_b + P_{a00}P_{b01}C_c + P_{a01}C_c) \\ & + P_1(P_{a10}P_{b10} + P_{a10}P_{b11}P_{c10} + P_{a11}P_{c10})C_{FN} \\ & + P_1(C_a + P_{a10}C_b + P_{a10}P_{b11}C_c + P_{a11}C_c) \\ & + P_0(P_{a00}P_{b01}P_{c01} + P_{a01}P_{c01})C_{FP} \end{aligned} \quad (1)$$

Here, P_0 and P_1 are the prior probabilities of occurrence of "good" (ok or 0) and "bad" (suspicious or 1) containers, respectively (so $P_0 + P_1 = 1$). For any sensor s, P_{s00} and P_{s11} are the probabilities of correct detection of good and bad containers respectively while P_{s10}, P_{s01} are the probabilities of false negative and false positive detection respectively (so $P_{s00} + P_{s01} = 1$ and $P_{s11} + P_{s10} = 1$). These probabilities are calculated using the same complementary error function of threshold as described in detail in [1]. C_s is cost of utilization of sensor s, and C_{FN} and C_{FP} are the costs of a false negative and a false positive. (The notation here differs from that in [1].) In the above expression, the first and third terms on the right hand side together give the cost of utilization of the tree τ while the second and fourth terms represent the costs of negative and positive misclassifications.

IV. Sensor Thresholds

Sensors make errors. For sensors that produce a real-valued reading (e.g., Gamma radiation sensors), a natural approach to modeling sensor errors involves a threshold. With every sensor s, we associate a hard threshold, T_s. If the sensor reading for a container falls below T_s, then the output of that particular sensor in the tree is 0;it is 1 otherwise. The variation of sensor thresholds obviously impacts the overall cost of the tree. While sensor characteristics are a function of design and environmental conditions, the thresholds can, at least in principle, be set by the decision maker. Therefore, mathematically, a set of optimum thresholds for a given tree τ can be defined as a vector of threshold values that minimizes the overall cost function $f(\tau)$ for that tree.

We model the design and environmental conditions by assuming that sensor values for good containers follow a particular Gaussian distribution and sensor values for bad containers follow a different Gaussian distribution. This model is described in detail in [1] and [2] along with approaches to finding optimal thresholds, based on assumptions about the parameters underlying the Gaussians. In particular, [1] describes the outcomes of experiments in which individual sensor thresholds are incremented in fixed-size steps in an exhaustive search for optimal threshold values, and trees of minimum cost are identified. For example, for n = 4, [1] reported 194,481 experiments leading to lowest cost trees, with the results being quite similar to those obtained in experiments in [2]. Unfortunately, the methods do not scale and quickly become infeasible as the number of sensors (different tests available) increases.

V. Optimum Threshold Computation

One of the aims of this paper is to calculate the optimum sensor thresholds for a tree more efficiently and avoid an exhaustive search over a large number of threshold values for every sensor. The exhaustive search method suffers from a lot of drawbacks like a large search step size and limited range of search. Apart from this, the exhaustive search algorithm grows

1. start with a random vector of thresholds, $\mathbf{T_{start}}$
2. assign $\mathbf{T}$:=**inf**
3. while $|\mathbf{T}-\mathbf{T_{start}}| > .1\%$ of $\mathbf{T_{start}}$, do
4. assign $\mathbf{T}:=\mathbf{T_{start}}$
5. compute $\partial\mathbf{f}$
6. assign $\mathbf{T_{start}}:=\mathbf{T_{start}}-\lambda\,\partial\mathbf{f}$
7. end while
8. assign $\mathbf{T_{opt}}:=\mathbf{T}$

Figure 2. Pseudocode for gradient descent method

exponentially in computational time with the number of sensors, hence making it practically infeasible to go beyond a very small number of sensors. To deal with these drawbacks, we implemented various standard algorithms for nonlinear optimization problems. We note that the objective function, $f(\tau)$ is expected to be multimodal with respect to the various sensor thresholds. We used random restarts to address this concern.

A. Gradient Descent Method

In this method we form a vector of thresholds by randomly picking a threshold value for each sensor within some fixed range. Further, we find the partial differentials of the total cost function $f(\tau)$, defined in Equation (1), with respect to each sensor threshold T_s, and form their vector $\partial\mathbf{f}$, by evaluating each of those partial differentials at the threshold values selected above. Therefore

$$\partial\mathbf{f} = \left[\frac{\partial f}{\partial T_a} \quad \frac{\partial f}{\partial T_b} \quad \frac{\partial f}{\partial T_c} \cdots \frac{\partial f}{\partial T_n}\right]^{\mathrm{T}} \tag{2}$$

The threshold vector is then updated according to line 6 of the pseudocode in Fig 2. By doing this iteratively, we perform a gradient descent on the overall cost function, $f(\tau)$ towards its minimum. The pseudocode in Fig. 2, which depends on a parameter λ, summarizes this method. The method is quite effective at limiting the exponential growth of computation with increasing number of sensors. Also, it usually gives a minimum lower than the exhaustive search method due to much finer resolution in step size. For our experiments with 3 and 4 sensor trees, $\lambda=10^{-4}$ gave fairly good results with convergence achieved in a few hundred iterations.

B. Newton's Method

To eliminate the problem of setting the value of λ heuristically, we try to search for the minimum cost by using Newton's optimization method. In this method, the constant λ is replaced by the inverse of the Hessian matrix $\mathbf{H}f(\tau)$. The Hessian matrix is a square matrix of second order partial derivatives of the overall cost function $f(\tau)$. Since all the second derivatives of $f(\tau)$ are continuous over the sensor thresholds, the Hessian matrix for our problem is symmetric and is given by

1. start with a random vector of thresholds, $\mathbf{T_{start}}$
2. assign $\mathbf{T}$:=**inf**
3. while $|\mathbf{T}-\mathbf{T_{start}}| > .1\%$ of $\mathbf{T_{start}}$, do
4. assign $\mathbf{T}:=\mathbf{T_{start}}$
5. compute $\partial\mathbf{f}$
6. compute $\mathbf{H}f(\tau)$
7. if $\mathbf{H}f(\tau)$ is not positive definite
8. make $\mathbf{H}f(\tau)$ positive definite
9. end if
10. if $\mathbf{H}f(\tau)$ is well-conditioned
11. assign $\mathbf{T_{start}}:=\mathbf{T_{start}}-\left[\mathbf{H}f(\tau)\right]^{-1}\partial\mathbf{f}$
12. else
13. assign $\mathbf{T_{start}}:=\mathbf{T_{start}}-\lambda\,\partial\mathbf{f}$
14. end if
15. end while
16. assign $\mathbf{T_{opt}}:=\mathbf{T}$

Figure 3. Pseudocode for a combined method

$$\mathbf{H}f(\tau) = \begin{bmatrix} \frac{\partial^2 f}{\partial T_a^2} & \frac{\partial^2 f}{\partial T_a \partial T_b} & \cdots & \frac{\partial^2 f}{\partial T_a \partial T_n} \\ \frac{\partial^2 f}{\partial T_b \partial T_a} & \frac{\partial^2 f}{\partial T_b^2} & \cdots & \frac{\partial^2 f}{\partial T_b \partial T_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial T_n \partial T_a} & \frac{\partial^2 f}{\partial T_n \partial T_b} & \cdots & \frac{\partial^2 f}{\partial T_n^2} \end{bmatrix} \tag{3}$$

To implement this method, we just need to compute $\mathbf{H}f(\tau)$ and replace line 6 of the pseudocode in Fig. 2 with the following line:

6. assign $\mathbf{T_{start}}:=\mathbf{T_{start}}-\left[\mathbf{H}f(\tau)\right]^{-1}\partial\mathbf{f}$.

Though the computation of the Hessian matrix is a little expensive and tedious, the method quickly converges in fewer iterations than the gradient descent method. The convergence of this method depends largely on the starting vector $\mathbf{T_{start}}$. Since an absolute prior knowledge of the neighborhood of the minimum is absent, this method occasionally drifts in the wrong direction and hence fails to converge.

C. A Combined Method

Since the Hessian matrix $\mathbf{H}f(\tau)$ might not be a well-conditioned, positive definite matrix, we explored alternative approaches to computing positive definite approximations to $\mathbf{H}f(\tau)$. These methods involve modified Cholesky decomposition schemes and have been nicely summarized by Fang and O'Leary [3]. For example, a naïve way to convert a non-positive definite matrix into a positive definite matrix is to decompose it to $\mathbf{LDL}^{\mathrm{T}}$ form and then make all the non-

positive elements of **D** positive. This crude approximation may result in the failure of factorization of the new matrix or make it very different from the original matrix. Therefore to address this issue more reasonably, we use a modified $\mathbf{LDL}^{T}$ factorization method from Gill et al. [4] which incorporates small error terms in both **L** and **D** at every step of factorization. If the Hessian matrix $\mathbf{H}f(\tau)$ is ill-conditioned, we take small steps towards the minimum using the gradient descent method until it becomes well conditioned. In this way we try to combine the advantages of both gradient descent and Newton's method. The pseudocode in Fig. 3 summarizes the final scheme for finding the optimum thresholds.

VI. Searching Through A Generalized Tree Space

A. Revisiting Completeness and Monotonicity

As noted in Section II, Stroud and Saeger [2] limit their analysis to complete, monotone Boolean functions. We propose here definitions of monotonicity and completeness for trees themselves rather than limiting them to just the Boolean functions from which the trees are derived. We do this because unlike Boolean functions, binary decision trees may not necessarily consider all individual sensor outputs to give a final classification. For example, consider the decision tree of Figure 1. All the containers that follow the left-most branch of the tree do not depend upon the output of sensor **c**, since they are classified without considering **c** along with **a** and **b**. This type of example motivates the following definition of complete and monotonic trees.

Complete Decision Trees

A binary decision tree will be called *complete* if every sensor (attribute) occurs at least once in the tree and, at any non-leaf node in the tree, its left and right sub-trees are not identical.

Monotonic Decision Trees

A binary decision tree will be called *monotonic* if the final class assigned to any container (with mutually independent attributes) is the same as the independent decision based on the last attribute in the branch corresponding to the container.

Note that it is possible to arrive at a complete binary decision tree from an incomplete Boolean function and likewise a monotonic tree from a non-monotonic Boolean function. For example, consider the incomplete Boolean function for 3 sensors in Fig. 4, and the corresponding decision trees obtained from it. The Boolean function is incomplete in sensor **a.** However, trees (i) and (ii) are complete while trees (iii) and (iv) are incomplete in **a**. Similarly, consider the non-monotonic Boolean function in Fig. 5 (non-monotonic in **a**) and the decision trees obtained from it. Tree (i) is monotonic while all the other trees are non-monotonic in **a**. It is not hard to show that we get 114 complete, monotonic binary trees with 3 sensors and 66,936 with 4 sensors.

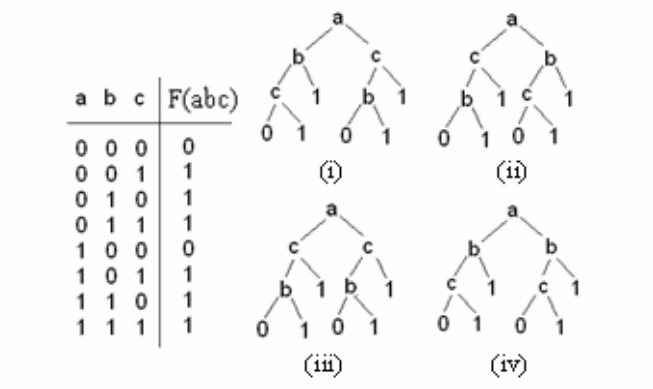

Figure 4. A Boolean function incomplete in sensor **a**, and the corresponding decision trees obtained from it

B. Tree Neighborhood and Tree Space

As shown in [2], the number of binary decision trees corresponding to complete, monotone Boolean functions increases exponentially with addition of each new sensor. Expanding the space of trees in which to search for a cost-minimizing tree to the space of complete, monotonic trees, CM tree space can be beneficial. While finding a cost-minimizing tree in CM tree space also presents a significant computational challenge as the number of sensors increases, we are able to address this challenge via heuristic search strategies that build on notions of neighborhoods in this space. Also, while CM tree space includes all the trees arising from complete, monotonic Boolean functions, it includes some trees that do not arise from complete and monotonic Boolean functions but still correspond to viable and potentially useful inspection strategies.

Chipman et al. [5] and Miglio and Soffritti [6] provide a comparison of various notions of neighborhood and proximity between trees. These methods can be classified roughly into classification-based and structure-based methods. Chipman et al. [7] describe methods to traverse the tree space. We modify these methods a little to define a notion of neighborhood that better suits our problem. Basically, we define the following four kinds of operations on a tree to get its neighboring trees. Fig. 6 gives an example of neighboring trees obtained from these operations for a particular tree.

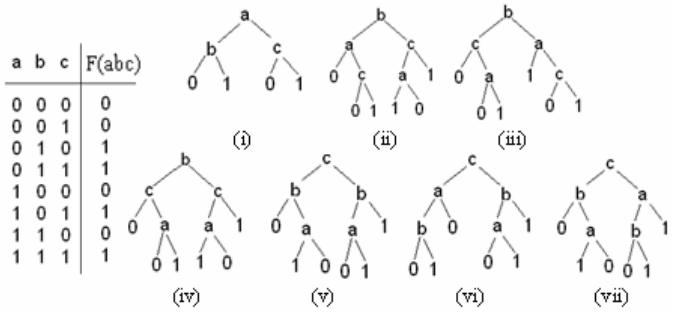

Figure 5. A Boolean function non-monotonic in sensor **a**, and the corresponding decision trees obtained from it

Split: Pick a leaf node and replace it with a sensor that is not already present in that branch, and then insert arcs from that sensor to 0 and to 1.

Swap: Pick a non-leaf node in the tree and swap it with its parent node such that the new tree is still monotonic and complete and no sensor occurs more than once in any branch.

Merge: Pick a parent node of two leaf nodes and make it a leaf node by collapsing the two leaf nodes below it, or pick a parent node with one leaf node, collapse both of them and shift the sub-tree up in the tree by one level.

Replace: Pick a node with a sensor occurring more than once in the tree and replace it with any other sensor such that no sensor occurs more than once in any branch.

It is not hard to show that these moves generate an irreducible process in the sense that one can get from any tree in CM tree space to any other tree using a sequence of the moves.

C. *Tree Space Traversal*

We have explored alternate ways to search for a tree with minimum cost in the entire CM tree space. Our initial approach was a simple greedy search: randomly start at any tree in the space, find its neighboring trees using the above operations, move to the neighbor with the lowest cost, and then iterate. As expected, however, the cost function is multimodal and the greedy strategy gets stuck at local minima. For example, there are 9 modes in the entire space of 114 trees for 3 sensors and 193 modes in the space of 66,936 trees for 4 sensors. To address the problem of getting stuck in a local minimum, we developed a stochastic search algorithm coupled with simulated annealing. The algorithm is stochastic insofar as it selects moves according to a probability distribution over neighboring trees. The simulated annealing aspect involves a so-called "temperature" t, initiated to one and lowered in discrete unequal steps after every m hops until we reach a minimum. Specifically, if we are at the *ith* tree τ_i, then the probability of going to its *kth* neighbor, denoted τ_{ik}, is given by

$$P_{ki} = \frac{\left(f(\tau_i)/f(\tau_{ik})\right)^{1/t}}{\sum_{j=1}^{n_i}\left(f(\tau_i)/f(\tau_{ij})\right)^{1/t}} \qquad (4)$$

where $f(\tau_i)$ and $f(\tau_{ij})$ are the costs of trees τ_i and τ_{ij}, respectively and n_i is the number of trees in the neighborhood of τ_i. Therefore, as the temperature is decreased, the probability of moving to the least expensive tree in the neighborhood increases. The pseudocode in Fig. 7 summarizes the stochastic search algorithm.

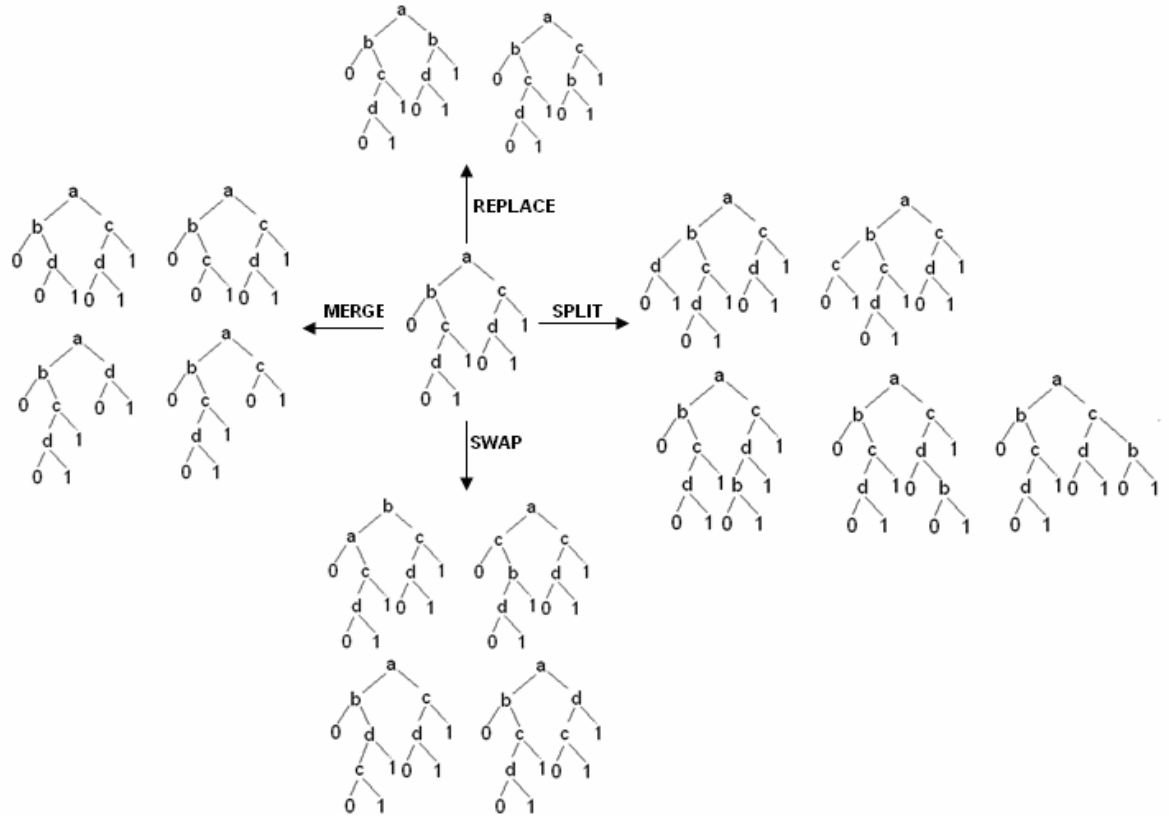

Figure 6. An example of notion of neighborhood

```
1. for number of start points do
2.    assign t:=1
3.    assign number of hops:=0
4.    assign currentTree:=random(allTrees)
5.    while a minimum is not reached
6.       evaluate c_i
7.       assign neighborTrees:=findNeighbors(currentTree)
8.       for each tree k in neighborTrees do
9.          evaluate c_ik
10.         evaluate P_ki
11.      end for
12.      assign currentTree:=random(neighborTrees, P_ki)
13.      assign number of hops:=number of hops + 1
14.      if number of hops is equal to m
15.         assign t:=t−Δt
16.         assign number of hops:=0
17.      end if
18.   end while
19. end for
```

Figure 7. Pseudocode for stochastic search method and simulated annealing for finding a minimum cost tree

VII. Results For Optimizing Thresholds

Our first set of experiments is described here. In these experiments, for any given tree, starting with some vector of sensor thresholds, we tried to reach a minimum cost in as few steps as possible. For comparison purposes, we did an exhaustive search for optimum thresholds with a fixed step size in a broad range for 3 and 4 sensors. Also, in all these experiments, the various sensor parameter values were kept the same as in the threshold variation experiments conducted in [1]. Both the misclassification costs and the prior probability of occurrence of a "bad" container were fixed as the respective averages of their minimum and maximum values suggested by Stroud and Saeger [2]. We did this for both the exhaustive search method and the optimization method described in Fig. 3, to maintain consistency throughout our experiments. With our new methods we were able reach a minimum every time with a modest number of iterations. For example, for 3 sensors, it took an average of 13.02 iterations (as opposed to 9,261 iterations using exhaustive search) to converge to a minimum for all 114 trees with $\mathbf{T}_{\mathbf{start}} = [2\ 2\ 2]^T$ as the starting point for every tree. Fig. 8 shows the plots for minimum costs for all 114 trees for 3 sensors using both the methods. In each case the minimum costs obtained using the optimization technique are equal to or less than those obtained using the exhaustive search. Also, many times the minimum obtained using the optimization method was considerably less than the one from the exhaustive search method.

VIII. Results For Searching CM Tree Space

For the second set of experiments, we utilized the notion of neighborhood around a tree using the four operations

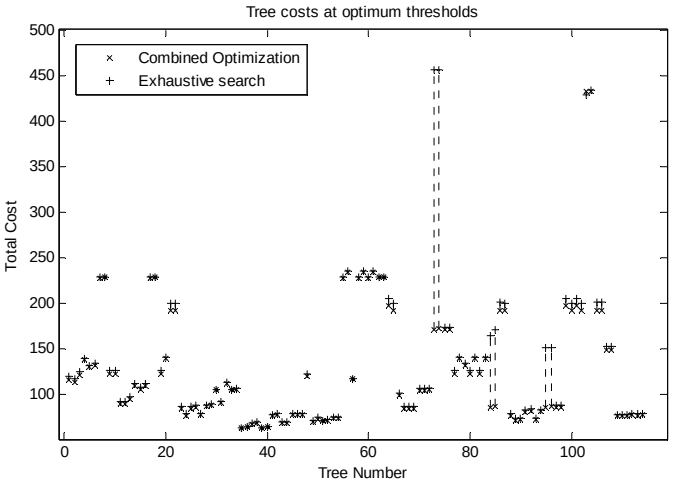

Figure 8. Minimum costs for all 114 trees for 3 sensors. To avoid confusion, dashed vertical lines join markers for the same tree.

described earlier. We randomly started 10 times with some tree in the CM tree space of 66,936 trees for 4 sensors and then kept moving stochastically in the neighborhood of the current tree, forming a chain of trees, until we reached a minimum. The exponent *1/t* was initialized to 1 and was incremented by 1 after every 10 hops in a chain. We found that the average number of trees evaluated for their costs for each chain for a set of 100 such experiments was 489. Table 1 summarizes the results of these experiments. Each row in the table corresponds to the tree number that was obtained as the least cost tree along with its cost and frequency (out of 100). The last column in the table gives the rank of each of these tree minima among all the local minima in the entire tree space. For example, the algorithm was able to find the best tree (global minimum, as determined using the methods of Section VI, part C) 42 times, second best tree 15 times and so on. Thus, the algorithm was able to find one of the least cost trees most of the time. However, these trees are different from the lowest cost trees obtained in Anand et al. [1] and are in fact less costly than those trees. Another important observation is that although each of these four trees differ in structure, they still correspond to the same Boolean function, F(abcd) = 0001010101111111, where the *i*th digit gives F(abcd) for the *i*th binary string abcd if strings are arranged in lexicographically increasing order. Also, interestingly, this Boolean function is both complete and monotonic. Detailed understanding of the nature of the differences in the trees will require understanding of the relevant properties of trees and we defer this to future work.

IX. Discussion

As we have already noted, the exhaustive search methods, both for finding the optimum thresholds for a given tree and for finding a minimum cost tree among all possible trees, become practically infeasible beyond a very small number of sensors. The various optimization techniques discussed in this

TABLE I
SUMMARY OF RESULTS FOR STOCHASTIC SEARCH FOR 4 SENSOR TREE SPACE

Tree Number [1]	Cost [2]	Frequency [3]	Mode Rank
30995	59.3364	42	1
30959	59.3364	15	2
31011	59.3364	25	3
31043	60.1924	10	4

[1] Tree numbers differ from those used in Anand et al [1].
[2] The costs of the first three trees differ only in the 14th place after the decimal, but all the trees are listed in the order of increasing costs.
[3] Frequency out of 100.

paper provide faster and better methods to limit the search space and arrive at a minimum quite efficiently. Although we were able to obtain results for 5 sensors using the stochastic search method described above, we have not included them in this paper. The reason is that the number of complete and monotonic trees obtained for 5 sensors is of the order of a few million. Since, it is computationally very hard to obtain least-cost trees using exhaustive search over all those trees, it is difficult to validate the results obtained from the stochastic search method. Also, since the notion of neighborhood that we use is structure-based, we allow only very short moves in the tree space while looking for the least cost tree. Although we tried to eliminate this problem by incorporating annealing, the results suggest that defining a better notion of neighborhood that aligns more to the sensor parameters will be a promising future direction of work. For example, if we could define "distance" between various sensors mathematically, we could use those distances to define the notion of overall distance between any two trees and hence define a neighborhood of a tree accordingly. Another possible direction of research could be the use of genetic algorithms or evolutionary techniques to build better decision trees from a given set of good trees. Examples of such methods can be found in [8] and [9]. References [10] and [11] describe applications where genetic and evolutionary algorithms successfully solved highly multi-modal problems. While our methods have led to substantially more efficient algorithms, even trees involving just 5 sensors still present a computational challenge so there is still a great deal of work to do.

ACKNOWLEDGMENTS

The authors thank Peter Meer and Oncel Tuzel for their ideas on implementing the Gradient Descent Method and Newton's Method for finding the optimum thresholds. We also thank Richard Mammone for many of the initial ideas that led to this research.

REFERENCES

[1] S. Anand, D. Madigan, R. Mammone, S. Pathak and F. Roberts, *"*Experimental Analysis of Sequential Decision Making Algorithms for Port of Entry Inspection Procedures,*"* in S. Mehrotra, D. Zeng, H. Chen, B. Thuraisingham, and F-X Wang (eds.), *Intelligence and Security Informatics, Proceedings of ISI-2006*, Lecture Notes in Computer Science #3975, Springer-Verlag, New York, 2006.

[2] P. D. Stroud, and K. J. Saeger, "Enumeration of Increasing Boolean Expressions and Alternative Digraph Implementations for Diagnostic Applications," *Proceedings Volume IV, Computer, Communication and Control Technologies*, (2003), 328-333

[3] H. Fang, and D. P. O'Leary, "Modified Cholesky Algorithms: A Catalog with New Approaches," *University of Maryland Technical Report CS-TR-4807*, 2006.

[4] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization,*Academic Press, 1981.

[5] H. A. Chipman, E. I. George and R. E. McCulloch, "Extracting Representative Tree Models From a Forest"*, working paper 98-07, Department of Statistics and Actual Science, University of Waterloo,* 1998.

[6] R. Miglio and G. Soffritti, "The Comparison between Classification Trees through Proximity Measures," *Computational Statistics and Data Analysis*, Vol. 45 (2004), pp. 577-593.

[7] H. A. Chipman, E. I. George and R. E. McCulloch, "Bayesian CART Model Search," *Journal of the American Statistical Association,* 93 (1998) 935-960.

[8] A. Papagelis and D. Kalles, "Breeding Decision Trees Using Evolutionary Techniques," *Proceedings of the Eighteenth International Conference on Machine Learning,* (2001) 393-400.

[9] Z. Bandar, H. Al-Attar and D. McLean, "Genetic Algorithm Based Multiple Decision Tree Induction", *Proceedings of the 6th International Conference on Neural Information Processing - ICONIP'99 - IEEE;* (pp 429-434); November 1999. IEEE Cat. No. 99EX378. ISBN 0-7803-5871-6.

[10] C. Im, H. Kim, H. Jung and K. Choi, "A Novel Algorithm for Multimodal Function Optimization Based on Evolution Strategy," *IEEE Transactions on Magnetics,* Vol. 40 (2004) 1224-1227.

[11] J.-P. Li, M. Balazs, G. Parks and P. Clarkson "A Species Conserving Genetic Algorithm for Multimodal Function Optimization, *Evolutionary Computation*, (2002) 10(3):207--234.