

A Stochastic Approach to Modeling Aviation Security Problems
Using the KNAPSACK Problem

Amy E. Simms

Thesis submitted to the Faculty of the Virginia Polytechnic Institute and State
University in partial fulfillment of the requirements for the degree of

Master of Science
in
Operations Research

Dr. Sheldon H. Jacobson, Co-Chair
Dr. John E. Kobza, Co-Chair
Dr. C. Patrick Koelling

June 20, 1997
Blacksburg, Virginia

Keywords: Access Control, Airport Security, Knapsack Problem, Probability Theory
Copyright 1997, Amy E. Simms

A Stochastic Approach to Modeling Aviation Security Problems
Using the KNAPSACK Problem

Amy E. Simms

(ABSTRACT)

Designers, operators, and users of multiple-device, access control security systems are challenged by the false alarm, false clear tradeoff. Given a particular access control security system, and a prespecified false clear standard, there is an optimal (minimal) false alarm rate that can be achieved. The objective of this research is to develop methods that can be used to determine this false alarm rate. Meeting this objective requires knowledge of the joint conditional probability density functions for the security device responses. Two sampling procedures, the *static grid estimation procedure* and the *dynamic grid estimation procedure*, are proposed to estimate these functions. The concept of a *system response function* is introduced and the problem of determining the optimal system response function that minimizes the false alarm rate, while meeting the false clear standard, is formulated as a decision problem and proven to be NP-complete. Two heuristic procedures, the Greedy algorithm and the Dynamic Programming algorithm, are formulated to address this problem. Computational results using simulated security data are reported. These results are compared to analytical results, obtained for a prespecified system response function form. Suggestions for future research are also included.

This research is done as part of the FAA National Center for Excellence in Aviation Operations Research at Virginia Tech. The computational results were obtained with support from the Simulation and Optimization Laboratory in the Department of Industrial and Systems Engineering at Virginia Tech.

ACKNOWLEDGEMENTS

I would like to thank Ray Easterling for his infinite patience and unending support. I would have never made it through the past two years without his help. I would also like to thank my parents, Al and Lois Simms and Jim and Jane Sthresley, for believing in me and for believing that I would graduate someday!

I thank my advisors, Dr. Sheldon Jacobson and Dr. John Kobza, for giving me the opportunity to work on this exciting project. I also thank them for the many helpful insights and suggestions that they provided. Additionally, I wish to thank Dr. Patrick Koelling for taking the time to serve on my committee. I thank Ms. Lovedia Cole for her help throughout my tenure in the Industrial and Systems Engineering Department – she is definitely the backbone of this department!

Finally, I would like to thank all of the friends that I have made in the ISE department, especially Kelly Sullivan. I will miss our long talks when we were supposed to be working! I will never forget all of the great times that I have had at Virginia Tech.

Contents

1	Introduction	1
2	Literature Review	3
2.1	Airport Security	3
2.1.1	Perimeter Security	3
2.1.2	Access Control	4
2.1.3	Detection and Estimation Theory	6
2.2	The KNAPSACK Problem	7
2.2.1	Variations of the KNAPSACK Problem	8
2.2.2	Applications of the KNAPSACK Problem	9
3	Methodology	13
3.1	Problem Statement	13
3.2	Sampling Procedures	14
3.2.1	Static Grid Estimation	14
3.2.2	Dynamic Grid Estimation	15
3.3	System Response Function	17

3.4	FALSE ALARM, FALSE CLEAR (FAFC) Problem	18
3.5	KNAPSACK Problem Heuristics	21
3.5.1	Reducing and Sorting	22
3.5.2	Greedy Algorithm	24
3.5.3	Dynamic Programming (DP) Algorithm	28
4	Analytical Results	34
4.1	Choosing the Density Functions	34
4.2	Two Continuous-Response Devices	36
4.3	Five Continuous-Response Devices	41
5	Computational Results	45
5.1	Two Continuous-Response Devices	46
5.2	Five Continuous-Response Devices	46
6	Conclusion	52
6.1	Summary	52
6.2	Future Research	54
A	C Code for Computational Evaluation	59

List of Figures

4.1	Joint Probability Density Functions	35
4.2	Piece-wise Response Function	37

List of Tables

3.1	Reduce and Sort Algorithm Example	25
3.2	Greedy Algorithm Example	27
3.3	Dynamic Programming Algorithm Example	32
4.1	Analytical Results for Two-Device System	40
4.2	Analytical Results for Five-Device System	44
5.1	Results for Two-Device System, $\varepsilon_{FC} = 10^{-7}$	47
5.2	Results for Two-Device System, $\varepsilon_{FC} = 10^{-8}$	47
5.3	Results for Two-Device System, $\varepsilon_{FC} = 10^{-9}$	48
5.4	Two-Device System Comparison, Greedy Algorithm, Grid Size = 20, $\varepsilon_{FC} = 10^{-7}$	48
5.5	Two-Device System Comparison, Greedy Algorithm, Grid Size = 20, $\varepsilon_{FC} = 10^{-8}$	48
5.6	Two-Device System Comparison, Greedy Algorithm, Grid Size = 20, $\varepsilon_{FC} = 10^{-9}$	49
5.7	Results for Five-Device System	50
5.8	Five-Device System Comparison, Greedy Algorithm, Grid Size = 5, $\varepsilon_{FC} = 10^{-7}$	50
5.9	Five-Device System Comparison, Greedy Algorithm, Grid Size = 5, $\varepsilon_{FC} = 10^{-8}$	50

5.10 Five-Device System Comparison, Greedy Algorithm, Grid Size = 5, $\varepsilon_{FC} = 10^{-9}$ 51

Chapter 1

Introduction

On December 21, 1988, Pan Am Flight 103 exploded over Lockerbie, Scotland, killing all 270 people aboard. The cause of this explosion was a terrorist bomb that passed through airport security undetected. In the wake of this and other disasters, and with terrorist threats to air transportation on the rise, aviation security has been in the spotlight [27]. The Commission on Aviation Safety and Security, established on July 25, 1996 and headed by vice-president Al Gore, recommended that the aviation industry utilize existing security technology, such as explosive detection, automated passenger profiling, and passenger-bag matching. The FAA administrator recently remarked that the FAA is in "...hot pursuit of equipment and procedures that can spot these [explosive] devices with high degrees of confidence for the nearly one billion pieces of luggage and 500 million passengers traveling annually on U.S. carriers" [22]. The primary goal of these efforts is to improve security at airports throughout the United States.

In order to meet this goal, the FAA must develop security systems that will optimally utilize and implement many different types of security technologies and procedures. Many existing security devices return a continuous response. For example, a passenger profile may produce a number between 0 and 100, indicating the level of threat posed by the passenger, or a chemical scan may yield a number between 0 and 10, indicating the amount of a particular chemical detected on the baggage. A security system must combine these responses into an appropriate overall security response, which should correctly classify the level of threat posed by the passenger or the baggage.

One important aspect of designing an access control security system is managing the trade-off between the two types of errors that can occur. A security system can allow a *false clear* (i.e., allow a threat item to pass through undetected), or the system can allow a

false alarm (i.e., not allow a non-threat item to gain access). Because these errors are dependent on one another, it is impossible to minimize both simultaneously. In addition, both types of errors have a significant impact on the aviation industry. High false clear rates obviously must be avoided, as a false clear can result in injury, destruction, and death. High false alarm rates also have an adverse, though more subtle, affect. False alarms may cause passengers to be unnecessarily delayed, resulting in missed flights that can cause a ripple effect, with significant human and economic implications. An optimal security system should find some middle ground between the two errors. For example, a security system could strive to minimize the false alarm rate, given that the false clear rate is held to a prespecified (and very small) bound. The purpose of this thesis is to study the false alarm, false clear tradeoff and provide tools to design effective security systems.

The thesis is organized as follows. In Chapter 2, literature reviews on airport security, as well as uses of the knapsack problem are presented. Chapter 3 presents the methodology used to develop this research. Two sampling procedures that can be used to estimate the joint probability density function for multiple-device security system responses are introduced. The concept of a *system response function* is also defined, and the problem of determining the optimal system response function that minimizes the false alarm rate while meeting a prespecified false clear standard is formulated as a decision problem and proven to be NP-complete. Finally, two heuristic procedures, the Greedy algorithm and the Dynamic Programming (DP) algorithm, are formulated to address this problem. Chapter 4 presents an analytical method for finding the optimal probabilities of false clears and false alarms for two particular security systems. Chapter 5 presents computational results, using the Greedy and DP algorithms for the two systems with simulated security data. Finally, Chapter 6 summarizes the work and presents suggestions for future research.

Chapter 2

Literature Review

2.1 Airport Security

Security systems can be broadly categorized as either perimeter security or access control. Perimeter security is concerned with protecting an entire facility from external intruders or internal escapees. To ensure the integrity of the system, threatening events must be detected and responded to quickly and efficiently. Access control systems, on the other hand, are primarily concerned with *gate keeping*. In an airport access control system, for instance, luggage and passengers are screened to prevent the introduction of firearms or explosives onto aircraft. The control procedures occur at a specific checkpoint, such as the entrance to the boarding area, where a decision is made as to whether the person or object will be allowed access.

2.1.1 Perimeter Security

The literature related to perimeter security contains numerous probabilistic approaches. Several authors use a probability network to model an attack scenario, where arcs represent specific tasks or actions of either the intruders or the defenders, and nodes represent possible situations. The tasks and actions represented by the arcs have general completion-time distributions, and the arcs leaving a node have a probability associated with them. The probability of intruder success or failure is determined, as well as the probability density function for the time until the intruder reaches his goal or is captured. Doyon [9] uses such a network to determine measures of effectiveness of a perimeter security system, where the

intrusion route is divided into zones, and intrusion into a zone activates an alarm at a guard station. An attack scenario is modeled analytically to facilitate system evaluation. Davis *et al.* [7] also use a probability network to determine a measure of effectiveness for a system consisting of an intruder detection system, a physical security barrier system, the response force procedures, the interconnecting communication links, and the management of these resources. The measure of effectiveness is defined as the probability that a security system will counter a threat, given analytical models of both intruder and defender scenarios. Peyron *et al.* [26] develop a simulation program based on a probability network. The Advanced Integration Protection System (AIPS) is designed to model a perimeter security system, consisting of several levels of electronic barriers or sensors, interfaces between each of these levels, and a command and control center. Site protection can be optimized through sensitivity analysis of this model. The AIPS model is also used to quantify the likelihood of intrusion and the cost of defense for each zone on the site. The probability network approach can also be applied to the problem of prison security; Tarr [33] uses such a network to calculate the probability of capture for a prison security system consisting of alarms, barriers, and response forces.

The models developed by Martz and Johnson [24] and Dessent [8] also use the network approach, but assume that all task times are normally distributed. Martz and Johnson use their model to allocate resources in either a fixed facility or a moving convoy. Sensitive areas are ranked according to survivability of a hypothesized attack, where the probability of a successful attack is computed. Dessent performs an analysis of perimeter security for prisons, studying the cost-effectiveness of various security measures using an attack scenario that allows different attack styles and weather conditions.

Rather than considering multiple tasks, the model of Hunt and Vanderslice [17] considers only a single task, the response of a guard team. The design variable is the number of guard teams necessary to implement effective perimeter security. This work is particularly significant, since it introduces the concept of false alarms and their effect on the security system.

2.1.2 Access Control

The literature related to access control contains only a few probabilistic approaches. Gilliam [12] uses queueing theory to examine the problem of screening passengers to allow access to a controlled area, such as a boarding area or an aircraft. Screening consists of two operations, inspecting the passenger and inspecting any carry-on luggage. Although Gilliam recognizes that the total time to screen a passenger is a function of the false alarm rate as well as the throughput rate of the security device, no suggestions are made towards reducing the

false alarm rate. Instead, the false alarm rate is incorporated into the screening rate and steady-state analysis is performed on simulated passenger data. The resulting simulation model is used to analyze the performance of existing access control systems in U.S. airports.

A personnel access control system is studied by Bradley [2]. Each attempt to gain access to the system is categorized as a success or a failure. In addition, an access attempt is performed by either an enrolled individual or an unauthorized individual, who can use the system in a normal manner, can claim to be someone else, or can attempt to tamper with the system. The subset of successful attempts are partitioned into mutually exclusive subsets from which the overall probability of a successful attempt is calculated. From this calculation, two types of terms are identified: conditional probabilities and joint probabilities. The conditional probabilities represent the response, or sensitivity, of the system to a specific user population. The joint probabilities represent the user population and the threat to the facility. In other words, the joint probabilities measure the likelihood that a specific access attempt will occur regardless of outcome. A similar situation occurs in quality control, when products are checked to ensure that defective items are not passed on to customers. An analysis of a system containing multiple tests was performed by Christer [5].

Analyzing an access control system based on the probability of error is a recent technique. Kobza and Jacobson [20] develop probability models to analyze and design access security system architectures. A model of security devices is constructed, where the devices are characterized in terms of the types of errors they are capable of making. Given that each device returns an alarm or a clear, two types of errors are possible. These errors are classified as Type I, where a non-threat item elicits an alarm, and Type II, where a threat item elicits a clear. Two important events are also defined. A false clear occurs when there is no alarm and a threat item is inspected, and a false alarm occurs when there is an alarm and a non-threat item is inspected. The trade-off between the probability of a false clear and the probability of a false alarm is examined for a single-device system, a cascading two-device system, and both cascading and parallel multiple-device systems. In this work, all devices are considered to be independent.

Kobza and Jacobson [19] present a method for evaluating an access control system when devices are dependent. Dependency occurs when knowing the decision at one device will increase or decrease the probability of an alarm at another device in the system. The effect of device dependence on the Type I and Type II errors and on the probabilities of false alarm and false clear are evaluated for a cascading two-device system and a cascading multiple-device system. In addition, desirable dependency relationships are defined for a two-device system.

2.1.3 Detection and Estimation Theory

The methods used in this document to solve the access control problem relate closely to classical detection and estimation theory [35]. A simple decision theory problem has three basic components. The first is a *source* that generates an output hypothesis that could consist of two or more choices. The second component is a *probabilistic transition mechanism*, while the third is an *observation space*. The transition mechanism is a device that determines whether or not the hypothesis is true, and generates a point in the observation space based on this knowledge.

The access control problem with two devices is related to the binary hypothesis testing problem. In the binary hypothesis testing problem, each of two source outputs corresponds to one of two hypotheses, H_0 and H_1 , and each hypothesis maps into a point in the observation space. The observation space is assumed to correspond to a set of N observations: r_1, r_2, \dots, r_N , which are denoted by the vector \underline{r} . The probabilistic transition mechanism generates points according to two known conditional probability densities, $p_{\underline{r}|H_0}(R | H_0)$ and $p_{\underline{r}|H_1}(R | H_1)$, where R is a random variable denoting the observation. Given the hypotheses H_0 and H_1 , there are four possible outcomes. These outcomes are:

1. H_0 true; choose H_0 .
2. H_0 true; choose H_1 .
3. H_1 true; choose H_1 .
4. H_1 true; choose H_0 .

The goal is to develop a suitable rule for deciding which output hypothesis to select. Two methods are developed, *Bayes Criterion* and *Neyman-Pearson Tests*. The Bayes Criterion method assumes that the two source outputs are governed by apriori probabilities, P_0 and P_1 , and that a cost is assigned to each of the four outcomes. These costs are denoted by C_{00} , C_{01} , C_{10} , and C_{11} , where the first subscript indicates the hypothesis chosen and the second indicates the true hypothesis. The decision rule is designed so as to minimize the average cost. To implement this method, the total observation space, Z , is divided into two parts, Z_0 and Z_1 . Observations that fall into Z_0 elicit the H_0 hypothesis, while observations that fall into Z_1 elicit the H_1 hypothesis. An expression for the risk, where risk is defined as the average cost of the system, is written in terms of these decision regions. Finally, the decision regions, Z_0 and Z_1 , are chosen so as to minimize the risk.

Bayes criterion leads to a *likelihood ratio test*, where the likelihood ratio is defined as

$$\frac{p_{\mathbf{r}|H_1}(R | H_1)}{p_{\mathbf{r}|H_0}(R | H_0)}.$$

This value is a random variable, and is tested against the threshold

$$\nu = \frac{P_0(C_{10} - C_{00})}{P_1(C_{01} - C_{11})}.$$

If the likelihood ratio is greater than ν , the output is H_1 ; otherwise the output is H_0 .

In many cases it may be difficult to determine costs or the apriori probabilities. The Neyman-Pearson test bypasses these factors by introducing the conditional probabilities

$$P_F = \int_{Z_1} p_{\mathbf{r}|H_0}(R | H_0),$$

which represents the likelihood of selecting the hypothesis H_1 when H_0 is true, and

$$P_D = \int_{Z_0} p_{\mathbf{r}|H_1}(R | H_1),$$

which represents the likelihood of selecting the hypothesis H_0 when H_1 is true. The objective of the Neyman-Pearson test is to make P_F as small as possible and P_D as large as possible. To accomplish this objective, P_F is constrained by a lower bound, α , and P_D is maximized using Lagrange multipliers. This objective is similar to the objective of the access control problem, which is solved in this thesis. In the access control problem, however, the observation space, Z , is discretized and then KNAPSACK heuristics are applied. This allows us to solve an access control problem even if the conditional probability density functions are unknown. The Neyman-Pearson test, on the other hand, requires that these density functions be known.

2.2 The KNAPSACK Problem

To describe the KNAPSACK problem, consider a knapsack with capacity c and a set of n objects, where object j has utility, p_j , and size, w_j . Define a vector of binary variables, \mathbf{x} , such that

$$x_j = \begin{cases} 1, & \text{if object } j \text{ is placed in the knapsack;} \\ 0, & \text{otherwise,} \end{cases}$$

for $j = 0, \dots, n - 1$. The **(binary) KNAPSACK problem** asks how a knapsack can be filled, such that the value of its contents is maximized and its capacity is not exceeded. The KNAPSACK problem can be written as an integer program

$$\begin{aligned} & \text{maximize} && \sum_{j=0}^{n-1} p_j x_j \\ & \text{subject to} && \sum_{j=0}^{n-1} w_j x_j \leq c \\ & && x_j = 0 \text{ or } 1, \quad j = 0, \dots, n - 1. \end{aligned}$$

The KNAPSACK problem is NP-Hard [23]. The only way to guarantee an optimal solution is to enumerate all 2^n feasible solutions; as n increases, the number of feasible solutions increases exponentially. In other words, for a computer capable of examining one billion solution vectors, \underline{x} , per second, solving the KNAPSACK problem would take 30 years for $n = 60$, more than 60 years for $n = 61$, and ten centuries for $n = 65$.

2.2.1 Variations of the KNAPSACK Problem

Several popular variations of the KNAPSACK problem include (Martello & Toth, 1990)

- **MULTIPLE CHOICE KNAPSACK Problem:** Partition the object set into subsets and pick at most one object per subset.
- **BOUNDED KNAPSACK Problem:** For each j , there are b_j objects of value p_j and size w_j available, so x_j must be a nonnegative integer less than or equal b_j .
- **SUBSET-SUM Problem:** A binary KNAPSACK problem where $p_j = w_j$ for all j .
- **MUTLIPLKE KNAPSACK Problem:** A KNAPSACK problem with m knapsacks of given capacities $c_j, j = 1, \dots, m$.
- **STOCHASTIC KNAPSACK Problem:** A KNAPSACK problem with random arrivals and departures from the knapsack. The KNAPSACK problem with stochastic rewards and the goal of maximizing the probability of reaching a certain total reward is also referred to as the STOCHASTIC KNAPSACK problem.

2.2.2 Applications of the KNAPSACK Problem

Cryptography

Cryptography, or data security, is the science of encrypting messages. A public-key cryptosystem uses two keys to encrypt a message \underline{x} , one public and one private. Anything that is encrypted with one key can be decrypted with the other. The goal of cryptography is to make it infeasible (i.e., require an unreasonable amount of time and effort) to break an encryption. Given one member of the pair, the public key, it should be intractable to discover the other, the private key. Because KNAPSACK problems are NP-Complete, they lend themselves to the encryption problem.

The most prevalent use of the KNAPSACK problem in cryptography is the TRAP-DOOR KNAPSACK problem, developed by Merkle and Hellman [25]. The Trap-Door Knapsack encryption code is based on the SUBSET-SUM problem, where the utility of each object is equal to its size. In other words, the objective is to fit as much as possible into the knapsack, with a minimum amount of knapsack capacity left over. Consider the case where the objects form a *superincreasing vector*. A superincreasing vector is defined as a vector of objects with sizes $(w_0, w_2, \dots, w_{n-1})$ where

$$w_i \geq \sum_{j=0}^{i-1} w_j, \quad i = 1, \dots, n-1.$$

To maximize the value in the knapsack, objects are selected by descending size order. Each object that does not exceed the capacity is placed in the knapsack.

Merkle's algorithm is as follows:

- Generate a random superincreasing vector \underline{a}' of dimension n , such that

$$a'_i \geq \sum_{j=0}^{i-1} a'_j, \quad i = 1, \dots, n-1.$$

- Generate a random integer m , where

$$m \geq \sum_{j=0}^{n-1} a'_j.$$

- Generate a random integer w , where w is relatively prime to m .

- The information w , \underline{a}' , and m are considered private. Set the public key, \underline{a} , such that $a_i = w (a'_i \bmod(m))$, $i = 0, \dots, n - 1$.
- To send message \underline{x} in binary format, compute and send $S = \underline{a} \cdot \underline{x}$.
- To decode message \underline{x} , compute

$$\begin{aligned}
 S' &= (w^{-1} S) \bmod(m) \\
 &= (w^{-1} \sum_{i=0}^{n-1} x_i a_i) \bmod(m) \\
 &= (w^{-1} \sum_{i=0}^{n-1} x_i w a'_i) \bmod(m) \\
 &= (\sum_{i=0}^{n-1} x_i a'_i) \bmod(m).
 \end{aligned}$$

Since $m \geq \sum_{j=0}^{n-1} a'_j$, then

$$\begin{aligned}
 S' &= \sum_{i=0}^{n-1} x_i a'_i \\
 &= \underline{a}' \cdot \underline{x}.
 \end{aligned}$$

- Solve the subset-sum KNAPSACK problem with size vector \underline{a}' and capacity S' to yield the binary message \underline{x} .

The private key (w, \underline{a}', m) was determined in polynomial time by Shamir [31], rendering this method no longer useful for encryption.

Chor and Rivest [4] create a new type of knapsack-encrypted public key cryptosystem. They use a method for unique representation of sums in dense finite sequences to make the KNAPSACK problem denser, where a dense knapsack is defined as one that has many objects. Another use of Merkle's system is found in access control in multi-user networks. An access matrix gives customer i 's privilege level (read, execute, write, own) to file j . Jan and Wang [18] use Merkle's knapsack algorithm, along with other security measures, to encode user privileges in a network.

Broadband Communication

Ross and Tsang [29] introduce the STOCHASTIC KNAPSACK problem, where objects arrive to and depart from the knapsack at random times. No assumption is made of the

sojourn distribution, for which class dependency is permitted. Interarrival times for any given class are assumed to be exponentially distributed with mean depending on the current number of objects of that class in the knapsack. The problem is to accept or block arriving objects as a function of the current system state (the current number of objects of each class in the knapsack) so as to maximize the long-run average revenue. This is motivated by the problem of accepting and blocking calls to a circuit-switched telecommunication system which supports a variety of traffic classes (voice, video, fax), where each class has different bandwidth requirements and holding-time distributions.

Ross and Tsang suggest that a DP algorithm be used to solve this problem. They prove for two call classes that a threshold policy is optimal, where one class is always accepted and the other is accepted up to a threshold. Gavious and Rosberg [11] attack the problem of dynamically allocated capacity of each circuit among a variety of traffic classes. They use Knapsack heuristics to bound the optimal expected reward from above, then propose a threshold policy that guarantees suboptimal rewards. Chiu *et al.* [3] introduce a new class of threshold policies and use stochastic optimization to determine policies that optimizes long-run average revenue for any number of traffic classes.

Cutting Stock

One of the oldest applications of the KNAPSACK problem occurs in cutting stock problems. The goal is to determine the cutting policy that minimizes the amount of excess trim. Gilmore and Gomory [13] propose using the KNAPSACK problem to select a small set of cutting patterns that yield a minimal amount of excess trim. Their methods appear in several cutting algorithms used in the glass and wood industries [15] [21] [30].

Resource Allocation

Problems where a large number of resources must be allocated over a large area can be modeled as KNAPSACK problems. Examples of specific areas of application of these problems are

- Acquisitions budgeting in a public library [14]:
A computerized Decision Support System (DSS) was set up for the Monroe County (Indiana) Public Library. The Budget Allocation module of the DSS produces a satisfactory allocation of funds by solving a MULTIPLE KNAPSACK problem. The goal is to maximize the expected use of materials with respect to the number of

objects purchased in each budget category, subject to the budget constraint and other bounds (e.g., past budgeting, patron interest, space). The librarian solves the MULTIPLE KNAPSACK problem, where each budget category is represented by a knapsack. The capacities of the knapsacks (i.e., the budget constraints for each budget category) are varied until a satisfactory budget is reached. When this method was used by the Monroe County Public Library, the budget was generated efficiently and was deemed to be superior to previous budgets.

- Flight and Fire Control [28]:
Assigning aircraft to targets during battle is modeled as a MULTIPLE KNAPSACK problem. A mission must be assigned to each plane such that the expected utility is maximized, and the assignment must be made in real time. All known information is taken into account, including location, velocity, fuel level, amount and type of ammunition. This problem is solved using temporal trees.
- Zone Hopping [1]:
A set of n packages must be moved by a common carrier (i.e., an airplane) from location A to a final destination located in billing zone B . Suppose that there exists an intermediate location, D , that is closer to the final destination and is located in billing zone B . If some of the packages are moved by vehicle from A to D , a cost savings may result. The binary KNAPSACK problem is used to determine which packages should be moved to D , in order to maximize the cost savings while not exceeding the capacity of the vehicle.
- Risk Criteria [16]:
This problem considers the allocation of investment funds, where the rewards are stochastic and the goal is to maximize the probability of attaining the target total reward. The deterministic version of this problem is a binary KNAPSACK problem. The stochastic version is modeled as a KNAPSACK problem, where each object has stochastic value. This KNAPSACK problem is solved using a DP algorithm.

Chapter 3

Methodology

3.1 Problem Statement

Consider a d -device access control security system, where each device provides a security response that can be represented by the random variable, R_i , $i = 1, \dots, d$. Devices can include security technology equipment, or security procedures that elicit security information from an item, where an item can be a passenger, baggage, or anything that is considered for entry into the system. Without loss of generality, assume that the device responses can be rescaled such that $0 \leq R_i \leq 1$, where a response close to one suggests a threat item and a response close to zero suggests a non-threat item.

Two types of errors can occur in an access control security system. The system can allow a *false clear* (i.e., allow a threat item to pass through undetected), or the system can allow a *false alarm* (i.e., not allow a non-threat item to gain access). Additionally, the system can return the correct response, in the form of a *true clear* (i.e., correctly determining that an item does not pose a threat) or a *true alarm* (i.e., correctly detecting a threat item). In the aviation industry, the FAA requires that all airport access control systems meet a prespecified false clear standard. Given this standard, it is possible for false alarms to become excessively high. This is a concern for the aviation industry, since false alarms cause unnecessary delays that may result in missed flights and passenger dissatisfaction. The research question addressed in this thesis is:

Given a particular access control security system and a prespecified false clear standard, what is the minimum false alarm rate that can be achieved?

This question will be investigated using tools from probability theory, Monte Carlo simulation, and mathematical programming.

3.2 Sampling Procedures

To address the false alarm, false clear tradeoff research question, the joint conditional probability density functions for the d -device security system responses, given a threat or a non-threat item, must be estimated. Define $F_{\underline{R}|T}(\underline{r}) = P\{R_1 \leq r_1, R_2 \leq r_2, \dots, R_d \leq r_d\}$ to be the joint probability distribution function of device responses, conditional on a threat item being passed through the d devices, where $\underline{R} = (R_1, R_2, \dots, R_d)$ is a vector of random variables representing the response from each security device. Also define $F_{\underline{R}|NT}(\underline{r}) = P\{R_1 \leq r_1, R_2 \leq r_2, \dots, R_d \leq r_d\}$ to be the joint probability distribution function of device responses, conditional on a non-threat item being passed through the d devices. The conditional joint probability density functions are defined as

$$f_{\underline{R}|T}(\underline{r}) = \frac{dF_{\underline{R}|T}(\underline{r})}{d\underline{r}}$$

and

$$f_{\underline{R}|NT}(\underline{r}) = \frac{dF_{\underline{R}|NT}(\underline{r})}{d\underline{r}}.$$

These conditional joint probability density functions can be estimated by sampling over various threat or non-threat items.

Two sampling procedures are used to estimate these density functions. The first procedure, called the *static grid estimation procedure*, uses an evenly spaced grid structure to create a d -dimensional histogram. The second procedure, called the *dynamic grid estimation procedure*, dynamically modifies the grid structure as additional sample points are collected, based on where the sample points cluster in the $(0, 1]^d$ hypercube.

3.2.1 Static Grid Estimation

The following notation is needed to develop the static grid estimation procedure. Classify the response from device j into one of n_j evenly spaced cells, each of length $1/n_j$, $j = 1, \dots, d$. Therefore, $(0, 1]^d$ is decomposed into $\prod_{j=1}^d n_j$ d -dimensional hypercubes. Define the notation (i_1, i_2, \dots, i_d) to denote the hypercube

$$\left(\frac{i_1}{n_1}, \frac{i_1 + 1}{n_1}\right] \times \left(\frac{i_2}{n_2}, \frac{i_2 + 1}{n_2}\right] \times \dots \times \left(\frac{i_d}{n_d}, \frac{i_d + 1}{n_d}\right], \quad i_j = 0, 1, \dots, n_j - 1, \quad j = 1, \dots, d.$$

Also define $h(i_1, i_2, \dots, i_d)$ to be the total number of sample points that fall into cube (i_1, i_2, \dots, i_d) and M to be the total number of sample points collected.

The algorithm for the static grid estimation procedure is as follows:

Initialization: Set the values $n_j, j = 1, 2, \dots, d$.

Set $h(i_1, i_2, \dots, i_d) = 0, i_j = 0, 1, \dots, n_j - 1, j = 1, 2, \dots, d$.

Set $m = 0$ to be the total number of sample points collected so far.

Sampling: While $m \leq M$ perform the following steps:

Independently sample a value for \underline{r} , given a threat item.

Determine the hypercube in which the \underline{r} value falls and increment (by one) the $h(\cdot)$ value associated with this hypercube.

Set $m = m + 1$.

Computation: $h(i_1, i_2, \dots, i_d)/M$ is an estimator for

$$\int_{i_1/n_1}^{(i_1+1)/n_1} \dots \int_{i_d/n_d}^{(i_d+1)/n_d} f_{\underline{R}|T}(\underline{r}) dr_1 \dots dr_d, \\ i_j = 0, 1, \dots, n_j - 1, \quad j = 1, 2, \dots, d.$$

This procedure is repeated for non-threat items to estimate $f_{\underline{R}|NT}(\underline{r})$.

The static grid estimation procedure is sensitive to how the grid points are set. In particular, the values for the n_j affect the quality of the estimators for $f_{\underline{R}|T}(\underline{r})$ and $f_{\underline{R}|NT}(\underline{r})$. If the n_j are set too small, the grid will be so fine that very few sample points will be observed in each hypercube. On the other hand, if the n_j are set too large, the grid will be too coarse to capture the form for $f_{\underline{R}|T}(\underline{r})$ or $f_{\underline{R}|NT}(\underline{r})$. These tradeoffs exist anytime a static grid is used to estimate a probability density function.

3.2.2 Dynamic Grid Estimation

The dynamic grid estimation procedure is introduced to address the problem of choosing the n_j values. This procedure starts with the $(0, 1]^d$ hypercube and samples points, as described in the static grid estimation procedure. If the number of sample points within a hypercube reaches some predetermined threshold value, L , this hypercube is decomposed into 2^d new hypercubes. To describe the dynamic grid estimation procedure, define the notation $(n, i_1, i_2, \dots, i_d), n \geq 0$, to denote the hypercube

$$\left(\frac{i_1}{2^n}, \frac{i_1+1}{2^n}\right] \times \left(\frac{i_2}{2^n}, \frac{i_2+1}{2^n}\right] \times \dots \times \left(\frac{i_d}{2^n}, \frac{i_d+1}{2^n}\right], \quad i_j = 0, 1, \dots, 2^n - 1, \quad j = 1, \dots, d.$$

Therefore, $(0, 0, \dots, 0)$ represents the $(0, 1]^d$ hypercube. If hypercube $(n, i_1, i_2, \dots, i_d)$ is decomposed, the resulting 2^d new hypercubes are

$$(n+1, 2i_1 + I(1), 2i_2 + I(2), \dots, 2i_d + I(d)), \quad I(i) = 0, 1, \quad i = 1, 2, \dots, d.$$

Also define $h(n, i_1, i_2, \dots, i_d)$ to be the total number of sample points that fall into hypercube $(n, i_1, i_2, \dots, i_d)$, and M to be the total number of sample points collected.

The algorithm for the dynamic grid estimation procedure is as follows:

Initialization: Set $h(0, 0, \dots, 0) = 0$.

Set L to be the sampling threshold number.

Set $m = 0$ to be the total number of sample points collected so far.

Sampling: While $m \leq M$ perform the following steps:

Independently sample a value for \underline{r} , given a threat item.

Determine the hypercube in which the \underline{r} value falls and increment (by one) the $h(n, i_1, i_2, \dots, i_d)$ value associated with this hypercube.

If for any hypercube, $h(n, i_1, i_2, \dots, i_d) = L$, decompose this hypercube into the 2^d hypercubes

$$(n+1, 2i_1 + I(1), 2i_2 + I(2), \dots, 2i_d + I(d)), \\ I(i) = 0, 1, \quad i = 1, 2, \dots, d.$$

Compute the values for $h(n+1, 2i_1 + I(1), 2i_2 + I(2), \dots, 2i_d + I(d))$, $I(i) = 0, 1, \quad i = 1, 2, \dots, d$, such that the L sample points in hypercube $(n, i_1, i_2, \dots, i_d)$ are now reclassified into these 2^d hypercubes.

Set $m = m + 1$.

Computation: $h(n, i_1, i_2, \dots, i_d)/M$ is an estimator for

$$\int_{i_1/2^n}^{(i_1+1)/2^n} \dots \int_{i_d/2^n}^{(i_d+1)/2^n} f_{\underline{R}|T}(\underline{r}) dr_1 \dots dr_d, \\ i_j = 0, 1, \dots, n_j - 1, \quad j = 1, 2, \dots, d.$$

This procedure is repeated for non-threat items to estimate $f_{\underline{R}|NT}(\underline{r})$.

When a dynamic grid is used, the choice for L determines the quality of the estimators for $f_{\underline{R}|T}(\underline{r})$ and $f_{\underline{R}|NT}(\underline{r})$. In general, L should be set large enough to ensure that each

hypercube contains a reasonable number of sample points. Trial and error, as well as experience with the security system devices, will determine how to best set the value of L . The dynamic grid estimation procedure is designed to provide a fine grid in the parts of the $(0, 1]^m$ hypercube where the greatest number of sample points are collected, while leaving the grid coarse in regions where few sample points are collected. In order to implement the dynamic grid estimation procedure, each sample point must be stored in memory, resulting in an additional memory cost that is not associated with the static grid procedure. However, the availability and low cost of computer memory makes such a burden reasonable to manage.

3.3 System Response Function

The *system response function* combines the responses of all devices in the access control security system and returns an overall response, either an alarm or a clear. Define the system response function, $g : (0, 1]^d \rightarrow [0, 1]$, where a *system alarm* occurs if $g(\underline{x}) \geq \gamma$ and a *system clear* occurs if $g(\underline{x}) < \gamma$, where $0 \leq \gamma \leq 1$. The system response function partitions the $(0, 1]^d$ hypercube into two subregions, I_A and I_C , where all responses in I_A generate a system alarm and all responses in I_C generate a system clear. When the system response function is measured for an item, its value is used to test a hypothesis, where the null and alternative hypotheses are

$$\begin{aligned} H_0 & : \text{The item is not a threat,} \\ H_A & : \text{The item is a threat.} \end{aligned}$$

Using the system response function, two errors are possible. Either the null hypothesis is falsely rejected (i.e., a Type I error), or the null hypothesis is falsely not rejected (i.e., a Type II error). To determine the probability of these errors, define the following events:

$$\begin{aligned} A & = \text{System gives Alarm response} \\ NA & = \text{System gives a Clear response} \\ T & = \text{Item is a threat} \\ NT & = \text{Item is not a threat} \end{aligned}$$

Using these events, the probabilities of the Type I and Type II errors are

$$P(A | NT) = \int_{\{\underline{x}: g(\underline{x}) \geq \gamma\}} f_{\underline{R}|NT}(\underline{x}) d\underline{x}, \quad (3.1)$$

and

$$P(NA | T) = \int_{\{\underline{r}:g(\underline{r})<\gamma\}} f_{\underline{R}|T}(\underline{r}) d\underline{r} \quad (3.2)$$

respectively. Additionally, the probabilities of the system response function returning the correct response are

$$P(A | T) = \int_{\{\underline{r}:g(\underline{r})\geq\gamma\}} f_{\underline{R}|T}(\underline{r}) d\underline{r} \quad (3.3)$$

and

$$P(NA | NT) = \int_{\{\underline{r}:g(\underline{r})<\gamma\}} f_{\underline{R}|NT}(\underline{r}) d\underline{r}. \quad (3.4)$$

If the probability of a threat is denoted by P_T , then from (3.1), (3.2), (3.3), and (3.4),

$$\begin{aligned} P(\text{False Clear}) &= P(NA \cap T) \\ &= P(T)P(NA | T) \\ &= P_T \int_{\{\underline{r}:g(\underline{r})<\gamma\}} f_{\underline{R}|T}(\underline{r}) d\underline{r}, \end{aligned} \quad (3.5)$$

$$\begin{aligned} P(\text{False Alarm}) &= P(A \cap NT) \\ &= P(NT)P(A | NT) \\ &= (1 - P_T) \int_{\{\underline{r}:g(\underline{r})\geq\gamma\}} f_{\underline{R}|NT}(\underline{r}) d\underline{r}, \end{aligned} \quad (3.6)$$

$$\begin{aligned} P(\text{True Alarm}) &= P(A \cap T) \\ &= P(T)P(A | T) \\ &= P_T \int_{\{\underline{r}:g(\underline{r})\geq\gamma\}} f_{\underline{R}|T}(\underline{r}) d\underline{r}, \end{aligned} \quad (3.7)$$

and

$$\begin{aligned} P(\text{True Clear}) &= P(NA \cap NT) \\ &= P(NT)P(NA | NT) \\ &= (1 - P_T) \int_{\{\underline{r}:g(\underline{r})<\gamma\}} f_{\underline{R}|NT}(\underline{r}) d\underline{r}. \end{aligned} \quad (3.8)$$

3.4 FALSE ALARM, FALSE CLEAR (FAFC) Problem

Denote the upper bound on the false clear probability by ε_{FC} . This value is set by the FAA and required at all access security control systems in airports throughout the United

States. Since the aviation industry must comply with this standard, the industry's goal is to meet the FAA requirements for false clears, while minimizing the false alarms (or, equivalently, maximizing the true clears). This leads to the mathematical programming problem formulation

$$\begin{aligned} & \text{Minimize } P(\text{False Alarm}) \\ & \text{subject to } P(\text{False Clear}) \leq \varepsilon_{FC}. \end{aligned} \quad (3.9)$$

The decision variables in (3.9) are the types of devices that are used in the security system, the system response function $g(\underline{r})$, and the γ value implemented. Given that the devices are fixed, the choice of a system response function and a value for γ that do not cause the false clear probability to exceed ε_{FC} defines the feasible region over which the minimum false alarm probability can be achieved.

To address this constrained optimization problem, suppose that the possible device responses are defined as a finite set, I , of non-intersecting subregions that define a partition of the entire $(0, 1]^d$ hypercube. This can be achieved, for example, by defining an equally spaced grid of size N over each device response (with each spacing of length $1/N$), so that $|I| = N^d$. Note that either of the sampling procedures described in Section 3.2 can be used to create the set I of subregions. Define n to be the number of subregions in I , where each subregion, $i \in I$, is indexed so that

$$I = \{i_0, i_2, \dots, i_{n-1}\}.$$

Each subregion has a *size*, measured by $q_1(i_j)$, $i_j \in I$, and a *utility*, measured by $q_2(i_j)$, $i_j \in I$. Define $q_1(i_j)$ and $q_2(i_j)$, respectively, as the probability that a system response falls into subregion i_j , given a threat item, and the probability that a system response falls into subregion i_j , given a non-threat item. Therefore,

$$q_1(i_j) = \int_{\{\underline{r}: \underline{r} \in \text{subregion } i_j\}} f_{\underline{R}|T}(\underline{r}) d\underline{r} \quad (3.10)$$

and

$$q_2(i_j) = \int_{\{\underline{r}: \underline{r} \in \text{subregion } i_j\}} f_{\underline{R}|NT}(\underline{r}) d\underline{r}. \quad (3.11)$$

Given this subregion discretization for $(0, 1]^d$, the goal in solving the mathematical program in (3.9) is to identify a set of subregions, I_C , such that $\sum_{i_j \in I_C} q_2(i_j)$ is maximized subject to $\sum_{i_j \in I_C} q_1(i_j) \leq \varepsilon = \varepsilon_{FC}/P_T$, where the set of subregions I_C represents the values for \underline{r} such that a system clear will occur following the inspection of an item. The complement of I_C , defined as I_A , denotes the set of subregions for which a system alarm will occur following the inspection of an item. This problem can be formally stated as a decision problem, termed the FALSE ALARM, FALSE CLEAR (FAFC) problem.

FALSE ALARM, FALSE CLEAR (FAFC) Problem

INSTANCE: - A finite set, I , of n non-intersecting subregions that cover the entire $(0, 1]^p$ region,
 - a size for each subregion, measured by $q_1(i_j)$, $i_j \in I$,
 - a utility for each subregion, measured by $q_2(i_j)$, $i_j \in I$,
 - an upper bound $\varepsilon \in [0, 1]$,
 - a lower bound $\delta \in [0, 1]$.

QUESTION: Is there a set of subregions $I_C \subseteq I$ such that $\sum_{i_j \in I_C} q_1(i_j) \leq \varepsilon$ and $\sum_{i_j \in I_C} q_2(i_j) \geq \delta$?

The set of subregions I_C defines a system response function such that the false alarm and false clear probabilities are appropriately bounded. This system response function is dependent on the types of devices in the system, which in turn are determined by the conditional probability density functions defined in Section 3.2. Therefore, the solution to the FAFC problem explicitly defines a system response function that meets the false alarm and false clear probability requirements.

Theorem 3.1 proves that the FAFC problem is NP-complete.

Theorem 3.1 *The FAFC problem is NP-complete.*

Proof 3.1 *First, the FAFC problem must be shown to be in NP. To see this, suppose that a set of subregions $I_C \subseteq I$ is guessed. It is clear that $\sum_{i_j \in I_C} q_1(i_j) \leq \varepsilon$ and $\sum_{i_j \in I_C} q_2(i_j) \geq \delta$ can both be checked in polynomial time in the size of the problem instance. Therefore, the FAFC problem is in NP.*

To complete the proof, a polynomial transformation from the KNAPSACK problem [10] to the FAFC problem must be constructed. The KNAPSACK problem is formally stated,

KNAPSACK Problem

INSTANCE: - A finite set, $U = \{u_0, u_2, \dots, u_{n-1}\}$,
 - a size $s(u_j) \in Z^+$, $u_j \in U$,
 - a value $v(u_j) \in Z^+$, $u_j \in U$,
 - an upper bound $B \in Z^+$,
 - a lower bound $K \in Z^+$.

QUESTION: Is there a subset $U' \subseteq U$ such that $\sum_{u_j \in U'} s(u_j) \leq B$ and $\sum_{u_j \in U'} v(u_j) \geq K$?

The transformation is obtained by defining, for an arbitrary instance of the KNAPSACK problem, a specific instance of FAFC where there is a one-to-one mapping between the finite set U and a set of non-intersecting subregions I that cover the entire $(0, 1]^d$ region. Define the size of subregion i_j by the size of the corresponding element $u_j \in U$, divided by the sum of all the sizes of the elements of U (i.e., $q_1(i_j) = s(u_j) / \sum_{u_j \in U} s(u_j)$). Define the utility of subregion i_j by the value of the corresponding element $u_j \in U$, divided by the sum of all the values of the elements of U (i.e., $q_2(i_j) = v(u_j) / \sum_{u_j \in U} v(u_j)$). Define ε to be B divided by the sum of all the sizes of the elements of U (i.e., $\varepsilon = B / \sum_{u_j \in U} s(u_j)$), and δ to be K divided by the sum of all values of the elements of U (i.e., $\delta = K / \sum_{u_j \in U} v(u_j)$).

This transformation can be done in linear time in the size of the KNAPSACK instance. To complete the proof, it is necessary to show that the response to the particular instance of the FAFC problem is yes if and only if the response to the arbitrary instance of the KNAPSACK problem is yes.

Suppose that the response to the FAFC problem instance is yes. Then there exists a set of subregions, I_C , such that $\sum_{i_j \in I_C} q_1(i_j) \leq \varepsilon$ and $\sum_{i_j \in I_C} q_2(i_j) \geq \delta$. By the subregion definition, these subregions map (one-to-one) to a subset $U' \in U$. The sum of the sizes for this subset are $\sum_{u_j \in U'} s(u_j) = (\sum_{i_j \in I_C} q_1(i_j)) (\sum_{u_j \in U} s(u_j))$, and $B = \varepsilon (\sum_{u_j \in U} s(u_j))$. Therefore, if $\sum_{i_j \in I_C} q_1(i_j) \leq \varepsilon$, then $\sum_{u_j \in U'} s(u_j) \leq B$. Similarly, if $\sum_{i_j \in I_C} q_2(i_j) \geq \delta$, then $\sum_{u_j \in U'} v(u_j) \geq K$. Therefore, the answer to the KNAPSACK problem instance is yes. Conversely, if the response to the KNAPSACK problem instance is yes, an identical argument shows that the answer to the FAFC problem instance is also yes.

The similarity between the FAFC problem and the discretized KNAPSACK problem is illustrated in the proof of Theorem 3.1. Since solving the mathematical program in (3.9), given a discretization of the $(0, 1]^d$ hypercube, is similar to solving a KNAPSACK problem, heuristic algorithms used to address the KNAPSACK problem can be adapted to address the FAFC problem.

3.5 KNAPSACK Problem Heuristics

To apply KNAPSACK problem heuristics to the FAFC problem, consider the clear region, I_C , to be a knapsack into which some or all of the n subregions that form the region I must be placed. The discretized mathematical program in (3.9) can be written as

$$\begin{aligned}
& \text{maximize} && \sum_{j=0}^{n-1} q_2(i_j)x(i_j) \\
& \text{subject to} && \sum_{j=0}^{n-1} q_1(i_j)x(i_j) \leq \varepsilon \\
& && x_j = \begin{cases} 1, & \text{if subregion } i_j \text{ is selected for the clear region;} \\ 0, & \text{otherwise,} \end{cases} \\
& && j = 0, \dots, n - 1.
\end{aligned}$$

Solving this problem using KNAPSACK heuristics can yield the clear region, I_C , for the given values of $q_1(i_j)$, $q_2(i_j)$, $i_j \in I$, and ε . These heuristics can also return the probability of a non-threat item eliciting a response in the clear region and the probability of a threat item eliciting a response in the clear region, defined respectively as

$$\sum_{i_j \in I_C} q_2(i_j)$$

and

$$\sum_{i_j \in I_C} q_1(i_j).$$

The resulting optimal values for the error probabilities, as defined in (3.5) and (3.6), are

$$P(\text{False Clear}) = P(T)P(NA | T) = P_T \sum_{i_j \in I_C} q_1(i_j) \quad (3.12)$$

and

$$P(\text{False Alarm}) = P(NT)(1 - P(NA | NT)) = (1 - P_T)(1 - \sum_{i_j \in I_C} q_2(i_j)). \quad (3.13)$$

Two heuristic methods for the KNAPSACK problem, the Greedy algorithm and the DP algorithm, are adapted to determine an approximate solution to the FAFC problem.

3.5.1 Reducing and Sorting

To reduce the amount of computation required by the Greedy and DP algorithms, certain subregions can be eliminated from I . In the FAFC problem, it is possible that $q_1(i_j) = 0$

or $q_2(i_j) = 0$ for some subregion i_j . If $q_1(i_j) = 0$, then there is a zero probability of a threat item eliciting a response in subregion i_j . If $q_2(i_j) = 0$, then there is zero probability of a non-threat item eliciting a response in subregion i_j . Since the goal of the FAFC problem is to identify subregions so as to maximize the probability of a non-threat item eliciting a response in the clear region while holding the probability of a threat item eliciting a response in the clear region to an upper bound, ε , a subregion i_j with $q_1(i_j) = 0$ should always be included. Conversely, a subregion with $q_2(i_j) = 0$ should never be included, unless the no alarm region, I_C , is the union of all subregions, I . An algorithm can be created to remove all subregions with $q_2(i_j) = 0$ and set aside all subregions with $q_1(i_j) = 0$, so that \hat{n} responses remain to be considered by the KNAPSACK problem heuristics. In addition, KNAPSACK problem heuristics typically require that the objects be ordered according to decreasing values of utility per unit size. The algorithm, termed the Reduce and Sort algorithm, removes unnecessary subregions and orders the remaining \hat{n} subregions so that

$$\frac{q_2(i_{(0)})}{q_1(i_{(0)})} \geq \frac{q_2(i_{(2)})}{q_1(i_{(2)})} \geq \dots \geq \frac{q_2(i_{(\hat{n}-1)})}{q_1(i_{(\hat{n}-1)})}.$$

To describe the Reduce and Sort algorithm, define $V \subseteq I$ as the set of subregions where $q_1(i_j) = 0$, v as the number of subregions in this set, and z_0 as the probability of non-threat item eliciting a response in these subregions. Define $\hat{I} \subseteq I$ to be the set of \hat{n} subregions that will be ordered (i.e., $q_1(i_j) \neq 0$ and $q_2(i_j) \neq 0$); subregions where $q_2(i_j) = 0$ need not be considered. Finally, define n to be the number of subregions in the original region, I , and let j be a counter through these subregions.

Reduce and Sort Algorithm

Initialization: Set V and \hat{I} to be empty.

Set $z_0 = 0$, $\hat{n} = 0$.

Set $j = 0$, $v = 0$.

Computation: While $j < n$ perform the following steps:

If $q_1(i_j) = 0$, add subregion i_j to V and set $z_0 = z_0 + q_2(i_j)$, $v = v + 1$.

If $q_1(i_j) \neq 0$ and $q_2(i_j) \neq 0$, add subregion i_j to \hat{I} and set $\hat{n} = \hat{n} + 1$.

Set $j = j + 1$.

Sorting: Sort the $\hat{n} \leq n$ subregions in set \hat{I} so that $\frac{q_2(i_{(0)})}{q_1(i_{(0)})} \geq \frac{q_2(i_{(2)})}{q_1(i_{(2)})} \geq \dots \geq \frac{q_2(i_{(\hat{n}-1)})}{q_1(i_{(\hat{n}-1)})}$.

Removing the unnecessary subregions can be done in $O(n)$ time. To sort the remaining subregions, an efficient sorting algorithm such as the *Heapsort* procedure, which has time complexity $O(n \log n)$, can be used. The following example illustrates the Reduce and Sort algorithm.

Example 3.1 Suppose $n = 16$,

$I = \{i_0, i_1, i_2, i_3, i_4, i_5, i_6, i_7, i_8, i_9, i_{10}, i_{11}, i_{12}, i_{13}, i_{14}, i_{15}\}$, where

$$\begin{array}{ll}
 q_1(i_0) = 0, & q_2(i_0) = 10/50, \\
 q_1(i_1) = 0, & q_2(i_1) = 8/50, \\
 q_1(i_2) = 1/50, & q_2(i_2) = 5/50, \\
 q_1(i_3) = 2/50, & q_2(i_3) = 2/50, \\
 q_1(i_4) = 0, & q_2(i_4) = 7/50, \\
 q_1(i_5) = 0, & q_2(i_5) = 5/50, \\
 q_1(i_6) = 2/50, & q_2(i_6) = 4/50, \\
 q_1(i_7) = 4/50, & q_2(i_7) = 1/50, \\
 q_1(i_8) = 3/50, & q_2(i_8) = 3/50, \\
 q_1(i_9) = 4/50, & q_2(i_9) = 2/50, \\
 q_1(i_{10}) = 5/50, & q_2(i_{10}) = 0, \\
 q_1(i_{11}) = 7/50, & q_2(i_{11}) = 0, \\
 q_1(i_{12}) = 1/50, & q_2(i_{12}) = 2/50, \\
 q_1(i_{13}) = 6/50, & q_2(i_{13}) = 1/50, \\
 q_1(i_{14}) = 7/50, & q_2(i_{14}) = 0, \\
 q_1(i_{15}) = 8/50, & q_2(i_{15}) = 0,
 \end{array}$$

$\varepsilon_{FC} = 0.0003$ and $P_T = 0.001$, hence $\varepsilon = \varepsilon_{FC}/P_T = 15/50$.

The Reduce and Sort algorithm yields the following results. Table 3.1 shows the value of each variable as the algorithm is executed. Upon completion of the algorithm,

$$V = \{i_0, i_1, i_4, i_5\}$$

and

$$\hat{I} = \{i_2, i_3, i_6, i_7, i_8, i_9, i_{12}, i_{13}\}.$$

The algorithm sorts, in ascending order, on the $\frac{q_2(i_i)}{q_1(i_j)}$ values of the subregions in \hat{I} , which are $\{5, 1, 2, \frac{1}{4}, 1, \frac{1}{2}, 2, \frac{1}{6}\}$. Applying heapsort yields the ordered subregions

$$\hat{I} = \{i_2, i_6, i_{12}, i_3, i_8, i_9, i_7, i_{13}\}.$$

3.5.2 Greedy Algorithm

Suppose that after the Reduce and Sort algorithm has been performed, there are $\hat{n} \leq n$ sorted subregions to be considered. These subregions make up the region $\hat{I} \subseteq I$.

j	v	z_0	\hat{n}
0	1	10/50	0
1	2	18/50	0
2	2	18/50	1
3	2	18/50	2
4	3	25/50	2
5	4	30/50	2
6	4	30/50	3
7	4	30/50	4
8	4	30/50	5
9	4	30/50	6
10	4	30/50	6
11	4	30/50	6
12	4	30/50	7
13	4	30/50	8
14	4	30/50	8
15	4	30/50	8

Table 3.1: Reduce and Sort Algorithm Example

The Greedy algorithm selects subregions consecutively until a subregion, with index k , is found that will cause the probability of a threat item eliciting a response in the clear region, $\sum_{j=1}^k q_1(i_j)$, to exceed the upper bound ε . Often, the union of subregions, $0, \dots, k-1$ provides a probability of a non-threat item eliciting a response in the clear region, $\sum_{j=0}^{k-1} q_2(i_j)$, close to the optimal solution. If, however, there exists a subregion, i_{j^*} , such that $q_2(i_{j^*}) > \sum_{j=0}^{k-1} q_2(i_j)$ and $q_1(i_{j^*}) \leq \varepsilon$, then that subregion alone will be closer to the optimal solution. The Greedy algorithm selects the solution, either the union of the $\{0, \dots, k-1\}$ subregions or the j^* subregion, that provides the maximum probability of a non-threat item eliciting a response in the clear region. The subregions selected will define the Clear region, I_C . Because the Greedy algorithm is a heuristic, this solution may not be optimal.

Define $\bar{\varepsilon}$ as the probability of a threat item eliciting a response in the clear region, I_C , z_g as the probability of a non-threat item eliciting a response in I_C , and j^* as the index of the subregion with the largest probability of a non-threat item eliciting a response in subregion i_{j^*} . Also define $x(i_j)$ so that $x(i_j) = 1$ if subregion i_j is included in the clear region, otherwise $x(i_j) = 0$. Let j be a counter through the \hat{n} subregions.

Greedy Algorithm

Initialization: Set $\bar{\varepsilon} = \varepsilon$, $z_g = 0$, $j^* = 0$, $j = 0$.

Computation: While $j < \hat{n}$ perform the following steps:

If $q_1(i_j) > \bar{\varepsilon}$ then set $x(i_j) = 0$.

Otherwise, set $x(i_j) = 1$, $\bar{\varepsilon} = \bar{\varepsilon} - q_1(i_j)$, $z_g = z_g + q_2(i_j)$.

If $q_2(i_j) > q_2(i_{j^*})$ and $q_1(i_j) \leq \varepsilon$, then set $j^* = j$.

Set $j = j + 1$.

If $q_2(i_{j^*}) > z_g$, then set $z_g = q_2(i_{j^*})$, $x(i_j) = 0$ for $j = 0, \dots, \hat{n} - 1$, and $i \neq j^*$. Set $\bar{\varepsilon} = q_1(i_{j^*})$ and set $x(i_{j^*}) = 1$.

The no alarm region is $I_C = V \cup \{i_j \in \hat{I} : x(i_j) = 1\}$. The probability of a non-threat item eliciting a response in the clear region is $P(NA|NT) = z_g + z_0$, and the probability of a threat item eliciting a response in the clear region is $P(NA|T) = \varepsilon - \bar{\varepsilon}$. The time complexity of the Greedy algorithm is $O(\hat{n})$. The following example illustrates the Greedy algorithm.

Example 3.1 (Continued): Suppose that the Reduce and Sort algorithm has been performed, so that $\hat{n} = 8$, and $\hat{I} = \{i_2, i_6, i_{12}, i_3, i_8, i_9, i_7, i_{13}\}$, where

$$\begin{aligned} q_1(i_2) &= 1/50, & q_2(i_2) &= 5/50, \\ q_1(i_6) &= 2/50, & q_2(i_6) &= 4/50, \end{aligned}$$

j	x_i	$\bar{\varepsilon}$	z_g	j^*
		15/50	0	0
0	1	14/50	5/50	0
1	1	12/50	9/50	0
2	1	11/50	11/50	0
3	1	9/50	13/50	0
4	1	6/50	16/50	0
5	1	2/50	18/50	0
6	0	2/50	18/50	0
7	0	2/50	18/50	0

Table 3.2: Greedy Algorithm Example

$$\begin{aligned}
q_1(i_{12}) &= 1/50, & q_2(i_{12}) &= 2/50, \\
q_1(i_3) &= 2/50, & q_2(i_3) &= 2/50, \\
q_1(i_8) &= 3/50, & q_2(i_8) &= 3/50, \\
q_1(i_9) &= 4/50, & q_2(i_9) &= 2/50, \\
q_1(i_7) &= 4/50, & q_2(i_7) &= 1/50, \\
q_1(i_{13}) &= 6/50, & q_2(i_{13}) &= 1/50,
\end{aligned}$$

$\varepsilon_{FC} = 0.0003$, $P_T = 0.001$, and $\varepsilon = \varepsilon_{FC}/P_T = 15/50$.

The Greedy algorithm yields the following results. Table 3.2 shows the value of each variable as the algorithm is executed.

Upon completion of the algorithm, the clear region is

$$\begin{aligned}
I_C &= V \cup \{i_j \in \hat{I} : x(i_j) = 1\} \\
&= \{i_0, i_1, i_4, i_5, i_2, i_6, i_{12}, i_3, i_8, i_9\},
\end{aligned}$$

the alarm region is

$$\begin{aligned}
I_A &= I - I_C \\
&= \{i_7, i_{10}, i_{11}, i_{13}, i_{14}, i_{15}\},
\end{aligned}$$

and the probabilities of a non-threat item and a threat item eliciting a response in the clear region are,

$$P(NA | NT) = z_g + z_0 = \frac{48}{50}$$

and

$$P(NA | T) = \varepsilon - \bar{\varepsilon} = \frac{13}{50},$$

respectively. Finally, the probabilities of a False Alarm and a False Clear are, from (3.12) and (3.13),

$$P(\text{False Clear}) = P(T)P(NA | T) = 0.00026$$

and

$$P(\text{False Alarm}) = P(NT)(1 - P(NA | NT)) = 0.03996.$$

Since this is a small example, all possible solutions can be enumerated. Performing such an enumeration verifies that this solution is indeed optimal.

3.5.3 Dynamic Programming (DP) Algorithm

The DP algorithm requires no specific ordering of the subregions. However, eliminating any subregion, i_j , with $q_1(i_j) = 0$ or $q_2(i_j) = 0$, and sorting those that remain according to decreasing $q_2(i_j)/q_1(i_j)$ ratios will improve the efficiency of DP by reducing the number of undominated states. (Undominated states will be introduced later in this discussion.) Therefore, suppose that the Reduce and Sort algorithm has been applied so that there are $\hat{n} \leq n$ sorted subregions to be considered. These subregions make up the region $\hat{I} \subseteq I$.

The DP algorithm also requires that $q_1(i_j)$ and $q_2(i_j)$ be integer valued for $i = 0, \dots, \hat{n} - 1$. To convert $q_1(i_j)$ and $q_2(i_j)$ to integer values, define M to be the number of sample points collected and let $q'_1(i_j) = Mq_1(i_j)$, $q'_2(i_j) = Mq_2(i_j)$, and $\varepsilon' = M\varepsilon$. Both proposed sampling methods estimate $q_1(i_j)$ and $q_2(i_j)$ by $h(\cdot)/M$, where $h(\cdot)$ is defined as the number of sample points that fall into subregion i given a threat or non-threat item. Therefore $Mq_1(i_j)$ and $Mq_2(i_j)$ are integer valued for $i = 0, \dots, \hat{n} - 1$. Since ε can be set to be a rational number with denominator M , then ε' can be assumed, without loss of generality, to be integer valued.

Given integers m ($1 \leq m \leq \hat{n}$) and $\hat{\varepsilon}$ ($0 \leq \hat{\varepsilon} \leq \varepsilon'$), consider the sub-instance of the FAFC problem consisting of subregions $0, \dots, m - 1$ and upper bound $\hat{\varepsilon}$. Let the optimal solution value of this sub-instance be defined as,

$$f_m(\hat{\varepsilon}) = \max\left\{ \sum_{j=0}^{m-1} q'_2(i_j)x(i_j) : \sum_{j=0}^{m-1} q'_1(i_j)x(i_j) \leq \hat{\varepsilon}, x(i_j) = 0, 1, j = 0, \dots, m - 1 \right\},$$

where the $x(i_j)$ values are chosen so as to maximize $f_m(\hat{\varepsilon})$.

The DP algorithm considers \hat{n} stages, where stage m ($1 \leq m \leq \hat{n}$) represents the FAFC problem with m subregions, $\{i_0, \dots, i_{m-1}\}$. The DP algorithm computes, at each stage $m > 1$, the values $f_m(\hat{\varepsilon})$ (for $\hat{\varepsilon}$ increasing from 0 to ε') using the recursion [6]:

$$f_m(\hat{\varepsilon}) = \begin{cases} f_{m-1}(\hat{\varepsilon}) & \text{for } \hat{\varepsilon} = 0, \dots, q'_1(i_m) - 1. \\ \max \{f_{m-1}(\hat{\varepsilon}), f_{m-1}(\hat{\varepsilon} - q'_1(i_m)) + q'_2(i_m)\} & \text{for } \hat{\varepsilon} = q'_1(i_m), \dots, \varepsilon. \end{cases}$$

States are defined as the feasible solutions (i.e., union of subregions) corresponding to $f_m(\hat{\varepsilon})$ for each value of $\hat{\varepsilon}$. The optimal solution of the FAFC problem is the state corresponding to $f_{\hat{n}}(\varepsilon')$.

Define $p_1(\hat{s})$ as the probability of a non-threat item eliciting a response in the clear region for state \hat{s} . A *dominated state* is a state $\hat{\varepsilon}$ for which there exists a state y with $p_1(y) \geq p_1(\hat{\varepsilon})$ and $y \leq \hat{\varepsilon}$. The number of states considered at each stage can be reduced by eliminating dominated states.

Before execution of the DP algorithm for stage m , $m = 2, \dots, \hat{n}$, define:

$$s_{m-1} = \text{number of states at stage } (m-1), \quad (3.14)$$

$$b = 2^{m-1}, \quad (3.15)$$

$$w_1(\hat{s}) = \text{probability of a threat item eliciting a response in the clear region} \quad (3.16) \\ \text{for state } \hat{s}, (\hat{s} = 1, \dots, s_{m-1})$$

$$p_1(\hat{s}) = \text{probability of a non-threat item eliciting a response in the clear} \quad (3.17) \\ \text{region for state } \hat{s}, (\hat{s} = 1, \dots, s_{m-1})$$

$$X1_{\hat{s}} = \{x(i_{m-1}), x(i_{m-2}), \dots, x(i_1)\}, (\hat{s} = 1, \dots, s_{m-1}) \quad (3.18)$$

$$\text{where } x(i_j) = \begin{cases} 1 & \text{if subregion } i_j \text{ is included in the clear region at state } \hat{s} \\ 0 & \text{otherwise.} \end{cases}$$

To reduce the amount of storage necessary, $X1_{\hat{s}}$ is encoded as a bit string.

The DP algorithm uses index \hat{s} to scan the states of the current stage and index k to store the states of the new stage. After execution of the procedure for stage m , values (3.14) and (3.15) are relative to the next stage, $m+1$. Define \underline{w}_2 , \underline{p}_2 , and $X2$ as the values of (3.17), (3.18), and (3.18), relative to stage $m+1$.

Dynamic Programming Algorithm [34]**Initialization:** Set $w_1(0) = 0$, $p_1(0) = 0$, $X1_0 = 0$.Set $w_1(1) = q'_1(i_0)$, $p_1(1) = q'_2(i_0)$, $X1_1 = 1$.Set $s_1 = 1$, $b = 2$, $m = 2$.**Computation:** For each stage m ($2 \leq m \leq \hat{n}$):**Inititalization:** Set $\hat{s} = 0$, $k = 0$, $h = 1$, $y = q'_1(i_{m-1})$.Set $w_1(s_{m-1} + 1) = \varepsilon' + 1$.Set $w_2(0) = 0$, $p_2(0) = 0$, $X2_0 = 0$.**While** $\min(y, w_1(h)) \leq \varepsilon'$:**if** $w_1(h) \leq y$:Set $p = p_1(h)$, $x = X1_h$.**if** $w_1(h) = y$:**if** $p_1(\hat{s}) + q'_2(i_{m-1}) > p$:Set $p = p_1(\hat{s}) + q'_2(i_{m-1})$, $x = X1_{\hat{s}} + b$.Set $\hat{s} = \hat{s} + 1$, $y = w_1(\hat{s}) + q'_1(i_{m-1})$.**if** $p > p_2(k)$:Set $k = k + 1$.Set $w_2(k) = w_1(h)$, $p_2(k) = p$, $X2_k = x$.Set $h = h + 1$.**Otherwise:****if** $p_1(\hat{s}) + q'_2(i_{m-1}) > p_2(k)$:Set $k = k + 1$.Set $w_2(k) = y$, $p_2(k) = p_1(\hat{s}) + q'_2(i_{m-1})$,Set $X2_k = X1_{\hat{s}} + b$.Set $\hat{s} = \hat{s} + 1$, $y = w_1(\hat{s}) + q'_1(i_{m-1})$.Set $s_m = k$, $b = 2b$.Rename \underline{w}_2 , \underline{p}_2 , and $X2$ as \underline{w}_1 , \underline{p}_1 , and $X1$, respectively.Set $m = m + 1$.**Decode:** Determine \underline{x} by decoding $X1_k$ as follows:Set $div = X1_k$, $rem = X1_k \bmod(2)$, $cnt = 1$.**While** $div > 0$:Set $x(i_{cnt}) = rem$.Set $div = div/2$, $rem = div \bmod(2)$.Set $cnt = cnt + 1$.

After all computations are complete for stage m , $w_1(s_m)$ and $p_1(s_m)$ are the optimal values of $P(NA|T)$ and $P(NA|NT)$ for the region made up of subregions $\{i_0, \dots, i_{m-1}\}$. The binary representation of $X1_{s_m}$ gives the optimal clear region for $\{i_0, \dots, i_{m-1}\}$. The final stage, \hat{n} , gives the optimal values that correspond to the region made up of all \hat{n} subregions. The clear region is $I_C = V \cup \{i_j \in \hat{I} : x(i_j) = 1\}$, where the $x(i_j)$ values are determined by

decoding the binary string, $X1_{s_{\hat{n}}}$. The probability of a non-threat item eliciting a response in the clear region is $P(NA|NT) = p_1(s_{\hat{n}}) + z_0$ and the probability of a threat item eliciting a response in the clear region is $P(NA|T) = w_1(s_{\hat{n}})$. To determine the time complexity of the DP algorithm, note that the order of the **For** loop is \hat{n} and the order of the **While** loop is the upper bound on s_m . This bound is given by $\min(2^{\hat{n}}, \varepsilon')$ [23]. Combining these two orders results in the time complexity of the DP Algorithm to be $O(\min(2^{\hat{n}+1}, n\varepsilon'))$. The following example illustrates the DP algorithm.

Example 3.1 (Continued): *Suppose that the Reduce and Sort algorithm has been performed, so that $\hat{n} = 8$, and $\hat{I} = \{i_2, i_6, i_{12}, i_3, i_8, i_9, i_7, i_{13}\}$, where*

$$\begin{array}{ll} q_1(i_2) = 1/50, & q_2(i_2) = 5/50, \\ q_1(i_6) = 2/50, & q_2(i_6) = 4/50, \\ q_1(i_{12}) = 1/50, & q_2(i_{12}) = 2/50, \\ q_1(i_3) = 2/50, & q_2(i_3) = 2/50, \\ q_1(i_8) = 3/50, & q_2(i_8) = 3/50, \\ q_1(i_9) = 4/50, & q_2(i_9) = 2/50, \\ q_1(i_7) = 4/50, & q_2(i_7) = 1/50, \\ q_1(i_{13}) = 6/50, & q_2(i_{13}) = 1/50, \end{array}$$

$$\varepsilon_{FC} = 0.0003, P_T = 0.001, \text{ and } \varepsilon = \varepsilon_{FC}/P_T = 15/50.$$

These values must be converted to integers. We will assume that $M = 50$, so

$$\begin{array}{ll} q'_1(i_2) = 1, & q'_2(i_2) = 5, \\ q'_1(i_6) = 2, & q'_2(i_6) = 4, \\ q'_1(i_{12}) = 1, & q'_2(i_{12}) = 2, \\ q'_1(i_3) = 2, & q'_2(i_3) = 2, \\ q'_1(i_8) = 3, & q'_2(i_8) = 3, \\ q'_1(i_9) = 4, & q'_2(i_9) = 2, \\ q'_1(i_7) = 4, & q'_2(i_7) = 1, \\ q'_1(i_{13}) = 6, & q'_2(i_{13}) = 1, \end{array}$$

and $\varepsilon' = 15$.

The DP algorithm yields the following results. Table 3.3 shows the value of each variable as the algorithm is executed. Upon completion of the algorithm,

m	s_m	$w_1(s_m)$	$p_1(s_m)$	I_C
1	1	1	5	$\{i_0\}$
2	2	3	9	$\{i_0, i_1\}$
3	3	4	11	$\{i_0, i_1, i_2\}$
4	4	6	13	$\{i_0, i_1, i_2, i_3\}$
5	6	9	16	$\{i_0, i_1, i_2, i_3, i_4\}$
6	7	13	18	$\{i_0, i_1, i_2, i_3, i_4, i_5\}$
7	7	13	18	$\{i_0, i_1, i_2, i_3, i_4, i_5\}$
8	7	13	18	$\{i_0, i_1, i_2, i_3, i_4, i_5\}$

Table 3.3: Dynamic Programming Algorithm Example

The clear region is

$$\begin{aligned} I_C &= V \cup \{i_j \in \hat{I} : x(i_j) = 1\} \\ &= \{i_0, i_1, i_4, i_5, i_2, i_6, i_{12}, i_3, i_8, i_9\}, \end{aligned}$$

the alarm region is

$$\begin{aligned} I_A &= I - I_C \\ &= \{i_7, i_{10}, i_{11}, i_{13}, i_{14}, i_{15}\}, \end{aligned}$$

the probabilities of a non-threat item and a threat item eliciting a response in the clear region are,

$$P(NA | NT) = \frac{p_1(7)}{50} + z_0 = \frac{48}{50}$$

and

$$P(NA | T) = \frac{w_1(7)}{50} = \frac{13}{50},$$

respectively. Finally, the probabilities of a False Alarm and a False Clear are, from (3.12) and (3.13),

$$P(\text{False Clear}) = P(T)P(NA | T) = 0.00026$$

and

$$P(\text{False Alarm}) = P(NT)(1 - P(NA | NT)) = 0.03996.$$

Since this is a small example, all possible solutions can be enumerated. Performing this enumeration verifies that this solution is indeed optimal.

Limitations of the DP Algorithm

The DP algorithm has certain computational limitations that should be noted. These limitations are caused by the value of $X1_{s_{\hat{n}}}$. Since this number must be converted to a binary string, its size is limited by the computational precision available. If a computer has precision to 48 bits, $X1_k \leq 10^{48}$. For particular examples (e.g., two devices with grid size 20), values of $X1_s$ larger than this limit have been observed. The only other alternative is to pass a two-dimensional vector, $X[s_{\hat{n}}][\hat{n}]$, that will record the appropriate values of $x_i, i = 1, \dots, \hat{n}$ for each $(w_1(i), p_1(i))$ value. As previously stated, the upper bound of s_m is given by $\min(2^{\hat{n}}, \varepsilon')$. The value $2^{\hat{n}}$ is exponential and can become very large. For a real example where M is large, $\varepsilon' = M\varepsilon$ can also be very large. For example, if $\varepsilon = 0.01$ and $M = 10,000,000$, $\varepsilon' = 100,000$. For the two-device example with grid size 20, the sizes of the \underline{w}_1 and \underline{p}_1 vectors are observed at 7000. Therefore a two-dimensional vector of size 7000×400 is required, which exceeds the available memory of a SUN ULTRA 1 with 128 megabytes of RAM.

Because of these limitations, the DP algorithm can not be used for large examples, and hence cannot be applied for many real-life problems. Therefore it is recommended that the Greedy algorithm be used for all but the smallest examples.

Chapter 4

Analytical Results

4.1 Choosing the Density Functions

Consider a system where linear marginal probability density functions are chosen for the device responses so that the marginal density function conditional on a threat item, $f_{R_i|T}(r_i)$, is symmetric to the marginal density function conditional on a non-threat item, $f_{R_i|NT}(r_i)$, for $i = 1, \dots, d$. In other words, the probability of a response near one, given a threat item, is equal to the probability of a response near zero, given a non-threat item. Additionally, all device responses are assumed to be independent of one another.

Define α and β such that

$$\alpha = f_{R_i|T}(0) = f_{R_i|NT}(1) \quad (4.1)$$

and

$$\beta = f_{R_i|T}(1) = f_{R_i|NT}(0). \quad (4.2)$$

Then the density functions are,

$$f_{R_i|T}(r_i) = \alpha + (\beta - \alpha)r_i \quad (4.3)$$

and

$$f_{R_i|NT}(r_i) = \beta + (\alpha - \beta)r_i. \quad (4.4)$$

By definition, $\int_0^1 f_{R_i|T}(r_i) dr_i = 1$ and $\int_0^1 f_{R_i|NT}(r_i) dr_i = 1$. Therefore,

$$\int_0^1 \alpha + (\beta - \alpha)r_i dr_i = 1 \quad (4.5)$$

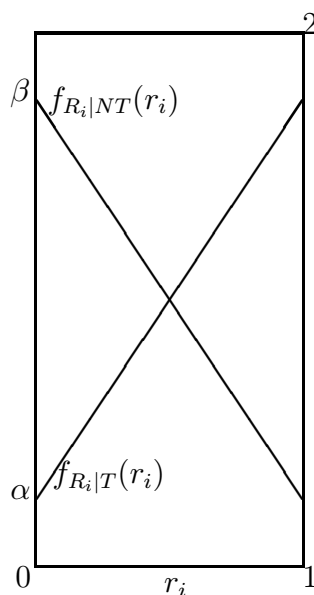


Figure 4.1: Joint Probability Density Functions

or

$$\alpha + \frac{1}{2}(\beta - \alpha) = 1 \quad (4.6)$$

and

$$\int_0^1 \beta + (\alpha - \beta)r_i \, dr_i = 1 \quad (4.7)$$

or

$$\beta + \frac{1}{2}(\alpha - \beta) = 1. \quad (4.8)$$

Therefore, $\beta = 2 - \alpha$, so the marginal density functions may be written as

$$f_{R_i|T}(r_i) = \alpha + 2(1 - \alpha)r_i \quad (4.9)$$

and

$$f_{R_i|NT}(r_i) = \beta + 2(1 - \beta)r_i. \quad (4.10)$$

In addition, it is required that $\alpha > 0$ and $\beta > 0$. Therefore, $0 \leq \alpha \leq \beta \leq 2$. The marginal density functions, $f_{R_i|T}(r_i)$ and $f_{R_i|NT}(r_i)$, can be represented graphically, as depicted in Figure 4.1.

4.2 Two Continuous-Response Devices

To solve the FAFC problem analytically, a system response function, $g(\underline{r})$, and a limit, γ , must be chosen. Consider the two-device system where all device responses are assumed to be independent. Let the marginal probability density functions be of the form (4.9) and (4.10), where $f_{R_1|T}(r_1) = f_{R_2|T}(r_2)$ and $f_{R_1|NT}(r_1) = f_{R_2|NT}(r_2)$. Because the devices are independent, the joint probability density functions are defined to be

$$f_{\underline{R}|T}(\underline{r}) = f_{R_1|T}(r_1)f_{R_2|T}(r_2)$$

and

$$f_{\underline{R}|NT}(\underline{r}) = f_{R_1|NT}(r_1)f_{R_2|NT}(r_2).$$

One possible system response function is the piece-wise linear function,

$$g(\underline{r}) = \begin{cases} a_1r_1 + r_2 & 0 \leq r_1 \leq m, & 0 \leq r_2 \leq 1, \\ a_2r_1 + r_2 & m < r_1 \leq \gamma_1, & 0 \leq r_2 \leq 1, \\ 0 & \gamma_1 < r_1 \leq 1, & 0 \leq r_2 \leq 1, \end{cases} \quad (4.11)$$

where a system alarm is sounded if

$$g(\underline{r}) \geq \begin{cases} \gamma_1 & 0 \leq r_1 \leq m, & 0 \leq r_2 \leq 1, \\ \gamma_2 & m < r_1 \leq \gamma_1, & 0 \leq r_2 \leq 1, \\ 0 & \gamma_1 < r_1 \leq 1, & 0 \leq r_2 \leq 1. \end{cases} \quad (4.12)$$

Because the density functions are identical for the two devices, the probability of a threat or a non-threat item eliciting a response near (0,1) is equal to the probability of an item eliciting a response near (1,0). Therefore, let the point of intersection of the two lines that make up the response function be the point (m, m) . In other words,

$$a_1m + m = \gamma_1 \quad (4.13)$$

and

$$a_2m + m = \gamma_2. \quad (4.14)$$

The piece-wise linear response function is depicted by Figure 4.2.

Using the piece-wise response function described in (4.11), the best values of $P(\text{False Alarm})$ and $P(\text{False Clear})$ possible for the two-device example are given by the following theorem. These values can be viewed as upper bounds of $P(\text{False Alarm})$ and $P(\text{False Clear})$.

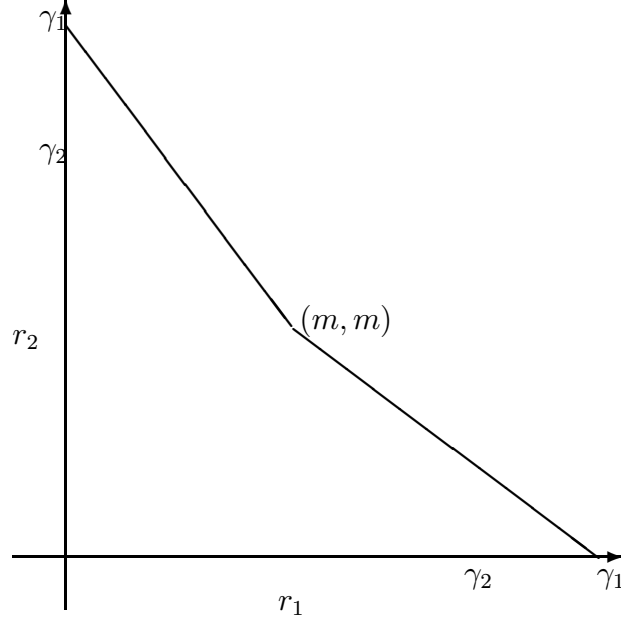


Figure 4.2: Piece-wise Response Function

Theorem 4.1 For a piece-wise linear system response function of the form given in (4.11), the probabilities of false clear and false alarm are

$$\begin{aligned}
 P(\text{False Clear}) &= P_T * & (4.15) \\
 &\left(\int_0^{\gamma_1 a_2 / (a_2 + 1)} \int_0^{\gamma_1 - r_1 / a_2} (\alpha + 2(1 - \alpha)r_1)(\alpha + 2(1 - \alpha)r_2) dr_2 dr_1 \right. \\
 &\quad \left. + \int_{\gamma_1 a_2 / (a_2 + 1)}^{\gamma_1} \int_0^{a_2(\gamma_1 - r_1)} (\alpha + 2(1 - \alpha)r_1)(\alpha + 2(1 - \alpha)r_2) dr_2 dr_1 \right)
 \end{aligned}$$

and

$$\begin{aligned}
 P(\text{False Alarm}) &= (1 - P_T) * & (4.16) \\
 &\left(\int_0^{\gamma_1 a_2 / (a_2 + 1)} \int_{\gamma_1 - r_1 / a_2}^1 (\beta + 2(1 - \beta)r_1)(\beta + 2(1 - \beta)r_2) dr_2 dr_1 \right. \\
 &\quad \left. + \int_{\gamma_1 a_2 / (a_2 + 1)}^{\gamma_1} \int_{a_2(\gamma_1 - r_1)}^1 (\beta + 2(1 - \beta)r_1)(\beta + 2(1 - \beta)r_2) dr_2 dr_1 \right. \\
 &\quad \left. + \int_{\gamma_1}^1 \int_0^1 (\beta + 2(1 - \beta)r_1)(\beta + 2(1 - \beta)r_2) dr_2 dr_1 \right).
 \end{aligned}$$

Therefore the optimal values of $P(\text{False Clear})$ and $P(\text{False Alarm})$ for this response

function can be found by calculating

$$(\gamma_1^*, a_2^*) = \{(\gamma_1, a_2) : P(\text{False Clear}) \leq \varepsilon_{FC} \text{ and } P(\text{False Alarm}) \text{ is minimized}\}. \quad (4.17)$$

After γ_1^* and a_2^* are determined, $P(\text{False Clear})$ is a function of α and P_T , and $P(\text{False Alarm})$ is a function of β and P_T .

Proof 4.1 From (3.2) and (3.1),

$$\begin{aligned} P(NA | T) &= \int_0^m \int_0^{\gamma_1 - a_1 r_1} f_{R_1|T}(r_1) f_{R_2|T}(r_2) dr_2 dr_1 \\ &\quad + \int_m^{\gamma_1} \int_0^{\gamma_2 - a_2 r_1} f_{R_1|T}(r_1) f_{R_2|T}(r_2) dr_2 dr_1 \end{aligned} \quad (4.18)$$

and

$$\begin{aligned} P(A | NT) &= \int_0^m \int_{\gamma_1 - a_1 r_1}^1 f_{R_1|NT}(r_1) f_{R_2|NT}(r_2) dr_2 dr_1 \\ &\quad + \int_m^{\gamma_1} \int_{\gamma_2 - a_2 r_1}^1 f_{R_1|NT}(r_1) f_{R_2|NT}(r_2) dr_2 dr_1 \\ &\quad + \int_{\gamma_1}^1 \int_0^1 f_{R_1|NT}(r_1) f_{R_2|NT}(r_2) dr_2 dr_1. \end{aligned} \quad (4.19)$$

Evaluating (4.18) and (4.19) results in five unknowns, a_1 , a_2 , γ_1 , γ_2 , and m . However, the number of unknowns may be reduced by recognizing from (4.13) and (4.14) that

$$\gamma_1 - a_1 m = \gamma_2 - a_2 m. \quad (4.20)$$

Therefore,

$$m = \frac{\gamma_2 - \gamma_1}{a_2 - a_1} \quad (4.21)$$

Because the lines that form the response function intercept at (m, m) , the r_1 intercept of the line $(a_1 r_1 + r_2)$ is γ_2 and the r_1 intercept of the line $(a_2 r_1 + r_2)$ is γ_1 . Therefore,

$$a_1 \gamma_2 = \gamma_1 \quad (4.22)$$

and

$$a_2 \gamma_1 = \gamma_2. \quad (4.23)$$

From (4.22) and (4.23),

$$a_1 = 1/a_2. \quad (4.24)$$

Given (4.23) and (4.24), (4.21) can be reduced to

$$m = \frac{\gamma_1 a_2}{a_2 + 1}. \quad (4.25)$$

Therefore (4.18) and (4.19) can be reduced to two equations in two unknowns,

$$\begin{aligned} P(NA | T) = &= \int_0^{\gamma_1 a_2 / (a_2 + 1)} \int_0^{\gamma_1 - r_1 / a_2} (\alpha + 2(1 - \alpha)r_1)(\alpha + 2(1 - \alpha)r_2) dr_2 dr_1 \\ &+ \int_{\gamma_1 a_2 / (a_2 + 1)}^{\gamma_1} \int_0^{a_2(\gamma_1 - r_1)} (\alpha + 2(1 - \alpha)r_1)(\alpha + 2(1 - \alpha)r_2) dr_2 dr_1 \end{aligned} \quad (4.26)$$

and

$$\begin{aligned} P(A | NT) = &= \int_0^{\gamma_1 a_2 / (a_2 + 1)} \int_{\gamma_1 - r_1 / a_2}^1 (\beta + 2(1 - \beta)r_1)(\beta + 2(1 - \beta)r_2) dr_2 dr_1 \\ &+ \int_{\gamma_1 a_2 / (a_2 + 1)}^{\gamma_1} \int_{a_2(\gamma_1 - r_1)}^1 (\beta + 2(1 - \beta)r_1)(\beta + 2(1 - \beta)r_2) dr_2 dr_1 \\ &+ \int_{\gamma_1}^1 \int_0^1 (\beta + 2(1 - \beta)r_1)(\beta + 2(1 - \beta)r_2) dr_2 dr_1. \end{aligned} \quad (4.27)$$

Mathematica can be used to integrate (4.26) and (4.27) to represent $P(NA | T)$ and $P(A | NT)$ in terms of γ_1 and a_2 . Finally, $P(\text{False Clear})$ and $P(\text{False Alarm})$ are calculated according to (3.5) and (3.6).

To find the values of γ_1 and a_2 that minimize $P(\text{False Alarm})$, (4.15) is set equal to the given value of ε_{FC} and γ_1 is found in terms of a_2 . This value of γ_1 is substituted into (4.16), so that $P(\text{False Alarm})$ is an equation in one unknown, a_2 . Mathematica is used to calculate the first derivative of $P(\text{False Alarm})$, which is then set to zero and solved for a_2 . By the laws of calculus [32], the resulting value of a_2 minimizes (4.16).

Table 4.1 shows the values of all coefficients for $P_T = 10^{-6}$ and $\varepsilon_{FC} = \{10^{-7}, 10^{-8}, 10^{-9}\}$. The values (α, β) have been chosen as $(0, 2)$, so the marginal probability density functions, conditional on threat or non-threat items passing through two independent devices, are

$$f_{R_i|T}(r_i) = 2r_i, \quad i = 1, 2 \quad (4.28)$$

and

$$f_{R_i|NT}(r_i) = 2 - 2r_i, \quad i = 1, 2. \quad (4.29)$$

Therefore, the joint probability density functions are

$$f_{\underline{R}|T}(\underline{x}) = 4r_1 r_2 \quad (4.30)$$

and

$$f_{\underline{R}|NT}(\underline{x}) = (2 - 2r_1)(2 - 2r_2). \quad (4.31)$$

	ε_{FC}		
	10^{-7}	10^{-8}	10^{-9}
a_1	1.3356	0.3086	0.1662
a_2	0.7487	3.2399	6.0183
γ_1	0.9690	0.7780	0.5823
γ_2	0.7255	0.2401	0.0968
m	0.4149	0.1835	0.0830
$P(\text{False Clear})$	1.00E-07	1.00E-08	1.00E-09
$P(\text{False Alarm})$	0.2574	0.6386	0.8646

Table 4.1: Analytical Results for Two-Device System

It may seem that the analytical value for $P(\text{False Alarm})$ is unnaturally high. This, however, can be explained. For the two-device example, the joint probability density functions of device responses are continuous functions that span the entire $[0,1]$ space. In other words, there is some probability, although small, that a threat item will elicit a response near zero or a non-threat item will elicit a response near one. Therefore, the region of the response surface that is normally reserved for non-threat item responses will contain some threat item responses and visa-versa. In a real example, it is reasonable to expect that a non-threat item will never elicit a response above some limit, u , where u is close to one. Conversely, a threat item will never elicit a response below some limit l , where l is close to zero. These bounds would reduce the probability of a false alarm considerably. Choosing the marginal probability density functions so that the density function conditional on a threat item is symmetric to the density function conditional on a non-threat item also causes $P(\text{False Alarm})$ to be high. As the false clear bound, ε_{FC} decreases, the feasible clear region is limited to small values of \underline{r} , where $F_{R_i|T}(r_i)$ for each device, i , is near α . This forces the alarm region to cover the region in which $F_{R_i|NT}(r_i)$ for each device, i , is large. In essence, as the amount of false clears allowed is decreased, the level of false alarm is increased at the same rate.

4.3 Five Continuous-Response Devices

Consider the five-device system where all device responses are assumed to be independent. Let the marginal probability density functions be of the form (4.9) and (4.10). Because the devices are independent, the joint probability density functions are defined to be

$$f_{\underline{R}|T}(\underline{r}) = \prod_{i=1}^5 f_{R_i|T}(r_i) \quad (4.32)$$

and

$$f_{\underline{R}|NT}(\underline{r}) = \prod_{i=1}^5 f_{R_i|NT}(r_i). \quad (4.33)$$

Many of the possible system response functions for this system prove to be computationally challenging, because of the integration involved. For example, consider the linear response function, $g(\underline{r}) = r_1 + r_2 + r_3 + r_4 + r_5$. Determining the optimal values of $P(\text{False Clear})$ and $P(\text{False Alarm})$ for this response function would require performing the integrations

$$\int_{\{\underline{r}: r_1+r_2+r_3+r_4+r_5 < \gamma\}} f_{\underline{R}|T}(\underline{r}) dr_1 \dots dr_5$$

and

$$\int_{\{\underline{r}: r_1+r_2+r_3+r_4+r_5 \geq \gamma\}} f_{\underline{R}|NT}(\underline{r}) dr_1 \dots dr_5.$$

Determining the appropriate limits of integration for these equations is extremely difficult.

One response function that can be integrated assigns an upper limit, u_i , to each device, $i = 1, \dots, 5$. If the response from any device, i , exceeds its predetermined limit, u_i , then a system alarm is declared. This system response function can be stated formally,

$$g(\underline{r}) = \begin{cases} 1, & \text{if } r_i \geq u_i \text{ for any } i \\ 0, & \text{otherwise} \end{cases} \quad (4.34)$$

where a system alarm is sounded if

$$g(\underline{r}) \geq 1. \quad (4.35)$$

The best levels of $P(\text{False Alarm})$ and $P(\text{False Clear})$ for the five-device example, using the system response function described in (4.34), are given by the following theorem. These values can be viewed as upper bounds of $P(\text{False Alarm})$ and $P(\text{False Clear})$.

Theorem 4.2 *For a system response function of the form given in (4.34), the probabilities of false clear and false alarm are*

$$P(\text{False Clear}) = P_T \prod_{i=1}^5 \int_0^{u_i} (\alpha + 2(1 - \alpha)r_i) dr_i \quad (4.36)$$

and

$$P(\text{False Alarm}) = (1 - P_T) \left(1 - \prod_{i=1}^5 \left(\int_0^{u_i} (\beta + 2(1 - \beta)r_i) dr_i\right)\right). \quad (4.37)$$

Therefore the optimal values of $P(\text{False Clear})$ and $P(\text{False Alarm})$ can be found by calculating

$$(u_1^*, u_2^*, u_3^*, u_4^*, u_5^*) = \{(u_1, u_2, u_3, u_4, u_5) : P(\text{False Clear}) \leq \varepsilon_{FC} \text{ and } P(\text{False Alarm}) \text{ is minimized}\}. \quad (4.38)$$

After $u_i^*, i = 1, \dots, 5$ are determined, $P(\text{False Clear})$ is a function of α and P_T , and $P(\text{False Alarm})$ is a function of β and P_T .

Proof 4.2 A system alarm occurs if the response from any device, i , exceeds its limit, u_i . Conversely, a system clear occurs if the responses from all devices do not exceed their respective limits. Note that all device responses are assumed to be independent. Therefore,

$$\begin{aligned} P(\text{Clear}) &= P((D_1 \leq u_1) \cap (D_2 \leq u_2) \cap (D_3 \leq u_3) \cap (D_4 \leq u_4) \cap (D_5 \leq u_5)) \\ &= P(D_1 \leq u_1)P(D_2 \leq u_2)P(D_3 \leq u_3)P(D_4 \leq u_4)P(D_5 \leq u_5) \end{aligned}$$

and

$$\begin{aligned} P(\text{Alarm}) &= 1 - P(\text{Clear}) \\ &= 1 - (P(D_1 \leq u_1)P(D_2 \leq u_2)P(D_3 \leq u_3)P(D_4 \leq u_4)P(D_5 \leq u_5)). \end{aligned}$$

From (3.2) and (3.1),

$$P(\text{NA} | T) = \prod_{i=1}^5 \int_0^{u_i} f_{R_i|T}(r_i) dr_i \quad (4.39)$$

and

$$P(A | NT) = 1 - \prod_{i=1}^5 \left(1 - \int_{u_1}^1 f_{R_i|NT}(r_i) dr_i\right). \quad (4.40)$$

Mathematica can be used to integrate (4.39) and (4.40) to represent $P(\text{NA} | T)$ and $P(A | NT)$ in terms of u_1, u_2, u_3, u_4 , and u_5 . Finally, $P(\text{False Clear})$ and $P(\text{False Alarm})$ are computed according to (3.5) and (3.6). The resulting five equations in two unknowns cannot be simplified any further, so an approximation method must be used to find the optimal values of u_1, u_2, u_3, u_4 , and u_5 .

To find the values of $u_i, i = 1, \dots, 5$ that minimize $P(\text{False Alarm})$, (4.36) is set equal to the given value of ε_{FC} and u_1 is found in terms of u_2, u_3, u_4 , and u_5 . The result is one

equation in four unknowns. To minimize $P(\text{False Alarm})$ in terms of these four variables, the Mathematica function *FindMinimum*, which applies the method of steepest descent to find a local minimum, was used.

Table 4.2 shows the values of all coefficients for $P_T = 10^{-6}$ and $\varepsilon_{FC} = \{10^{-7}, 10^{-8}, 10^{-9}\}$. The (α, β) pairs used are $(0, 2)$, $(0.1, 1.9)$, $(0.2, 1.8)$, $(0.3, 1.7)$, and $(0.4, 1.6)$. Therefore, the marginal probability density functions, conditional on a threat item being passed through device i , are

$$f_{R_1|T}(r_1) = 2r_1, \quad (4.41)$$

$$f_{R_2|T}(r_2) = 0.1 + 1.8r_2, \quad (4.42)$$

$$f_{R_3|T}(r_3) = 0.2 + 1.6r_3, \quad (4.43)$$

$$f_{R_4|T}(r_4) = 0.3 + 1.4r_4, \text{ and} \quad (4.44)$$

$$f_{R_5|T}(r_5) = 0.4 + 1.2r_5. \quad (4.45)$$

The marginal probability density functions, conditional on a non-threat item being passed through device i , are

$$f_{R_1|NT}(r_1) = 2 - 2r_1, \quad (4.46)$$

$$f_{R_2|NT}(r_2) = 1.9 - 1.8r_2, \quad (4.47)$$

$$f_{R_3|NT}(r_3) = 1.8 - 1.6r_3, \quad (4.48)$$

$$f_{R_4|NT}(r_4) = 1.7 - 1.4r_4, \text{ and} \quad (4.49)$$

$$f_{R_5|NT}(r_5) = 1.6 - 1.2r_5. \quad (4.50)$$

$$(4.51)$$

By (4.32) and (4.33), the joint probability density function, conditional on a threat item being passed through the 5 devices is

$$f_{\underline{R}|T}(\underline{r}) = (2r_1)(0.1 + 1.8r_2)(0.2 + 1.6r_3)(0.3 + 1.4r_4)(0.4 + 1.2r_5) \quad (4.52)$$

and the joint probability density function, conditional on a non-threat item being passed through the 5 devices is

$$f_{\underline{R}|T}(\underline{r}) = (2 - 2r_1)(1.9 - 1.8r_2)(1.8 - 1.6r_3)(1.7 - 1.4r_4)(1.6 - 1.2r_5). \quad (4.53)$$

	ε_{FC}		
	10^{-7}	10^{-8}	10^{-9}
u_1	0.6345	0.4310	0.2613
u_2	0.7047	0.5325	0.4098
u_3	0.7791	0.6185	0.5067
u_4	0.8651	0.7080	0.5991
u_5	0.9719	0.8124	0.7019
$P(\text{False Clear})$	1.00E-07	1.00E-08	1.00E-09
$P(\text{False Alarm})$	0.3371	0.6817	0.8721

Table 4.2: Analytical Results for Five-Device System

Chapter 5

Computational Results

The goal set forth in this thesis is to determine the minimum false alarm rate that can be achieved, given a particular multiple-device access control security system and a pre-specified false clear standard. To accomplish this goal, joint probability density functions, conditional on a threat item or a non-threat item being passed through each device, are established to describe the security system. Although either of the proposed grid estimation procedure could have been used to estimate these density functions, this thesis only utilizes the static grid estimation procedure. The response from each device is classified into N evenly spaced cells, so that the $(0, 1]^d$ hypercube is discretized into $n = N^d$ mutually exclusive hypercubes. Each of these hypercubes is represented by a subregion in the set I . Then, independent, identically distributed (IID) data points from each of the conditional joint probability density functions are sampled in order to determine the probabilities of a threat item eliciting a response in each subregion and a non-threat item eliciting a response in each subregion. These probabilities are represented by the vectors, \underline{q}_1 and \underline{q}_2 , respectively. Finally, KNAPSACK heuristics may be applied to the FAFC Problem with these vectors, a minimum false alarm rate, and a predetermined probability of a threat item. Numerous replications of the data points are generated, so that the mean and standard deviation of the probabilities of a false clear and a false alarm can be calculated. These computational results, determined using the Greedy and DP algorithms, are presented for the two-device system and the five-device system.

5.1 Two Continuous-Response Devices

Monte Carlo simulation was used to sample 1,000,000 IID data points from each of the conditional joint probability functions, given in (4.30) and (4.31). Static grid sizes of five, ten, and twenty were used, resulting in 25, 100, and 400 subregions, respectively. Fifty replications of the data points were generated for each grid size. The probabilities of false alarm and false clear were tested using the Greedy and DP algorithms with the total allowable probability of false clear, ε_{FC} , equal to 10^{-7} , 10^{-8} , and 10^{-9} . The probability of a threat item was set to 10^{-6} .

As would be expected, a grid size of twenty gives the smallest average probability of false alarm, since it gives the finest discretization of the sample space. For each grid size, the DP algorithm gives a smaller average probability of false alarm than the Greedy algorithm. In addition, the ratio of the probability of false alarm obtained with the DP algorithm over the probability of false alarm obtained with the Greedy algorithm increases as ε_{FC} decreases. However, because of the limitations discussed in Section 3.5.3, the DP results are computationally challenging for grid sizes of twenty or larger. The increase in the average probability of false alarm from using the Greedy algorithm instead of the DP algorithm for a grid size of twenty is very small, 0.047 % for $\varepsilon_{FC} = 10^{-7}$, 0.113 % for $\varepsilon_{FC} = 10^{-8}$, and 0.169 % for $\varepsilon_{FC} = 10^{-9}$. It should be noted that the standard deviation was very small for all tests. Tables 5.1, 5.2, and 5.3 give the mean and standard deviation of $P(\text{False Clear})$ and $P(\text{False Alarm})$ for each level of ε_{FC} . Finally, Tables 5.4, 5.5, and 5.6 provide a comparison between the analytical results and the computational results using the Greedy algorithm with a grid size of twenty. The C program that generated all computational results for a grid size of twenty took approximately 4 hours to run on a Sun ULTRA 1 with 128 megabytes of RAM.

5.2 Five Continuous-Response Devices

Monte Carlo simulation was used to sample 10,000,000 IID data points from each of the conditional joint probability density functions, given in (4.52) and (4.53). A static grid size of five was used, resulting in 3125 subregions, and fifty replications of the data points were generated. Because of the exponential growth in the number of subregions, no larger grid sizes were used. For example, a grid size of ten would result in 100,000 subregions, creating a computationally difficult problem. The probabilities of false alarm and false clear were tested using the Greedy algorithm with the total allowable probability of false clear, ε_{FC} , equal to 10^{-7} , 10^{-8} , and 10^{-9} . The DP algorithm was not used, because of the

	P(FC)		P(FA)	
	AVG.	STD. DEV.	AVG.	STD. DEV.
Grid = 5				
Greedy	8.800E-08	7.458E-11	0.30395	0.000103
DP	8.802E-08	7.387E-11	0.30393	0.000107
Grid = 10				
Greedy	9.960E-08	2.642E-10	0.26277	0.00227
DP	9.970E-08	2.140E-10	0.26271	0.00222
Grid = 20				
Greedy	9.989E-08	6.246E-11	0.25755	0.00014
DP	9.999E-08	6.623E-12	0.25742	0.00015

Table 5.1: Results for Two-Device System, $\varepsilon_{FC} = 10^{-7}$

	P(FC)		P(FA)	
	AVG.	STD. DEV.	AVG.	STD. DEV.
Grid = 5				
Greedy	6.395E-09	1.763E-11	0.76962	9.882E-05
DP	6.406E-09	1.958E-11	0.76957	9.420E-05
Grid = 10				
Greedy	9.901E-09	2.253E-11	0.65411	0.000116
DP	9.909E-09	2.334E-11	0.65410	0.000112
Grid = 20				
Greedy	9.886E-09	3.292E-11	0.63937	0.000553
DP	9.989E-09	6.910E-12	0.63865	0.000497

Table 5.2: Results for Two-Device System, $\varepsilon_{FC} = 10^{-8}$

	P(FC)		P(FA)	
	AVG.	STD. DEV.	AVG.	STD. DEV.
Grid = 5				
Greedy	0	0	0.99999	1.11E-16
DP	0	0	0.99999	1.11E-16
Grid = 10				
Greedy	7.010E-10	6.976E-12	0.89932	6.455E-05
DP	7.010E-10	6.976E-12	0.89932	6.455E-05
Grid = 20				
Greedy	9.431E-10	8.596E-12	0.86607	9.722E-05
DP	9.840E-10	8.489E-12	0.86460	0.000202

Table 5.3: Results for Two-Device System, $\varepsilon_{FC} = 10^{-9}$

	Analytical	Computational	Difference
P(FC)	1.000E-07	9.989E-08	1.100E-10
P(FA)	0.2574	0.2575	0.0001

Table 5.4: Two-Device System Comparison, Greedy Algorithm, Grid Size = 20, $\varepsilon_{FC} = 10^{-7}$

	Analytical	Computational	Difference
PFC	1.000E-08	9.886E-09	1.140E-10
PFA	0.6386	0.6394	0.0008

Table 5.5: Two-Device System Comparison, Greedy Algorithm, Grid Size = 20, $\varepsilon_{FC} = 10^{-8}$

	Analytical	Computational	Difference
PFC	1.000E-09	9.431E-10	5.690E-11
PFA	0.8646	0.8661	0.0015

Table 5.6: Two-Device System Comparison, Greedy Algorithm, Grid Size = 20, $\varepsilon_{FC} = 10^{-9}$

computational limitations discussed in Section 3.5.3. The probability of a threat item was set to 10^{-6} .

Table 5.7 gives the mean and standard deviation of $P(\text{False Clear})$ and $P(\text{False Alarm})$ for each level of ε_{FC} . It should be noted that the standard deviation was very small for all tests. Tables 5.8, 5.9, and 5.10 provide a comparison between the analytical results and the computational results. The analytical values of $P(\text{False Alarm})$, using the system response function (4.34), are very poor when compared to the computational results. When calculated computationally for $\varepsilon_{FC} = 10^{-7}$, the average probability of a false alarm decreases 53.9 % from the analytical value. For $\varepsilon_{FC} = 10^{-8}$, the decrease is 23.4 %, and for $\varepsilon_{FC} = 10^{-9}$, the decrease is 7.4 %. This is caused by the restricted selection of a system response function. Specifically, many system response functions are rendered infeasible because they are too difficult to integrate. Finding a better system response function, perhaps by using parametric equations, is a possible direction for future research. The C program that generated all computational results for a grid size of five took approximately 10 hours to run on a RS/6000 processor with an AIX operating system.

	ε_{FC}		
	10^{-7}	10^{-8}	10^{-9}
P(FC)			
AVG	9.998E-08	9.991E-09	9.970E-10
STD DEV	1.585E-11	5.293E-12	2.420E-12
P(FA)			
AVG	0.15510	0.52242	0.80762
STD DEV	0.00012	0.00039	0.00051

Table 5.7: Results for Five-Device System

	Analytical	Computational	Difference
P(FC)	1.000E-07	9.998E-08	2.000E-11
P(FA)	0.3371	0.1551	-0.1820

Table 5.8: Five-Device System Comparison, Greedy Algorithm, Grid Size = 5, $\varepsilon_{FC} = 10^{-7}$

	Analytical	Computational	Difference
P(FC)	1.000E-08	9.991E-09	9.000E-12
P(FA)	0.6817	0.5224	-0.1593

Table 5.9: Five-Device System Comparison, Greedy Algorithm, Grid Size = 5, $\varepsilon_{FC} = 10^{-8}$

	Analytical	Computational	Difference
P(FC)	1.000E-09	9.970E-10	3.000E-12
P(FA)	0.8721	0.8077	-0.0644

Table 5.10: Five-Device System Comparison, Greedy Algorithm, Grid Size = 5, $\varepsilon_{FC} = 10^{-9}$

Chapter 6

Conclusion

6.1 Summary

This thesis begins with a discussion of the need for better airport security systems, especially those that can optimally utilize and implement many different types of security technologies and procedures. Some of the key concepts used throughout this thesis, such as multiple-device security systems, and the false alarm, false clear (FAFC) problem, are introduced. Designers, operators, and users of multiple-device, access control security systems are challenged by the false alarm, false clear tradeoff. Given a particular access control security system, and a prespecified false clear standard, there is an optimal (minimal) false alarm rate that can be achieved. The objective presented in this thesis is to develop methods that can be used to determine this false alarm rate.

To provide an idea of previous work performed in the security arena, a literature review on airport security is presented. Security systems are categorized as either access control or perimeter security. Perimeter security systems are concerned with protecting an entire facility from external intruders or internal escapees. Access control systems, on the other hand, are primarily concerned with *gate keeping*. The system that is evaluated in this thesis is an airport access control security system, where luggage and passengers are screened to prevent the introduction of firearms or explosives onto aircraft. Significant developments in the literature on both perimeter security and access control are presented. A comparison between the FAFC problem and detection theory is also given. Because KNAPSACK heuristics are adapted to address the FAFC problem, a literature review on the variations and applications of the KNAPSACK problem is presented. The KNAPSACK problem has been used to solve many different classes of problems, including cryptography, broadband

communication, cutting stock, and resource allocation.

Once all preliminary information has been developed, a stochastic model for a d -device access control security systems is created and used to investigate the tradeoff between false alarms and false clears in this system. The research question addressed in this thesis is:

Given a particular access control security system and a prespecified false clear standard, what is the minimum false alarm rate that can be achieved?

The response from each device is normalized, without loss of generality, so that it lies in the $[0, 1]$ region. Since any item that passes through the security system must elicit a response from each of the d devices, the sample space is the $(0, 1]^d$ hypercube. The joint conditional probability density functions for the d -device security system responses, given a threat or a non-threat item, are then estimated. Two sampling procedures are proposed to estimate these density functions. The first procedure, called the *static grid estimation procedure*, uses an evenly spaced grid structure to create a d -dimensional histogram. The second procedure, called the *dynamic grid estimation procedure*, dynamically modifies the grid structure as additional sample points are collected, based on where the sample points cluster in the $(0, 1]^d$ hypercube. Notation and algorithms for both of the estimation procedures are presented.

Next, the concept of a *system response function* is introduced. This function combines the responses of all devices in the access control security system and returns an overall response, either a *system alarm* or a *system clear*. The probabilities of the system response function returning a correct or erroneous response are calculated from the joint probability density functions. Given that there is a prespecified probability of a threat item passing through the system, the probabilities of a false clear and a false alarm are also determined.

The FAFC problem, which minimizes the probability of a false alarm while holding the probability of a false clear to an upper bound, is then developed. It is formulated as a decision problem and proven to be NP-complete, by constructing a polynomial transformation to the KNAPSACK problem. Because of the similarity between the FAFC problem and the KNAPSACK problem, two heuristic algorithms used to address the KNAPSACK problem, the Greedy algorithm and the Dynamic Programming (DP) algorithm, are adapted to address the FAFC problem. In addition, the Reduce and Sort algorithm, which reduces the amount of computation required by the Greedy and DP algorithms, is presented. Computational limitations of the DP algorithm are presented.

The FAFC problem can be solved analytically for a system with prespecified conditional joint probability density functions, threat probability, and false clear standard. First,

linear marginal density functions are chosen for a system with d devices, where the device responses are independent of one another. Then system response functions are selected for the two-device system and the five-device system. Theorems are developed that will analytically determine the optimal probabilities of false clear and false alarm for each of these systems.

Computational results for the two-device system and the five-device system are also reported. The static grid estimation procedure is used to sample independent, identically distributed (IID) data points from each of the conditional joint probability density functions so as to determine the probabilities of a threat item eliciting a response in each subregion and a non-threat item eliciting a response in each subregion. The Greedy and DP algorithms are then used to calculate the minimum false alarm probability, given that the false clear probability is held to the prespecified false clear standard. Fifty replications of the data points are generated, so that the mean and standard deviation of the probabilities of a false clear and a false alarm are calculated. The computational results are compared to the analytical results for both systems.

6.2 Future Research

This research methodology is not limited to aviation security. It is applicable in several other fields of study, including military, health care systems, and industrial quality control. Military applications include automatic target recognition, detection, and identification systems, as well as unexploded mine detection and recognition systems. Health care applications include the diagnosis of cancer and other diseases. An industrial quality control application would be the problem of minimizing false alarms and false clears in the detection of defective items.

In the area of aviation security, there are many directions in which this research can be extended. One potential area involves grid estimation procedures. All computational tests performed in this thesis used security data that was simulated with the static grid estimation procedure. Future work would include the dynamic grid estimation procedure. Utilizing this procedure will involve determining whether all subregions should be dynamic. A hybrid grid, that incorporates both a static and a dynamic structure, is a possibility. For instance, the dynamic grid structure could be exploited for the subregions that are close to the division between the clear and alarm regions. This would involve determining which subregions to include in this critical subregion.

Other future research could develop computational examples for multiple-device access se-

curity systems with dependent devices. Although all of the examples illustrated in this thesis involve independent devices, the methodology presented can easily handle dependency. If appropriate conditional marginal probability density functions are developed, either sampling procedure can be used to estimate these functions and create a set of subregions. As in all other examples, the sampling procedure will estimate the probability of a threat item eliciting a response in each subregion and the probability of a non-threat item eliciting a response in each subregion. KNAPSACK heuristics can then be applied and computational results can be generated.

Another area of research focuses on creating better analytical solutions, by choosing more appropriate system response function forms. This effort is needed in the five-device example, where the analytical results are extremely poor compared to the computational results. The challenge is to choose system response functions that are computationally feasible, given the integration that is necessary to determine the analytical results. Parametric equations, based on a relationship between the responses from each of the five devices, may be useful in doing this.

Finally, a great opportunity for future research is the development of a multiple-response system. Rather than simply giving a clear or an alarm, this system can return one of several possible responses, dependent on the level of threat posed. For instance, a system could return a clear, an order to question the passenger, a request to match the passenger with his or her baggage, an order to search the passenger, or an alarm. This type of problem may be solved by applying KNAPSACK heuristics to responses in each of these subregions.

As more and more people travel by air, the FAA and the airline industry must strive to develop better security technology. The creation of a multiple-response system would significantly impact the entire aviation security industry. By correctly determining the level of threat posed by certain passengers and baggage, customer satisfaction could be improved, flight delays could be reduced, and lives could be saved.

Bibliography

- [1] L. Bodin and A. Kashani. The zone hopping problem. *Computers and Operations Research*, 18:75–86, 1991.
- [2] R. G. Bradley. Performance estimates for personnel access control systems. In *1981 IEEE Carnahan Conference on Crime Countermeasures*, pages 23–27, Lexington, Kentucky, 1981.
- [3] S.Y. Chiu, L. Lu, and L. A. Cox. Optimal access control for broadband services: stochastic knapsack with advance information. *European Journal of Operational Research*, 89:127–134, 1996.
- [4] B. Chor and R. L. Rivest. Knapsack-type public key cryptosystem based on arithmetic in finite fields. *IEEE Transactions on Information Theory*, 34:901–909, 1988.
- [5] A. H. Christer. Modelling the quality of automatic quality checks. *Journal of the Operational Research Society*, 45:806–816, 1994.
- [6] G. B. Dantzig. Discrete variable extremum problems. *Operations Research*, 5:266–277, 1957.
- [7] C. F. Davis, L. C. Chan, and T. D. Wormington. Measures of effectiveness for physical security system assessment. In *1982 IEEE Carnahan Conference on Crime Countermeasures*, pages 177–185, Lexington, Kentucky, 1982.
- [8] G. H. Dessent. Prison perimeter cost-effectiveness. *Journal of the Operational Research Society*, 38:975–980, 1987.
- [9] L. R. Doyon. Stochastic modeling of facility security systems for analytical solutions. *Computers and Industrial Engineering*, 5:127–138, 1981.
- [10] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

- [11] A. Gavious and Z. Rosberg. A restricted complete sharing policy for a stochastic knapsack problem in b-isdn. *IEEE Transactions on Communications*, 42:2375–2379, 1994.
- [12] R. R. Gilliam. An application of queueing theory to airport passenger security screening. *Interfaces*, 9:117–123, 1979.
- [13] R. Gilmore and R.E. Gomory. A linear programming approach to cutting stock problem. *Operations Research*, 11:863–888, 1961.
- [14] M. E. Gleeson and J. R. Ottensman. A decision support system for acquisitions budgeting in public libraries. *Interfaces*, 24:107–117, 1994.
- [15] S. G. Hahn. On the optimal cutting of defective sheets. *Operations Research*, 16:1100–1114, 1968.
- [16] M. I. Henig. Risk criteria in a stochastic knapsack problem. *Operations Research*, 38:820–825, 1990.
- [17] A. R. Hunt and J. A. Vanderslice. Detection and assesment requirements definition for the small icbm physical security system. In *1985 IEEE Carnahan Conference on Crime Countermeasures*, pages 93–100, Lexington, Kentucky, 1985.
- [18] J. Jan and S. Wang. A dynamic access control scheme based upon the knapsack problem. *Computers and Mathematics with Applications*, 26:75–86, 1993.
- [19] J. E. Kobza and S. H. Jacobson. Addressing the dependency problem in access security system architecture design. *Risk Analysis*, 16:801–812, 1996.
- [20] J. E. Kobza and S. H. Jacobson. Probability models for access security system architectures. *Journal of the Operational Research Society*, 48:255–263, 1997.
- [21] O. B. G. Madsen. Glass cutting in a small firm. *Mathematical Programming*, 17:85–90, 1979.
- [22] L. O. Malotky. Introduction to the sixth international civil aviation security conference. *Journal of Testing and Evaluation*, 22:235–237, 1994.
- [23] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, 1990.
- [24] H. F. Martz and M. E. Johnson. Risk analysis of terrorist attacks. *Risk Analysis*, 7:35–46, 1987.

- [25] R. C. Merkle and M. E. Hellman. Hiding information and signatures in trapdoor knapsack. *IEEE Transactions on Information Theory*, IT-24:525–530, 1978.
- [26] F. Peyron, L. R. Doyon, and J. Hanse. Computer analysis of security systems. In *1983 International Carnahan Conference on Crime Countermeasures*, pages 23–29, Zurich, Switzerland, 1983.
- [27] P. A. Polski. International aviation security research and development. *Journal of Testing and Evaluation*, 22:267–274, 1994.
- [28] E. Y. Rodin and D. Geist. Flight and fire control with logic programming. *Computers & Mathematics with Applications*, 20:15–27, 1990.
- [29] K.W. Ross and D. Tsang. The stochastic knapsack problem. *IEEE Transactions on Communications*, 37:740–747, 1989.
- [30] A. Seth. Wastage reduction in wood cutting. *Opsearch*, 24:94–105, 1987.
- [31] A. Shamir. A polynomial-time algorithm for breaking the basic merkle-hellman cryptosystem. *IEEE Transactions on Information Theory*, IT-30:699–704, 1984.
- [32] S. K. Stein. *Calculus and Analytic Geometry*. McGraw-Hill Book Company, 1987.
- [33] C. G. Tarr. Clasp: A computerized aid to cost effective perimeter security. In *1992 International Carnahan Conference on Crime Countermeasures*, pages 164–168, Zurich, Switzerland, 1992.
- [34] P. Toth. Dynamic programming for the zero-one knapsack problem. *Computing*, 25:29–45, 1980.
- [35] H. L. Van Trees. *Classical Detection and Estimation Theory*. John Wiley & Sons, 1968.

Appendix A

C Code for Computational Evaluation

```
/******  
/*The following is code for False Alarm, False Clear research problem */  
/*This code applies to the two device example, but is easily modified */  
/*for the five-device example */  
/*Programmer: Amy Simms */  
/******  
  
#include <stdio.h>  
#include <math.h>  
#include <stdlib.h>  
  
#define M 100000000 /*number of samples*/  
#define m 50 /*grid size*/  
#define size 2500 /*size of arrays - m^p*/  
#define PT 0.000001 /*probabilty of a Threat*/  
#define p 2 /*number of devices*/  
#define sbnd 10000 /*bound for dynamic programming arrays*/  
#define sample 50 /*sample size*/  
  
/*if vsize=0, initialize V to 1*/  
void generate(float Q1[size], float Q2[size], int locate[size][p]);  
void heapsort(float Q1[size], float Q2[size], int locate[size][p]);  
void reheapdown(float Q1[size], float Q2[size], int locate[size][p],
```

```

        int root, int bottom);
void swapf(float *element1, float *element2);
void swapi(int *element1, int *element2);
float greedy(float PFC, float Q1[size], float Q2[size],
             int X[3][2][size], int locate[size][p], int fcnt,
             float *zt);
float dynamic(float PFC, float Q1[size], float Q2[size],
             int X[3][2][size], int locate[size][p], int fcnt,
             float *zt);
void rec(double W1[sbnd], double P1[sbnd], double X1[sbnd], long *s,
        double *b, float q1, float q2, int sigma);
void output(float z[3][2][sample], float zt[3][2][sample],
           int locate[size][p], int X[3][2][size]);

main()
{
    /*define variables*/
    float z[3][2][sample];    /*sum of non-threat responses in CLEAR*/
    float zt[3][2][sample];   /*sum of threat responses in CLEAR*/
    int X[3][2][size] = {0}; /*accept indicator*/
    int locate[size][p];      /*original location*/
    float Q1[size];           /*threat responses */
    float Q2[size];           /*non-threat responses*/
    float PFC;                /*upper bound on P(False Clear)*/
    int cnt;
    int seed;

    for (seed=0; seed< sample; seed++)
    {

        PFC=0.0000001;
        srand48(seed+1);
        generate(Q1,Q2,locate);
        heapsort(Q1,Q2,locate);

        for (cnt=0; cnt < 3; cnt++)
        {
            zt[cnt][0][seed]=0;
            zt[cnt][1][seed]=0;
            z[cnt][0][seed] =

```

```
        greedy(PFC,Q1,Q2,X,locate,cnt,&zt[cnt][0][seed]);
    z[cnt][1][seed] =
        dynamic(PFC,Q1,Q2,X,locate,cnt,&zt[cnt][1][seed]);
    PFC=PFC/10;
}

}
output(z,zt,locate,X);
return;
}
```

```

/*****
/*Function: generate
/*This function creates randomly generated data to be used in the
/*knapsack algorithm. Note that the data read in IS NOT the
/*probabilistic values q1 and q2. Since many knapsack algorithms are
/*integer programming algorithms, the data read is the sum of non-threat
/*and threat responses in each subregion. In all algorithms, the limit
/*on q1, that is PFC/PT must be multiplied by M.
/*Once the knapsack algorithm is complete, all values must be divided by
/*M to get the corresponding probabilities. When z (sum of q2 in CLEAR
/*region) is divided by M, it represents P{NA|NT}. To determine
/*P{TC}=P{NA and NT}, multiply z by P{NT}
*****/

```

```

void generate(float Q1[size],float Q2[size],int locate[size][p])
{
    int response1[m][m]={0}; /*original storage grid*/
    int response2[m][m]={0};
    int cnt,cnt1,cnt2; /*sample counter*/
    int devcnt; /*device counter*/
    int devresp[m]; /*stores responses of each device*/
    double u, x; /*random numbers*/
    float temp;

    for (cnt=1; cnt<=M; cnt++)
    {
        for (cnt2=0; cnt2<2; cnt2++)
        {
            for (devcnt=0; devcnt<p; devcnt++)
            {
                u = drand48(); /*use method of inversion*/
                if (cnt2==0) /*threat item*/
                { x = sqrt(u); }
                else /*non-threat item*/
                { x = 1-sqrt(1-u); }

                temp = m*x;
                if (temp < 1) /*round down to find grid square*/
                { devresp[devcnt] = 0; }
                else if (temp < 2)

```

```
{    devresp[devcnt] = 1; }
else if (temp < 3)
{    devresp[devcnt] = 2; }
else if (temp < 4)
{    devresp[devcnt] = 3; }
else if (temp < 5)
{    devresp[devcnt] = 4; }
else if (temp < 6)
{    devresp[devcnt] = 5; }
else if (temp < 7)
{    devresp[devcnt] = 6; }
else if (temp < 8)
{    devresp[devcnt] = 7; }
else if (temp < 9)
{    devresp[devcnt] = 8; }
else if (temp < 10)
{    devresp[devcnt] = 9; }
else if (temp < 11)
{    devresp[devcnt] = 10; }
else if (temp < 12)
{    devresp[devcnt] = 11; }
else if (temp < 13)
{    devresp[devcnt] = 12; }
else if (temp < 14)
{    devresp[devcnt] = 13; }
else if (temp < 15)
{    devresp[devcnt] = 14; }
else if (temp < 16)
{    devresp[devcnt] = 15; }
else if (temp < 17)
{    devresp[devcnt] = 16; }
else if (temp < 18)
{    devresp[devcnt] = 17; }
else if (temp < 19)
{    devresp[devcnt] = 18; }
else
{    devresp[devcnt] = 19; }
}
if (cnt2==0)
{    response1[devresp[0]][devresp[1]]++; }
```

```
        else
        {   response2[devresp[0]][devresp[1]]++; }
    }
}

/*convert to one-dimensional arrays*/
cnt=0;
for (cnt1=0; cnt1<m; cnt1++)
{   for (cnt2=0; cnt2<m; cnt2++)
    {
        Q1[cnt]=response1[cnt1][cnt2];
        Q2[cnt]=response2[cnt1][cnt2];
        locate[cnt][0]=cnt1;
        locate[cnt][1]=cnt2;
        cnt++;
    }
}

return;
}
```

```

/*****
/*Function:  heapsort
/*This function uses a heap structure to sort the arrays Q1, Q2, and
/*locate.
/*A heap is an array that is ordered in a tree-like structure, where the
/* first entry in the array is the root and the second and third entries
/* are the children of that root.  Entries 4 and 5 are the children of
/* the second entry, entries 6 and 7 are the children of the third entry
/* and so on.  Each entry may have at most 2 children.  Futhermore,
/* (for descending order), each child must be greater than its parent.
/*Because of the order property of a heap, the minimum value in the
/* array must be the root node.  The general approach of the function
/* Heapsort is:
/* 1. Arrange the original array elements in a heap
/* 2. Remove the root (minimum) element from the heap and put it in
/* its appropriate position in the array.
/* 3. Arrange the remaining elements in a heap - this puts the next-
/* smallest element in the root position.
/* 4. Repeat from step 2 until there are no more elements in the heap
/*This version of heapsort uses Q2/Q1 as the measure of size
/*Efficiency of Heapsort:  Heapsort is  $O(n \log n)$ 
/*****

```

```

void heapsort(float Q1[size], float Q2[size], int locate[size][p])
{
    int index;

    for (index=size/2-1; index>=0; index--)
    { reheapdown(Q1,Q2,locate,index,size-1); } /*build original heap*/

    for (index=size-1; index>=1; index--)
    {
        swapf(Q2+0, Q2+index); /*swap the root value with the */
        swapf(Q1+0, Q1+index); /*swap Q1, Q2, locate*/
        swapi(&locate[0][0], &locate[index][0]);
        swapi(&locate[0][1], &locate[index][1]);
        /*reheap the remaining part of array*/
        reheapdown(Q1,Q2,locate,0,index-1);
    }
    return; }

```

```

/*****
/*Function:  reheardown
/*This function creates a heap, as described above, from an array
/* Order is restored, starting at the root, by the size measure Q2/Q1
/*****

void reheardown(float Q1[size], float Q2[size], int locate[size][p],
                int root, int bottom)
{
    int minchild;           /*index of child with smaller value*/
    int rightchild;        /*index of right child node*/
    int leftchild;         /*index of left child node*/

    leftchild = root*2+1;
    rightchild = root*2 + 2;

    if (leftchild <= bottom) /*root in not a leaf*/
    {
        if (leftchild == bottom) /*there is only one child node*/
            { minchild = leftchild; }
        else /*pick the smaller of the two children*/
        {
            if ((Q2[leftchild]/Q1[leftchild]) <
                (Q2[rightchild]/Q1[rightchild]))
                { minchild = leftchild; }
            else
                { minchild = rightchild; }
        }
    }

    if ((Q2[root]/Q1[root]) > (Q2[minchild]/Q1[minchild]))
    { /*not in order*/
        swapf(Q2+root, Q2+minchild); /*swap and reheap*/
        swapf(Q1+root, Q1+minchild);
        swapi(&locate[root][0], &locate[minchild][0]);
        swapi(&locate[root][1], &locate[minchild][1]);
        reheardown(Q1,Q2,locate,minchild,bottom);
    }
}
return;
}

```

```
/******  
/*Function: swapi  
/*This function swaps two integer elements by address  
/******
```

```
void swapi(int *element1, int *element2)  
{  
    int temp;  
  
    temp = *element1;  
    *element1 = *element2;  
    *element2 = temp;  
    return;  
  
}
```

```
/******  
/*Function: swapf  
/*This function swaps two float elements by address  
/******
```

```
void swapf(float *element1, float *element2)  
{  
    float temp;  
  
    temp = *element1;  
    *element1 = *element2;  
    *element2 = temp;  
    return;  
  
}
```

```

/*****
/*Function: greedy
/*Reference: Knapsack Problems: Algorithms & Computer Implementations,
/*           Martello & Toth, p. 27-29
/*This function applies the greedy algorithm to a knapsack problem
/*It returns z, the total accumulated true clear probability
/*It also modifies the X vector to reflect which subregions are rejected*/
/* accepted to the Clear region
/*****

float greedy(float PFC, float Q1[size], float Q2[size],
            int X[3][2][size], int locate[size][p], int fcnt,
            float *zt)
{
    float sigma; /*accumulated threat responses in clear region*/
    float z;     /*accumulated non-threat responses in clear region*/
    int jstar;   /*index of subregion with largest TC probability*/
    int cnt;     /*counter*/
    int Xthis[size]={0};
    int index;

    sigma = PFC/PT*M; /*initialization*/
    z = 0;
    jstar = 0;

    for (cnt=0; cnt<size; cnt++)
    {
        index=m*locate[cnt][0]+locate[cnt][1];
        if (Q1[cnt] > sigma)
        { Xthis[cnt]=0; }
        else
        {
            Xthis[cnt]=1;
            X[fcnt][0][index]++; /*accept subregion*/
            sigma = sigma - Q1[cnt];
            z = z + Q2[cnt];
            *zt = *zt + Q1[cnt];
        }

        if (Q2[cnt] > Q2[jstar])

```

```
        {      jstar = cnt;      }
    }

    if ((Q2[jstar] > z)&(Q1[jstar]<=sigma))
    {
        z = Q2[jstar];
        *zt = Q1[jstar];
        for (cnt=0; cnt<size; cnt++)
        {
            if (Xthis[cnt]==1)
            {      index=m*locate[cnt][0]+locate[cnt][1];
                X[fcnt][0][index]--; }
        }
        X[fcnt][0][jstar]++;
    }

    return z;
}
```

```

/*****
/*Function: dynamic
/*Reference: Knapsack Problems: Algorithms & Computer Implementations,
/*           Martello & Toth, p. 36-42
/*This function applies the dynamic programming algorithm to a knapsack
/* problem. It returns z, the total accumulated TC probability
/* It also modifies the X vector to show which subregions are rejected
/* or accepted to the Clear region
/*Dynamic Programming requires that all values are integers, so it is
/* passed the values, q1(i)*M, q2(i)*M, and sigma*M, which are integers
/*****

float dynamic(float PFC, float Q1[size], float Q2[size],
             int X[3][2][size], int locate[size][p], int fcnt,
             float *zt)
{
    int sigma = PFC/PT*M;      /*total threat responses in CLEAR region*/

    double W1[sbnd];          /*total weight of each state*/
    double P1[sbnd];          /*total profit of each state*/
    double X1[sbnd];          /*status - accept/deny - of each state*/

    long s=1;                 /*number of states at stage (m-1)*/
    double b=2;               /*2^(m-1), where m=stage*/
    int cnt,cnt1;             /*counter for stages*/
    double div;               /*divisor for decoding binary string X1*/
    double divrnd;
    int remain;               /*remainder for decoding binary string X1*/
    float z=0;                /*true clear probability returned*/
    int index;

    /*define values for stage 1*/
    W1[0]=0;
    P1[0]=0;
    X1[0]=0;
    W1[1]=Q1[0];
    P1[1]=Q2[0];
    X1[1]=1;

    /*define values for stages 2 to qcount*/

```

```
for (cnt=2; cnt<=size; cnt++)
{
    rec(W1,P1,X1,&s,&b,Q1[cnt-1],Q2[cnt-1],sigma);
}

/*assign optimal true clear probability*/
z = P1[s];
*zt = W1[s];

/*decode X1[s] for optimal subregion status*/
/*find the binary number representation for X1[s]*/

divrnd = floor(X1[s]/2);
div = X1[s]/2;
cnt=0;
index=m*locate[cnt][0]+locate[cnt][1];
if (div > divrnd)
{    X[fcnt][1][index]++;    }

while (divrnd > 0)
{
    cnt++;
    index=m*locate[cnt][0]+locate[cnt][1];
    div = divrnd/2;
    divrnd=floor(divrnd/2);
    if (div > divrnd)
    {    X[fcnt][1][index]++;    }
}

return z;
}
```

```

/*****
/*Function: rec
/*Reference: Knapsack Problems: Algorithms & Computer Implementations,
/*           Martello & Toth, p. 36-42
/*This function determines the undominated states of a stage - it is
/* called by the dynamic programming algorithm
/*It modifies the variables W1,P1,X1,s, and b
*****/

void rec(double W1[sbnd], double P1[sbnd], double X1[sbnd], long *s,
        double *b, float q1, float q2, int sigma)
{
    long i=0;           /*counts states scanned*/
    long k=0;           /*counts states stored*/
    long h=1;
    double y=q1;
    double W2[sbnd];   /*accumulated threat responses of new stage*/
    double P2[sbnd];   /*accumulated non-threat responses of new stage*/
    double X2[sbnd];   /*subregion status of new stage*/
    double min;
    double prof;
    double x;

    W1[*s+1] = M+1;     /*will be larger than any q1(i)*/
    W2[0] = 0;
    P2[0] = 0;
    X2[0] = 0;
    min = (y < W1[h]) ? y : W1[h];    /*minimum of y and W1[h]*/

    while (min <= sigma)
    {
        if (W1[h] <= y)           /*define next state*/
        {
            prof = P1[h];
            x = X1[h];
            if (W1[h] == y)
            {
                if ((P1[i]+q2) > prof)
                {
                    prof = P1[i] + q2;
                }
            }
        }
    }
}

```

```

        x = X1[i]+ *b;
    }
    i = i + 1;
    y = W1[i] + q1;
}
if (prof > P2[k]) /*store next state, if not dominated*/
{
    k=k+1;
    W2[k]=W1[h];
    P2[k]=prof;
    X2[k]=x;
}
h = h + 1;

} /*end if*/
else /*store the new state, if not dominated*/
{
    if (P1[i]+q2 > P2[k])
    {
        k = k + 1;
        W2[k] = y;
        P2[k] = P1[i] + q2;
        X2[k] = X1[i] + *b;
    }
    i = i + 1;
    y = W1[i] + q1;
}
min = (y < W1[h]) ? y : W1[h]; /*minimum of y and W1[h]*/

} /*end while*/

*s = k;
*b = 2>(*b);
for (i=0; i<=*s; i++)
{
    P1[i] = P2[i];
    W1[i] = W2[i];
    X1[i] = X2[i];
}
}

```

```

/*****
/*Function: output
/*This function computes and outputs to file the subregions contained in*/
/* the CLEAR and ALARM regions, P{FC}, P{FA}
/*MUST BE edited for the appropriate number of devices
*****/

void output(float z[3][2][sample],float zt[3][2][sample],
            int locate[size][p], int X[3][2][size])
{

    int cnt1,cnt2,cnt3,index;
    float accumz, accumzt;
    float PFC;
    FILE *out_ptr;          /*pointer to output file*/
    float xtemp;

    out_ptr = fopen("grid50.out", "w");

    fprintf(out_ptr,
            "The following results apply to a grid of size 50\n\n");

    PFC=0.0000001;
    for (cnt1=0; cnt1<3; cnt1++)
    {
        fprintf(out_ptr,
            "\nThe results for allowable P{FC} = %e:\n",PFC);

        fprintf(out_ptr,"Greedy Algorithm:\n\n");
        fprintf(out_ptr,"Run P{FC} P{FA}\n");
        accumz=0; accumzt=0;
        for (cnt2=0; cnt2<sample; cnt2++)
        {
            zt[cnt1][0][cnt2]=zt[cnt1][0][cnt2]/M*PT;
            accumzt=accumzt+zt[cnt1][0][cnt2];
            z[cnt1][0][cnt2]=(1-z[cnt1][0][cnt2]/M)*(1-PT);
            accumz=accumz+z[cnt1][0][cnt2];
            fprintf(out_ptr,"%d %.15f %.15f\n",
                    cnt2+1,zt[cnt1][0][cnt2],z[cnt1][0][cnt2]);
        }
    }
}

```

```

fprintf(out_ptr,
        "AVG   %.15f   %.15f\n",accumzt/sample,accumz/sample);
fprintf(out_ptr,"\nThe probability of each grid square
        appearing in the CLEAR region is:\n");
fprintf(out_ptr,"Loc1  Loc2  Prob\n");
for (cnt2=0; cnt2<m; cnt2++)
{   for (cnt3=0; cnt3<m; cnt3++)
    {
        index = m*cnt2+cnt3;
        xtemp=X[cnt1][0][index];
        fprintf(out_ptr,
                "%d   %d   %f\n",cnt2,cnt3,xtemp/sample);
    }
}

fprintf(out_ptr,"\nDynamic Programming Algorithm:\n\n");
fprintf(out_ptr,"Run   P{FC}   P{FA}\n");
accumz=0; accumzt=0;
for (cnt2=0; cnt2<sample; cnt2++)
{
    zt[cnt1][1][cnt2]=zt[cnt1][1][cnt2]/M*PT;
    accumzt=accumzt+zt[cnt1][1][cnt2];
    z[cnt1][1][cnt2]=(1-z[cnt1][1][cnt2]/M)*(1-PT);
    accumz=accumz+z[cnt1][1][cnt2];
    fprintf(out_ptr,"%d   %.15f   %.15f\n",
            cnt2+1,zt[cnt1][1][cnt2],z[cnt1][1][cnt2]);
}
fprintf(out_ptr,
        "AVG   %.15f   %.15f\n",accumzt/sample,accumz/sample);
fprintf(out_ptr,"\nThe probability of each grid square is:\n");
fprintf(out_ptr,"Loc1  Loc2  Prob\n");
for (cnt2=0; cnt2<m; cnt2++)
{   for (cnt3=0; cnt3<m; cnt3++)
    {
        index=m*cnt2+cnt3;
        xtemp=X[cnt1][1][index];
        fprintf(out_ptr,
                "%d   %d   %f\n",cnt2,cnt3,xtemp/sample);
    }
}

```

```
        PFC=PFC/10;  
    }  
  
    return;  
}
```

VITA

Amy E. Simms was born on April 10, 1973 in Fredericksburg, Virginia. She graduated Cum Laude with a Bachelor of Science degree in Mathematics from Virginia Polytechnic and State University in 1995. While an undergraduate at VPI & SU, Amy was a member of the Phi Sigma Pi National Honor Fraternity, the Pi Mu Epsilon Mathematics Honor Society, and the Golden Key National Honor Society. Amy also participated in the Cooperative Education Program, working alternating semesters at TRW in McLean, Virginia. In 1996, Amy began work towards a Masters of Science degree in Operations Research at Virginia Polytechnic Institute and State University. She is a member of the Institute for Operations Research and the Management Sciences. In March 1997, Amy was hired by SolutionWorks, a division of Andersen Consulting, in St. Petersburg, Florida. She will begin work as an Analyst in July 1997.

Permanent Address:

10263 Gandy Blvd. N. # 614
St. Petersburg, Florida 33702