The Thesis Committee for Nathan Andrew Stewart
Certifies that this is the approved version of the following Thesis:

Decision Analysis Perspectives on Sequential Testing

APPROVED BY
SUPERVISING COMMITTEE:

James Eric Bickel, Supervisor

John Hasenbein

Decision Analysis Perspectives on Sequential Testing

by

Nathan Andrew Stewart

Thesis

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

December 2019

Dedication

Dedicated to Moniqua Jadielle Ray.

Acknowledgements

I wish to thank the countless number of groups and individuals that made this research possible. I would first and foremost like to thank Sandia National Laboratories, who provided both funding and inspiration. I would also like to thank my peers and advisors at The University of Texas at Austin, who offered support and guidance.

Abstract

Decision Analysis Perspectives on Sequential Testing

Nathan Andrew Stewart, M.S.E.

The University of Texas at Austin, 2019

Supervisor: James Eric Bickel

Expanding from proof-load testing, this paper utilizes a decision analysis framework to determine adequate bounds for how much one should be willing to pay to conduct testing when the load distribution of a device is uncertain. Optimal testing policy strategies are constructed and evaluated when sequentially testing a population of devices for given prior beliefs on the intrinsic toughness of the device. Heuristic extensions are analyzed to reduce computational time for practical use.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

There is often a need to determine the behavior of a population of devices or components with uncertain properties. To determine these properties, tests are conducted to acquire information such that decisions can be made regarding the population, such as approving the devices to be sold or utilized in some other function. An example of such an event would be steel rebar being manufactured on a production line. It is standard to test a series of these rebars to determine how much load each individual rebar could withstand before determing whether one should release the lot for use. Given that these rebars are used in the construction of buildings, it is imperative that we are confident in the population being able to withstand the desired load. In this section, we provide the underlying motivation for this research, the specific mechanics of the problem, and describe how this research compares to related work.

1.1 Problem Description

Consider a population of n devices, and there is a need to either approve the population for sale or utilization, or to reject them. For each device, there is an intrinsic threshold capacity under which the device can withstand some load, such

as a computer power supply withstanding a particular voltage. Assuming that the devices share a similar integrity upon construction with some natural variation due to product tolerances, we assign a prior distribution for the load that the device can withstand. There is an expectation of the device to withstand a given load to perform its function properly. Let X denote the uncertain load capacity of a specified device that has a requirement to meet a given level, τ. In other words, X is the intrinsic toughness of the device — any load subjected to a device that is greater than its X would result in device failure. Figure 2.1 depicts an example of such a distribution. We denote the units of this as a generic unit, decibels (dB). We designate this device toughness as the environmental threshold capacity, or ETC.

With a testing schema, we should be able to gain a better understanding of the load capacity distribution, thus making our decision of whether to approve or reject the entire population more accurate, and potentially decrease expected cost. In this paper, we will analyze the case of one-shot devices under a sequential series of tests. One-shot testing, for our purposes, is when we test a device at a given load, and only determine whether or not the device fails. That is, we determine if the intrinsic toughness for this particular device was able to withstand the tested load. Furthermore, we can only test these devices once, as we are assuming that the integrity of the device is compromised after being tested. In this respect, we will be selecting a new device from the population every time we conduct testing. The goal is to determine how much testing to conduct and at what test levels to conduct the tests by comparing the expected cost of simply approving the batch without any testing and the information we expect to gain from testing. The goal

of this paper is to introduce a decision-analytic approach to designing test plans for one-shot devices, complementing existing approaches by explicitly considering cost and prior information in the test planning.

1.2 Literature Review

Statistical design of experiments is often used to design one-shot device test plans, often referred to as sensitivity testing [3, 7]. Statistical criteria are employed to determine the number of tests to conduct and the test levels. For instance, efficient estimation of the mean and standard deviation of the underlying threshold distribution is a common objective of sensitivity test design; reliability and quantile estimation is another common goal [12]. Overall sample sizes can be selected based on such statistical criteria. Many methods for selecting test-levels have been proposed [3, 7]; one of the most common is the Neyer sequential testing procedure, where test levels are sequentially selected using a D-optimality criteria [7]. Sensitivity test plan design remains an active area of research in statistics, including sequential design and Bayesian design of experiments [4, 2, 11].

Another common analog for this problem is that of proof-load testing in the field of civil engineering. There is typically a system of parts that must holistically withstand a load, such as a bridge sustaining a certain weight. One example is when a population of structures need to have a safety factor assigned to them without damaging the structure in testing — the result from this analysis showed that the conventional solution to this problem, that of testing roughly 1.5 times the load capacity only a handful of times, could be circumvented by testing a certain per-

centage of the population (in this case, 3%) with relatively small loads [6]. Given the results, updating one's beliefs on the lower tail of the threshold distribution can then be extrapolated to the safety factor of the structure. In more recent research, testing at lower proof loads numerous times can be both cheaper and equally informative compared to higher proof loads [1]. However, in all of these load testing schemata, the cost is merely mentioned rather than taken into account explicitly, with the optimal number of loads (or in our case, devices) to test is not explored fully. Furthermore, a decision framework is not utilized, as in many civil engineering cases, the structure itself needs to exist, and thus the only guiding principle is to determine information on the integrity of the structure as accurately as possible.

Research has been conducted that uses a similar Bayesian updating scheme and utilization of Markov decision processes to achieve optimal policies for sequential testing on generalized cost and reward structures at varying states, but not within the specific context of determining information regarding the intrinsic properties of devices while undergoing destructive testing [5]. We believe that our research is unique in that it considers cost when determining the physical qualities of a population of devices through testing.

Chapter 2

Sequential Testing as a Decision Problem

In this section, we detail the initial population decision whether to approve or reject the population, and then introduce testing as a decision. We then detail the Bayesian updating that occurs upon testing as well as the cost structure applied to the result of testing. With these components, we can begin to analyze the problem from a decision-analytic approach.

2.1 Initial Population Decision

Before we conduct any testing, our initial population decision is to decide whether we should approve or reject the population. Approving the population has an associated cost that is dependent on the number of devices that fail after approving the population. This cost structure will be explored fully later in this section. We can calculate the likelihood that we expect a device to fail as a function of our prior beliefs. We will denote this as $p = P(X \leq \tau)$, which represents the probability that the given device fails to meet the specified requirement.

Let $F(x \mid \theta)$ indicate a probability distribution for X with parameter(s), θ. Let $G(\theta)$ be defined as a prior distribution for θ, and let $F(x) = E_\theta[F(x \mid \theta)]$ denote the prior predictive distribution for X, where E_θ is the expected value operator

evaluated with respect to θ. Furthermore, let $f(x \mid \theta)$, $g(\theta)$, and $f(x)$ represent density functions for $F(x \mid \theta)$, $G(\theta)$, and $F(x)$, respectively. Using this nomenclature, we define $p_\theta = P(X \leq \tau \mid \theta) = F(\tau \mid \theta)$. Thus, uncertainty in p_θ results from uncertainty in θ.

Figures 2.1 and 2.2 are examples of a possible ETC and prior distributions respectively, $f(x \mid \theta)$ for some θ. The vertical black line in Figure 2.1 indicates an example of a load requirement τ. The area under the density function to the left of this level represents p_θ.

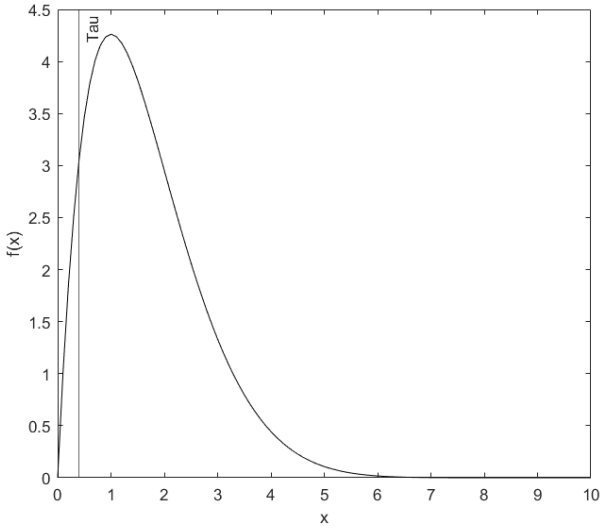

Figure 2.1: Example of Possible ETC Distribution

We use a linear cost function in this paper to determine the expected cost of approving the population dependent on the number of device failures, denoted by j. Let b be the associated cost per device failure. This cost can then be expressed by $c_j = j \cdot b$, where $j \geq 0$, $b > 0$. The cost incurred for j failures is a non-decreasing

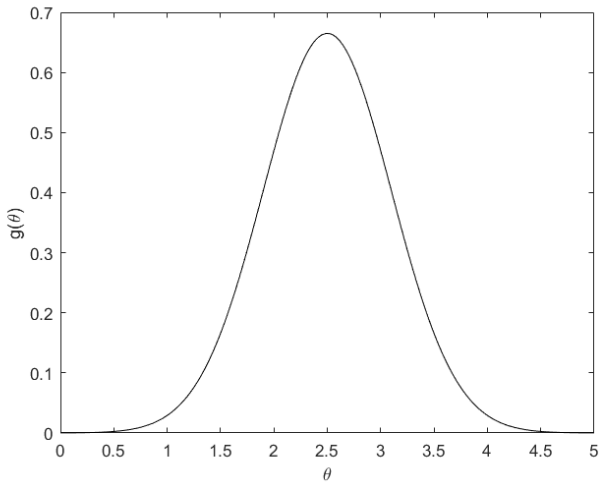

Figure 2.2: Example of Possible Prior Distribution on θ

linear function of j for $1 \leq j \leq n$, while no cost is incurred for observing zero failures. In this case, with a given θ, we can determine the expected number of failures j from the binomial distribution, where the probability of failure is p_θ, and the number of devices is the population n. The expected cost is then just the product of this linear cost and the expected number of failures:

$$\begin{aligned} E[C(p_\theta)] &= \sum_{j=0}^{n} \binom{n}{j} p_\theta^j (1-p_\theta)^{n-j} c_j \\ &= \sum_{j=0}^{n} \binom{n}{j} p_\theta^j (1-p_\theta)^{n-j} jb \\ &= bnp_\theta. \end{aligned}$$

Although we only consider a linear cost structure, any arbitrary cost structure could be applied dependent on the number of devices that fail.

If we reject the population, we will incur a predefined rejection cost of R. Therefore, our initial population decision is the minimum between our rejection cost and the expected cost of the failed devices if we approve the population. Figure 2.3 is the decision tree for this problem. The square represents a node where a decision is made between approving or rejecting the population, and the circle represents uncertainty in the number of devices that fail if we approve the population.

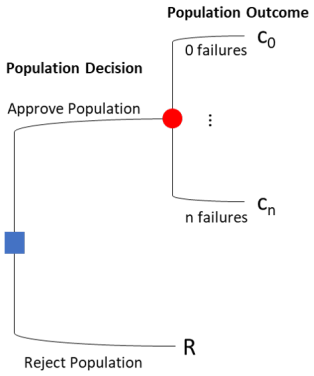

Figure 2.3: Decision Tree of Initial Population Decision

For example, suppose that we have a set of prior beliefs θ_0 with a $Normal(\mu = 5, \sigma = 3)$ distribution. The rejection cost is $R = 1$ and the failure cost is $b = 300$. Let $\tau = 1$ and our ETC distribution to be distributed as a $Lognormal(\theta_0, 1)$. From this, we can calculate the likelihood that a device selected randomly from the population will fail, which is:

$$\int_0^\tau \frac{1}{\sqrt{2\pi}\sigma x} e^{-(\log(x)-\mu)^2/2\sigma^2} dx = \int_0^\tau \frac{1}{\sqrt{2\pi}x} e^{-(\log(x)-\theta)^2/2} dx \approx 0.07.$$

The value of the optimal cost decision is therefore:

$$min(R, C(p_\theta)) = min(R, C(0.07)) = min(1, 19.7) = 1.$$

In this example, the expected cost to approve the population is 19.7. Therefore, it is optimal to reject the population and incur a cost of 1.

2.2 Adding Testing as a Decision

Now we add the capability of testing devices in order to minimize our expected cost. When testing a single device, we first choose a threshold t (in dB) at which to test the device (a decision), and then we observe whether the device fails or passes. More formally, for multiple test devices, let X_i denote the uncertain ETC for the i^{th} test device, and let t_i denote the chosen test threshold for this device. Therefore, an additional decision is made on which testing level to conduct if one decides to test. Let $q_{\theta,t} = P(X \leq t \mid \theta) = F(t \mid \theta)$ be the probability of failure for a device given a θ and load value t. This is a more generalized form of p_θ. We will only be considering testing levels from a finite set of permissible testing levels, which we denote by T.

The outcome for this test can be expressed as $\Delta_i = 1\{X \leq t_i\}$, where $1\{\}$ is an indicator function that takes on value 1 if true and 0 if false. Upon observing the result of an ETC test of a device or set of devices, we can update θ and subsequently update the prior distribution on the load capacity. We do this by applying a Bayesian update on the posterior distribution given the test result. Over time, our prior distribution should better reflect the ETC of the population of devices with

a higher degree of confidence. Given limitless testing, we could determine the intrinsic ETC behavior. For one-shot testing, the likelihood for the observed data $D = (\Delta_i, t_i)_i$ is:

$$l(\theta \mid D) = \prod F(t_i \mid \theta)^{\Delta_i} (1 - F(t_i \mid \theta))^{1-\Delta_i}.$$

Therefore, a Bayesian update of θ is:

$$p(\theta \mid D) = l(\theta \mid D) g(\theta) / f(D).$$

Figure 2.4 depicts the same initial prior distribution of the example before, $Normal(\mu = 5, \sigma = 3)$, and the resulting posterior distributions testing at $t = 3$.

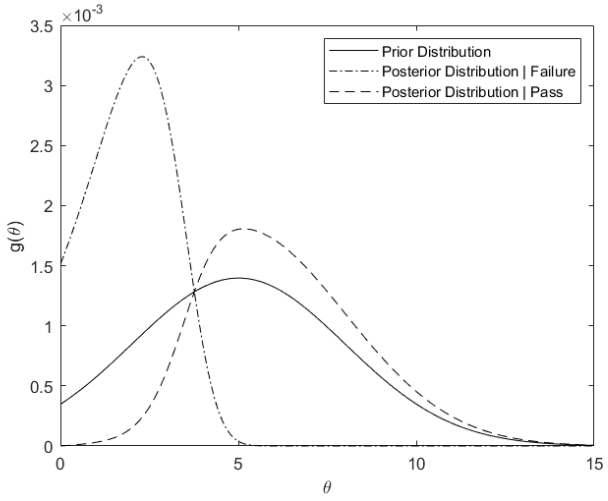

Figure 2.4: Example of Posterior Distributions After Observing Test Outcome

Testing incurs a cost of c for each device that is tested, which comprises the destruction of the device and resources expended to test. As such, the decision

maker must conduct testing intelligently in order to optimize overall costs. After each test, we will have the same set of three decisions: approve the population, reject the population, or test again. Figure 2.5 depicts the associated decision tree. The number of possible decisions at the end of the tree, given m number of tests and T testing levels, is $(2T)^m$.

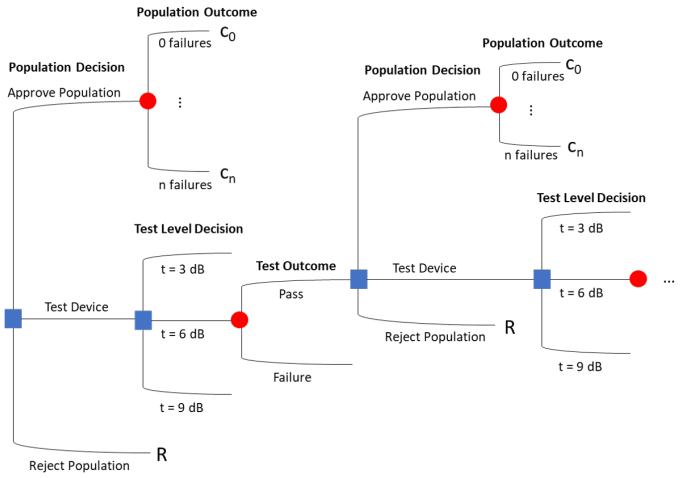

Figure 2.5: Decision Tree Including Sequential Testing

In practice, having a discretized number of testing levels is reasonable, as it makes the testing procedure easier to conduct. Having the number of possible tests m be finite is also reasonable, as there is a natural upper bound on the testing depth, which will be covered later in Section 3.2. Therefore, keeping the set of testing levels T and the maximum number of tests m finite, we can determine the optimal testing policy in a practical amount of time.

2.3 Selecting Initial Prior Beliefs

For our analysis, we want to have the flexibility to utilize any prior distribution when computing the optimal policy. For the remainder of the paper, we will be analyzing instances where the prior distribution on θ is of the form $g(\theta) = Normal(\mu_0, \sigma_0^2)$, and our ETC distribution is of the form $f(\theta) = Lognormal(\theta, \sigma^2)$, where σ is known. Our prior beliefs will represent the mean of the device distribution with a known variance. Note that, for a normally distributed prior belief on the mean, there will always be a nonzero sum of probabilistic mass that is below zero (assuming a non-zero variance). Therefore, when using the initial prior beliefs, we redistribute this mass such that all possible values for the mean are positive. For example, for a putative $Normal(\mu_0, \sigma_0^2)$ prior distribution, the normalized distribution for $x \in (0, \infty)$ is:

$$p(x) = \frac{\frac{1}{\sigma_0\sqrt{2\pi}} e^{-(x-\mu_0)^2 \big/ 2\sigma_0^2}}{\int_{-\infty}^{0} \frac{1}{\sigma_0\sqrt{2\pi}} e^{-(x-\mu_0)^2 \big/ 2\sigma_0^2} dx}.$$

Chapter 3

Formulation

To formally and systematically approach solving this problem at scale, we use a Markov decision process (MDP) to determine the optimal policy and its cost. In this section, we describe the formulation of this framework, derive an upper bound on the number of tests to conduct, and provide an example on how the solution methodology can be applied.

3.1 Sequential Testing Policy as a Markov Decision Process

To determine the optimal testing policy, we utilize a MDP framework. We can express the total cost of the policy as:

$$V = E[\sum_{k=0}^{m} g_k(x_k, u_k, w_k)].$$

g_k represents the cost function at time k, for $k \geq 0$, x_k represents the state of testing outcomes, u_k represents the decision made, and w_k is the uncertainty given x_k and u_k, or the likelihood that a device fails given a testing level and the current beliefs. Since m denotes the maximum number of tests, u_{m+1} is the final decision after the m^{th} test. We restrict the controls, or alternatives, on the decisions at time k to $C_k = (Approve, Reject) \cup T$ for $k < m$, where T represents the set of testing

levels. $C_m = (Approve, Reject)$, since we restrict the maximum number of tests to m, we make the final approval/rejection decision at the m^{th} test.

For our model, since there is a testing cost of c, we incur that cost after each testing decision. Therefore, the cost of each state given the decision of approving the population is $g_k(x_k, Approve, w_k) = C(p_\theta)$, or the expected cost of approving the population given the current beliefs. $g_k(x_k, Reject, w_k) = R$ is the cost of the decision if the population is rejected, and $g_k(x_k, t \in T, w_k) = c$ is the cost if we decide to test at level t.

x_k is the state that contains the set of testing levels and their observed outcomes as well as our initial prior beliefs. For example, if we test at level $t = 2$ and fail, then at $t = 1$ and pass, and finally at $t = 3$ and fail, we could write x_3 as $[\theta_0, 2F, 1P, 3F]$, which induces a posterior distribution on our belief regarding θ.

w_k is a random variable which represents the transition from a prior belief to a new posterior distribution in the event we test. I For example, if we decided to test at level $t = 4$ at state $x_0 = [\theta_0]$, then we either transition to state $x_1 = [\theta_0, 4F]$ with probability of failure $q_{\theta,4}$, or $x_1 = [\theta_0, 4P]$ with the probability of passing $1 - q_{\theta,4}$.

Let V^* be the optimal policy cost with no restriction on the number of tests. For many of these analyses, however, we will be restricted by the number of tests we can perform, m. Thus, V_m^* denotes the optimal policy cost when we restrict the maximum number of tests to m.

3.2 Determining Bounds for Depth of Testing

As the cost to test c decreases, we expect that more tests would be performed under the optimal policy. An absolute upper bound for this number of tests m is the ratio between the rejection cost and c, i.e.,

$$m^* \leq \frac{R}{c}.$$

In this case, m^* represents the optimal maximum number of tests that are performed under the optimal policy. From this, we can see that m^* will be some finite number, as the cost to test is nonzero in our analysis. For future analysis, we keep R and c on the order of magnitude of 1 and $(0,.2]$ respectively to maintain a maximum number of tests that is both easy to compute and representative of problems in practice. Therefore, $V^* = V^*_{m^*}$. Note, however, that in most cases, the optimal policy will require fewer than m^* tests, but the bound above is useful in determining how many tests should be conducted given computational constraints.

3.3 Example at Testing Depth of $m = 2$

Continuing from the example in Section 2.1, we now consider testing to improve our original decision. For this problem, we only consider a maximum number of tests of $m = 2$, with the testing levels being restricted to $T = (1,2,3)$. Furthermore, we employ the same cost and prior parameters as before, with a testing cost of $c = 0.05$, a failure cost of $b = 300$, a rejection cost of $R = 1$, a desired ETC level of $\tau = 1$, and prior beliefs on θ_0 to have a $Normal(\mu = 5, \sigma = 3)$ distribution.

As before, when using these parameters, if we did not conduct any testing, our expected cost to approve the population would be 19.7, so we would therefore reject the population and incur the rejection cost of 1. Tables 3.1 and 3.2 give the likelihoods that a device would fail the test based solely on our prior distribution and testing level, and the resulting posterior distribution and testing level respectively.

Table 3.1: Probability q_{θ_0,t_1} that the first test fails

t_1	q
1	0.066
2	0.136
3	0.230

Table 3.2: Probability q_{θ,t_2} that the second test fails conditional on the results of the first test

t_2	$q \mid (1,F)$	$q \mid (1,P)$	$q \mid (2,F)$	$q \mid (2,P)$	$q \mid (3,F)$	$q \mid (3,P)$
1	0.667	0.023	0.423	0.009	0.272	0.004
2	0.878	0.084	0.707	0.046	0.520	0.021
3	0.955	0.179	0.881	0.128	0.750	0.074

Furthermore, we can construct the expected costs of approving the population given each permutation of our potential outcomes. Since we are only dealing with 3 testing levels, there are only 6 possible posterior distributions after the first test, and 21 possible posterior distributions after the second test. Tables 3.3 and 3.4 show the expected cost of approving the population given the posterior beliefs after testing. We have emboldened entries where the expected cost is less than the rejection cost.

Note that for a majority of these posterior distributions, the cost of approval is so immense (in some cases, rising above 200 times the rejection cost), we would

Table 3.3: Cost to approve population given the outcomes of the first test

$(1,F)$	$(1,P)$	$(2,F)$	$(2,P)$	$(3,F)$	$(3,P)$
200.27	7.06	127.23	2.83	81.77	1.21

Table 3.4: Cost to approve population given the outcomes of both tests

	$(1,F)$	$(1,P)$	$(2,F)$	$(2,P)$	$(3,F)$	$(3,P)$
$(1,F)$	233.09					
$(1,P)$	134.57	4.06				
$(2,F)$	210.94	65.71	161.12			
$(2,P)$	123.93	1.75	45.78	**0.81**		
$(3,F)$	203.57	36.23	138.78	20.00	104.26	
$(3,P)$	131.98	**0.75**	42.38	**0.38**	14.35	**0.20**

simply reject the population. Indeed, even if we only conducted one test, we would still reject the population. However, if we test intelligently and observe two passes, then we have the opportunity to approve the population with a high degree of confidence that our expected cost will be lower than the rejection cost.

Tables 3.5 and 3.6 are the updated tables of the approval/rejection decisions of $min(C(p),R)+c\cdot i$ for these costs, where i is the number of tests that have been performed. This represents the minimum between the expected cost of approval and the rejection cost with the sum of testing costs up to that point. We can see from Table 3.5 that if no more testing was permitted, the population would be rejected irrespective of the testing level or outcome. In Table 3.6, we see that there are four instances where the cost of approving the population is less than the rejection cost. Note that if we perform more tests, and devices pass a sufficient number of times, then the resulting posterior distribution would imply that the expected approval cost is lower than the rejection cost.

Table 3.5: Cost of the approve/reject decision after the first test

$(1,F)$	$(1,P)$	$(2,F)$	$(2,P)$	$(3,F)$	$(3,P)$
1.05	1.05	1.05	1.05	1.05	1.05

Table 3.6: Cost of the approve/reject decision after both tests

	$(1,F)$	$(1,P)$	$(2,F)$	$(2,P)$	$(3,F)$	$(3,P)$
$(1,F)$	1.1					
$(1,P)$	1.1	1.1				
$(2,F)$	1.1	1.1	1.1			
$(2,P)$	1.1	1.1	1.1	0.81		
$(3,F)$	1.1	1.1	1.1	1.1	1.1	
$(3,P)$	1.1	0.75	1.1	0.38	1.1	0.20

Using these values and the transition probabilities derived from $q_{\theta,t}$, we can quickly formulate the optimal solution by applying backward induction from the leaf nodes in the decision tree. Figure 3.1 shows the optimal policy tree with the possible test outcomes and the decision given each testing outcome. For reference, moving towards the top of the decision tree indicates the corresponding decision given a device failure at the previous testing level. In contrast, moving towards the bottom of the tree corresponds to passing a test at the earlier testing level. If we go down on the decision tree, it is the decision given the device had passed at the previous testing level. These are denoted by F and P respectively. The numbers in the boxes represent the testing levels that result in the expected minimum cost, and the "Approve" and "Reject" terms are to approve or reject the population and receive the expected cost. Note that all policies eventually end with the approval/rejection decision, where it is forced after the m^{th} test by design.

The expected cost of the optimal policy as seen below is 0.4507, which is

exactly the product of costs and probabilities depicted in Figure 3.1. From this optimal policy, one should initially test a device at $t = 3$. In the event the device fails, the population should be rejected. If the device passes, then one should test a second device at $t = 3$ again. If the second device fails, the population should be rejected. If it passes, the population should be approved. Although this could have been computed by hand, for there were only a few dozen calculations required, as the number of testing levels and the maximum number of tests increases, a more systematic approach is required.

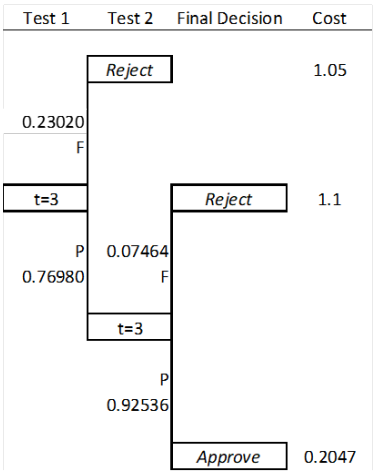

Figure 3.1: Decision Tree of the Optimal Policy

3.4 Alternative Policies in Practice

As we have suggested earlier, this approach to determining the optimal policy might be justified for certain parameters, but might be too complex for application. Therefore, it might be natural to assume one of two policies. First, one might assume that we can make the approval/rejection decision without any testing

at all. However, even from the rudimentary example in Section 3.3, if we were to make such a decision with no testing at all, we would expect to incur $min(R, C(p))$, or $min(1, 19.7)$, which would be the rejection cost of 1. Even from two intelligent tests, we would expect a decrease in cost of 0.5493 compared to simply rejecting the population given our prior beliefs.

The second natural policy one might consider would be to test at the threshold τ repeatedly. However, even at $m = 2$, we would make no improvement on our original decision if we were solely restricted to testing at τ, which can be seen in Table 3.6 and the resulting posterior distributions given the outcomes of testing at $t = 1$.

3.5 Optimal Policy Results over Varying Prior Beliefs

Given the dependence of the prior distribution for this solution methodology, and the high variance in optimal policies, it is imperative to have a prior distribution that accurately reflects one's beliefs on the mean of the device population toughness. Even minor changes in the variance of one's beliefs can result in significantly different policies. Let us consider an example where we use a prior distribution of $Normal \sim (5, 3)$, testing cost $c = 0.05$, an ETC variance of $s = 1$, and a cost of $b = 300$ as before. Furthermore, we test at a maximum depth of 4 tests, or $m = 4$, and our testing level set is $T = (1, 2, 3, 4, 5)$. Figure 3.2 shows the optimal policy generated using these parameters.

By changing the variance from $\sigma = 3$ to $\sigma = 2$, we get a significantly different optimal policy, as we see in Figure 3.3.

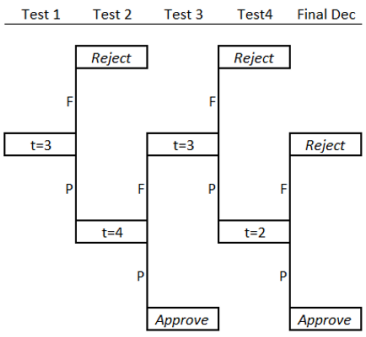

Figure 3.2: Decision Tree with a Prior Distribution of Normal(5,3) and $T = (1,2,3,4,5)$

When the expected cost of the optimal policy is determined from the MDP logic, the two components that are optimized are (1) the information being acquired when one tests at a particular testing level given the current prior beliefs, and (2) the probability that one acquires that information. If we were to test at a immensely high testing level, and the device survived that level and passed, then we would learn a significant amount of information regarding our beliefs of the population. That is, we would be much more confident that the population of devices would pass at the ETC τ, which we are assuming is lower than the mean intrinsic toughness of the population. However, the probability of the device actually surviving such a high testing level would be low. Conversely, we could test at a low testing level, to which we would expect the device to pass, but we would not be acquiring much information on the mean. We can see how the algorithm balances these two components via the optimal policy testing levels and the probability that the optimal policy makes the Approval/Rejection decision after the m^{th} test.

Depending on the prior distribution and testing cost, a maximum number of

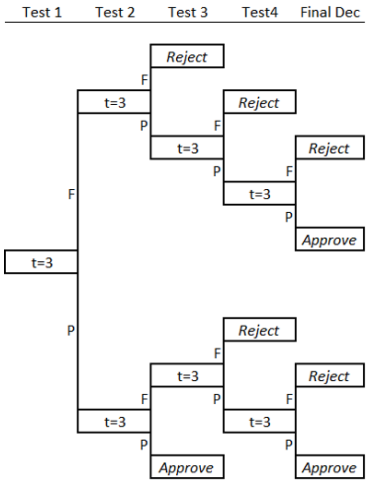

Figure 3.3: Decision Tree with a Prior Distribution of Normal(5,2) and $T = (1,2,3,4,5)$

tests $m \leq 4$ may not be sufficiently deep. In Figure 3.4, we plot the expected cost of the optimal policy at a testing cost of $c = 0.05$ against the range of maximum testing depth $m = (1,10)$. Again, the range of testing levels is $T = (1,2,3,4,5)$. From this, we can see that the expected optimal policy cost V_m^* asymptotically approaches the optimal policy cost V^* as we increase m.

Although the expected cost decreases, the incremental improvements in expected cost of the optimal policy comes at the cost of increasing computational time as a function of m. In the following section, we explore methods for approximating the optimal policy V^*.

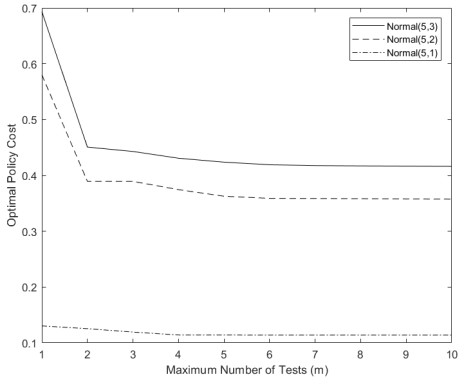

Figure 3.4: Optimal Policy Cost V_m^* for Different Prior Beliefs

Chapter 4

Experimentation and Results

Although the proposed model can be used to determine the optimal policy, numerical experiments indicate that determining the optimal policy is computationally intractable, even for relatively small problems. For this research, we used a computer with a AMD Ryzen 7 3750H processor and 16 GB of RAM. Furthermore, Matlab was used to code the algorithm and run the experiments. Given this, running even a single iteration with a maximum testing depth of $m = 10$ can be on the order of magnitude of ten hours, with $m = 11$ testing taking several days to compute. Therefore, we have developed a simple and direct method to approximate the optimal policy when computation is the bottleneck.

4.1 Heuristic Rollout Method to Approximate Optimal Policy

One potential solution to the issue of computational time as a constraint is to approximate the optimal policy using a set of heuristics past a certain point of the tree. Upon computing a full decision tree with a maximum testing depth of m, we can apply a simple selection mechanism that will continue to test past the testing depth m. Although using a heuristic can lead to sub-optimal policies compared to increasing the testing depth m, as m gets large with even a small set of testing

levels T, the time of computing additional levels increases dramatically. Therefore, by applying heuristics, we can reduce the number of calculations after the testing depth m. In doing so, we can obtain a policy that is at least as good as testing at m without any heuristic. This will allow the MDP algorithm to test deeper into the tree without calculating policies that are outside of the scope of the heuristic. Doing this will reduce the number of computations greatly, thus resulting in lower cost policies calculated in a fraction of the time.

One such heuristic is to, given a certain testing threshold, test at the same level repeatedly until there is a failure. Given the failure, one would reject the population. The algorithm stops testing once the expected decrease in the cost is less than the testing cost c. As such, if all tests pass after the m^{th} test outcome under the heuristic, the resulting posterior distribution has an expected cost of approval $C(p)$ less than had we made the approval decision immediately after the m^{th} test outcome. Figure 4.1 depicts the decision tree of this heuristic, where the heuristic testing level is $t = h$.

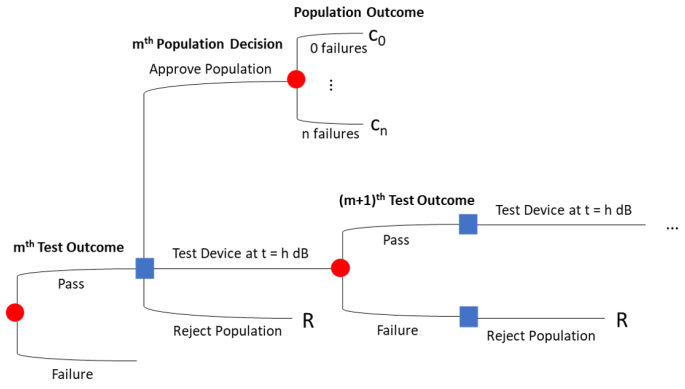

Figure 4.1: Decision Tree of Heuristic after m^{th} Test Outcome

Although this heuristic is incredibly simple, it can significantly reduce the needed computation time, while giving an expected cost that has the potential to be lower than V_m^*, and possibly as low as V^* if the heuristic aligns with the optimal policy provided from V^*. In Table 4.1, m is the maximum testing depth, and h is the heuristic level that we would continue to test at level h until we observe a failure or the expected reduction in cost of testing again is less than the testing cost c.

Table 4.1: Optimal cost at $c = 0.05$ with prior distribution $Normal(5,3)$ of standard method and heuristic method

m	$h=1$	$h=2$	$h=3$	$h=4$	$h=5$	V_m^*
3	0.44290	0.41909	0.43079	0.44099	0.44290	0.44290
4	0.43077	0.41823	0.42333	0.42917	0.43077	0.43077
5	0.42306	0.41649	0.42112	0.42373	0.42373	0.42373
6	0.41909	0.41497	0.41898	0.41909	0.41909	0.41909
7	0.41766	0.41487	0.41753	0.41766	0.41766	0.41766

With a rejection cost of $R = 1$ and testing cost of $c = 0.05$, we would expect a maximum of $m = 20$ tests necessary to achieve V^* from the crude bound discussed in Section 3.2. However, given the computer used for this research, running beyond a testing depth of $m = 10$ is not feasible. Therefore, this heuristic with a sufficiently high m can be quite powerful in approximating V^* without having to compute all the possible posterior distributions and testing decisions with $m = 20$. We can see that h is most effective at reducing the cost when it is in the interval $(2,4)$, again suggesting that testing levels at τ do not provide much information when making the decision to approve or reject the population. If we compare the heuristic approach to the standard approach of determining the optimal cost in terms of computational time, we find that there is potential to achieve a similar or lower optimal cost V_m^* in

a fraction of the time when using a heuristic at a lower testing depth m.

In some cases, it is also possible to achieve V_m^* with a lower restricted testing depth m. If we set the cost to test $c = 0.1$, and the prior distribution stays the same at $Normal(5,3)$, we can determine the true optimal cost by setting our maximum testing depth to $m = 10$, as this is our crude bound for the maximum number of needed tests. From this, we have computed that the optimal cost is 0.52622. Working backwards from $m = 10$, the lowest number of required tests to achieve this optimal policy is found at $m = 7$, which takes approximately 4.49 minutes to compute. However, if we use the heuristic method, we can achieve the exact same policy and cost if we set our $m = 5$ and $h = 2$, with the computational time being 0.32 minutes, which is an 92.9% decrease in time needed. Although this is one example with a specific prior distribution and cost parameters, it demonstrates that we can obtain the optimal policy with the heuristic.

If we plot the optimal policy cost V_m^* as a function of m against the heuristic, as in Figure 4.2, and compare that to the computational time required, as in Figure 4.3, we can see that the heuristic is quite powerful in attaining information at a greatly reduced computational cost.

By utilizing a cost to test of $c = 0.1$, we can see in Figure 4.4 that the optimal cost V_m^* can be achieved by the heuristic with less computational time. Note how, in some circumstances, the heuristic will do no better than computing the optimal policy cost without the heuristic.

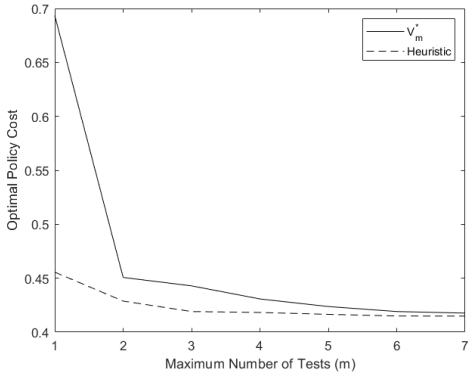

Figure 4.2: Optimal Cost Comparison between V_m^* and Heuristic with Prior Beliefs of $Normal(5,3)$ and $c = 0.05$

4.2 Sensitivity of Approximation of Heuristic

So far, we have only looked at how the approximation of the optimal cost V_m^* varies as a function of relative testing cost, but it is also important to analyze the effects of how our initial beliefs can affect the efficacy of this approximation. To this effect, we shall vary τ, as this will determine how confident we need to be about the population of devices in the approval/rejection decision. Table 4.2 is generated from the same set of prior beliefs, $Normal(5,3)$, with a cost to test of $c = 0.1$.

Table 4.2: Comparison of Optimal Cost and Computational Time between V^* and the Heuristic Method

τ	Heuristic	V^*	Optimality Gap	Time Ratio (mins)
0.5	0.42875	0.42872	0.0064%	5.17/4307.77
1.0	0.52621	0.52621	0%	0.33/4.71
1.5	0.60833	0.60833	0%	1.30/25.26
2.0	0.69124	0.69121	0.0048%	5.17/4307.77

As we can see from Table 4.2, the difference between V^* and the heuristic

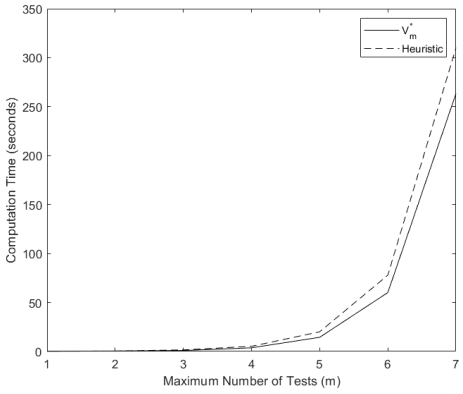

Figure 4.3: Computational Time Comparison between V_m^* and Heuristic with Prior Beliefs of $Normal(5,3)$ and $c = 0.05$

method at a testing depth of $m = 7$ can often times achieve the same cost in significantly less amount of computational time. In many of these cases, hours were shaven off of the computational time — in the event that a large number of tests are required, such as 15-20, utilizing such a heuristic could be immensely powerful in making intelligent decisions in a shorter time frame.

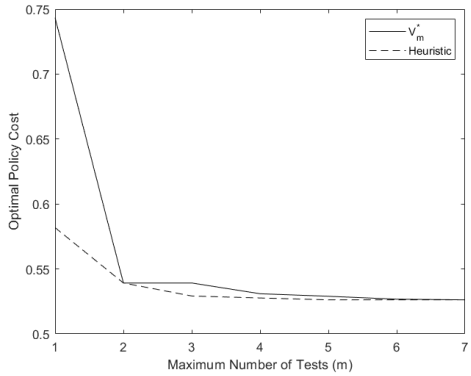

Figure 4.4: Optimal Cost Comparison between V_m^* and Heuristic with Prior Beliefs of $Normal(5,3)$ and $c = 0.1$

Chapter 5

Conclusion

Through this unique situation of embedded uncertainty over a population of devices, we have developed solution methodologies that are uniquely positioned to reduce expected cost when compared to the traditional statistics method. Even when compared to similar problems within the civil engineering sphere, our consideration of cost within the testing procedure better reflects the pragmatic utilization of the problem, and is thereby more useful for a variety of industries. Given the flexibility of our models, whether through prior distributions or heuristics means to approximate optimal policies while reducing computational time, we feel that this testing procedure has immense value.

Throughout the construction of these models and testing, several key insights emerged:

- Testing at or below the testing threshold τ was almost never valuable within any optimal testing policy. When maximizing the product of information gained and the probability of receiving that information, optimal policies tend to test below the mean, but much higher than τ.

- Heuristic methods of approximating the optimal policy, even if rudimentary,

can be quite effective in reducing computational time while maintaining similar optimal policies and expected cost.

- Even small variation in the prior confidence or mean can lead to significant changes in the optimal testing policy.

- When testing with a testing cost that is approximately 5% of the minimum of the initial approval/rejection decision, many optimal policies can be computed with only a maximum testing depth of ten or less.

If the cost to test a device is very small, then the impact of simply testing a statistically significant number of devices compared to intelligently testing at various levels would be negligible. However, if the cost to test is relatively high, such that only a small number of tests can be performed, then it is imperative to employ a testing schema that can maximize the information acquired to make the most informed decision possible.

Appendix

Listing 1: Sequential Testing Optimal Policy Algorithm

```
tic
clear

%Prior, ETC, and cost parameters
mu0=5;
s0=3;
s=1;
tau=1.5;
b=300;
C=@(p) p.*b;
R=1;
testCost=0.1;

%Generate values of theta
mintheta=.01;
maxtheta=15;
m=mintheta:mintheta:maxtheta;
theta=log((m.^2)./sqrt(s+m.^2));
sigma=sqrt(log(s./(m.^2)+1));
lt=length(theta);

%Generate prior distribution probabilities
   through discretization of
%initial prior distribution
P0=zeros(lt,1);
for i=1:lt
    if i==1
        P0(i,1)=normcdf(m(i),mu0,s0)-normcdf
            (0,mu0,s0);
    else
        P0(i,1)=normcdf(m(i),mu0,s0)-normcdf(m
            (i-1),mu0,s0);
    end
end
```

```
%Normalize prior discretization
all=sum(P0);
P0=P0/all;

%Generate liklihood p with list of thetas at
    tau
logp=logncdf(tau,theta,sigma);
p=sum(logp.*transpose(P0(:,1)));
C0=C(p);

%Probability matrix parameters
t=1:1:5;
numt=length(t);
numtests=8;

%Generate pascal numbers to determine number
    of unique posteriors
deep=max(numtests+2,2*numt);
pasc=pascal(deep);
pasclength=pasc(2*numt,1:numtests+2);

%Generate Xi matrices
%Xi is a matrix containing the unique sequence
     of test level and outcomes;
%for example, X2 has two columns, with column
    one being the test and
%outcome of the first test, and the second
    column being the test and
%outcome of the secoond test.

%The numerical value at the i-jth is as
    follows:
%1 corresponds to a failure at t(1), 2 a pass
    at t(1)
%3 corresponds to a failure at t(2), 4 a pass
    at t(2), and so on.
%Therefore, the range is from 1:2*numtests, to
```

```
    account for all test levesl
%and outcomes

i=2;
X1=1:1:numt*2;
X1=X1';
while i<=numtests
    genvarname('X',num2str(i));
    eval(['X' num2str(i) '=[];']);
    j=1;
    while j<=length(eval(['X' num2str(i-1)]))
        temp=eval(['X' num2str(i-1) '(j,i-1);'
            ]);
        val=eval(['X' num2str(i-1) '(j,:);']);
        while temp<=numt*2
            eval(['X' num2str(i) '=[X' num2str
                (i) ';val temp];']);
            temp=temp+1;
        end
        j=j+1;
    end
    i=i+1;
end

%Construct Pi and Ci matrices, where Pi is the
    posterior distributions
%after the ith test, and Ci is the cost of
   simply approving the population
%after the ith test (which corresponds to the
   Pith distribution)
i=1;
j=1;
while i<=numtests
    numpost=(numt*2)^i;
    genvarname('P',num2str(i));
    eval(['P' num2str(i) '=zeros(lt, 
        pasclength(i+1));']);
```

```
genvarname('C',num2str(i));
eval(['C' num2str(i) '=zeros(1,pasclength
   (i+1));']);
while j<=pasclength(i+1)

    %Set prior distribution
    if i==1
       prior=P0;
    elseif i>1
        val=eval(['X' num2str(i) '(j,1:i
           -1);']);
        ind=eval(['find(ismember(X'
           num2str(i-1) ',val,"rows"));'])
           ;
        prior=eval(['P' num2str(i-1) '(:,
           ind);']);
    end
    prior=transpose(prior);

    %Most recent test index
    temp=eval(['X' num2str(i) '(j,i);']);
    testind=ceil(temp/2);
    fail=logncdf(t(testind),theta,sigma);
    if mod(temp,2)==1
        outcome=fail;
    else
        outcome=1-fail;
    end

    outpost=outcome.*prior;
    totoutpost=sum(outpost);
    post=outpost/totoutpost;

    %Store values for Pi and Ci
    eval(['P' num2str(i) '(:,j)=post;']);
    eval(['C' num2str(i) '(1,j)=C(sum(post
       .*logp))+testCost*i;']);
```

```
        j=j+1;
    end
    i=i+1;
    j=1;
end

%Construct failure transition probabilities
i=1;
j=1;
while i<=numtests
    genvarname('F',num2str(i));
    eval(['F' num2str(i) '=zeros(numt,␣
       pasclength(i));']);
    while j<=pasclength(i)
        if i==1
            prior=P0;
        elseif i>1
            val=eval(['X' num2str(i-1) '(j,1:i
               -1);']);
            ind=eval(['find(ismember(X'
               num2str(i-1) ',val,"rows"));'])
               ;
            prior=eval(['P' num2str(i-1) '(:,
               ind);']);
        end
        prior=transpose(prior);

        z=1;
        for k=t
            fail=logncdf(k,theta,sigma);
            failpost=fail.*prior;
            totalfailpost=sum(failpost);
            eval(['F' num2str(i) '(z,j)=
               totalfailpost;']);
            z=z+1;
        end
```

```
        j=j+1;
    end
    i=i+1;
    j=1;
end

%Di is the decision vector at each round of
   testing
%Vi is the value of the Dith decision

i=numtests;
j=1;
k=1;
while i>=1
    genvarname('D',num2str(i));
    eval(['D' num2str(i) '=zeros(1,pasclength
       (i));']);
    genvarname('V',num2str(i));
    eval(['V' num2str(i) '=zeros(1,pasclength
       (i));']);
    while j<=pasclength(i)
        if i>1
            tout=eval(['X' num2str(i-1) '(j,:)
               ;']);
            tempcost=min(costa(tout),testCost
               *(i-1)+R);
        elseif i==1
            tout=[];
            tempcost=min(C0,R);
        end

        if tempcost == testCost*(i-1)+R
            eval(['D' num2str(i) '(1,j)=-1;'])
               ;
        end

        k=1;
```

```
        while k<=numt
            prob=probf([tout k]);
            if i==numtests
                cost1=prob*min(costa([tout k
                    *2-1]),testCost*i+R);
                cost2=(1-prob)*min(costa([tout
                     k*2]),testCost*i+R);
            else
                ind1=ismember(eval(['X'
                    num2str(i)]),sort([tout k
                    *2-1]),"rows");
                ind2=ismember(eval(['X'
                    num2str(i)]),sort([tout k
                    *2]),"rows");
                val1=eval(['V' num2str(i+1) '
                    (1,ind1);']);
                val2=eval(['V' num2str(i+1) '
                    (1,ind2);']);
                cost1=prob*val1;
                cost2=(1-prob)*val2;
            end
            newcost=cost1+cost2;
            if newcost<tempcost
                eval(['D' num2str(i) '(1,j)=k;
                    ']);
                tempcost=newcost;
            end
           k=k+1;
        end
        eval(['V' num2str(i) '(1,j)=tempcost;'
            ])
        j=j+1;
        k=1;
    end
    i=i-1;
    j=1;
    k=1;
```

```
end

i=2;
j=1;
d1=D1;
while i<=numtests
    genvarname('d',num2str(i));
    eval(['d' num2str(i) '=zeros(2^(i-1),i);'
        ]);
    while j<=2^(i-2)
        test=eval(['d' num2str(i-1) '(j,i-1);'
            ]);
        outrest=eval(['d' num2str(i-1) '(j,1:i
            -2);']);
        if test<=0
            eval(['d' num2str(i) '(j*2-1,:)=[d
                ' num2str(i-1) '(j,:) -2];']);
            eval(['d' num2str(i) '(j*2,:)=[d'
                num2str(i-1) '(j,:) -2];']);
        else
            out1=test*2-1;
            out2=test*2;
            newout1=[outrest out1];
            newout2=[outrest out2];
            sn1=sort(newout1);
            sn2=sort(newout2);
            ind1=ismember(eval(['X' num2str(i
                -1)]),sn1,"rows");
            ind2=ismember(eval(['X' num2str(i
                -1)]),sn2,"rows");
            val1=eval(['D' num2str(i) '(1,ind1
                );']);
            val2=eval(['D' num2str(i) '(1,ind2
                );']);
            eval(['d' num2str(i) '(j*2-1,:)=[
                newout1 val1];']);
            eval(['d' num2str(i) '(j*2,:)=[
```

```
                newout2_val2];']);
        end
        j=j+1;
    end
    i=i+1;
    j=1;
end

l=0;
p=1;
i=1;
j=1;
while i<=length(eval(['d' num2str(numtests)]))
    if eval(['d' num2str(numtests) '(i,
       numtests)>0'])
        while j<=numtests-1
            test=eval(['d' num2str(numtests) '
               (i,j);']);
            tlevel=ceil(test/2);
            ptrans=probf([eval(['d' num2str(
               numtests) '(i,1:j-1)']) tlevel
               ]);
            if mod(test,2)==0
                ptrans=1-ptrans;
            end
            p=p*ptrans;
            j=j+1;
        end
        l=l+p;
    end
    i=i+1;
    j=1;
    p=1;
end
```

V1

```
toc

%FUNCTIONS

%probf takes a vector of test levels/outcomes,
    with the final value being
%the index of the ith test -- with i being the
    length of the vector x
%probf returns the probability that the ith
   test will fail given the test
%level/outcomes.
function f = probf(x)
    i=length(x);
    t=x(i);
    F="F"+num2str(i);
    fi=evalin("base", F);

    if i==1
       f=fi(t,1);
    elseif i>1
        val=x(1:i-1);
        val=sort(val);
        xj="X"+num2str(i-1);
        xj=evalin("base", xj);
        ind=ismember(xj,val,"rows");
        f=fi(t,ind);
    end
end

%costa takes a vector of the test levels/
   outcomes and returns the cost of
%simply approving, ca.
function ca = costa(y)
    i=length(y);
    C="C"+num2str(i);
    ci=evalin("base", C);
```

```
    if i==1
        ca=ci(1,y);
    elseif i>1
         val=sort(y);
         xj="X"+num2str(i);
         xj=evalin("base", xj);
         ind=ismember(xj,val,"rows");
         ca=ci(1,ind);
    end
end
```

Listing 2: Sequential Testing Optimal Policy Algorithm with Heuristic

```
tic
clear

%Prior, ETC, and cost parameters
mu0=5;
s0=3;
s=1;
tau=1;
b=300;
C=@(p) p.*b;
R=1;
testCost=0.05;

%Generate values of theta
mintheta=.01;
maxtheta=15;
m=mintheta:mintheta:maxtheta;
theta=log((m.^2)./sqrt(s+m.^2));
sigma=sqrt(log(s./(m.^2)+1));
lt=length(theta);

%Generate prior distribution probabilities
```

```
    through discretization of
%initial prior distribution
P0=zeros(lt,1);
for i=1:lt
    if i==1
        P0(i,1)=normcdf(m(i),mu0,s0)-normcdf
            (0,mu0,s0);
    else
        P0(i,1)=normcdf(m(i),mu0,s0)-normcdf(m
            (i-1),mu0,s0);
    end
end

%Normalize prior discretization
all=sum(P0);
P0=P0/all;

%Generate liklihood p with list of thetas at
    tau
logp=logncdf(tau,theta,sigma);
p=sum(logp.*transpose(P0(:,1)));
C0=C(p);

%Probability matrix parameters
t=1:1:5;
numt=length(t);
numtests=1;

%Rollout heuristic
h=2;

%Generate pascal numbers to determine number
    of unique posteriors
deep=max(numtests+2,2*numt);
pasc=pascal(deep);
pasclength=pasc(2*numt,1:numtests+2);
```

```
%Generate Xi matrices
%Xi is a matrix containing the unique sequence
    of test level and outcomes;
%for example, X2 has two columns, with column
   one being the test and
%outcome of the first test, and the second
   column being the test and
%outcome of the secoond test.

%The numerical value at the i-jth is as
   follows:
%1 corresponds to a failure at t(1), 2 a pass
   at t(1)
%3 corresponds to a failure at t(2), 4 a pass
   at t(2), and so on.
%Therefore, the range is from 1:2*numtests, to
    account for all test levesl
%and outcomes

i=2;
X1=1:1:numt*2;
X1=X1';
while i<=numtests
    genvarname('X',num2str(i));
    eval(['X' num2str(i) '=[];']);
    j=1;
    while j<=length(eval(['X' num2str(i-1)]))
        temp=eval(['X' num2str(i-1) '(j,i-1);'
            ]);
        val=eval(['X' num2str(i-1) '(j,:);']);
        while temp<=numt*2
            eval(['X' num2str(i) '=[X' num2str
                (i) ';val temp];']);
            temp=temp+1;
        end
        j=j+1;
    end
```

```
    i=i+1;
end

%Construct Pi and Ci matrices, where Pi is the
    posterior distributions
%after the ith test, and Ci is the cost of
   simply approving the population
%after the ith test (which corresponds to the
   Pith distribution)
i=1;
j=1;
while i<=numtests
    numpost=(numt*2)^i;
    genvarname('P',num2str(i));
    eval(['P' num2str(i) '=zeros(lt,␣
       pasclength(i+1));']);
    genvarname('C',num2str(i));
    eval(['C' num2str(i) '=zeros(1,␣pasclength
       (i+1));']);
    while j<=pasclength(i+1)

        %Set prior distribution
        if i==1
            prior=P0;
        elseif i>1
            val=eval(['X' num2str(i) '(j,1:i
               -1);']);
            ind=eval(['find(ismember(X'
               num2str(i-1) ',val,"rows"));'])
               ;
            prior=eval(['P' num2str(i-1) '(:,
               ind);']);
        end
        prior=transpose(prior);

        %Most recent test index
        temp=eval(['X' num2str(i) '(j,i);']);
```

```
            testind=ceil(temp/2);
            fail=logncdf(t(testind),theta,sigma);
            if mod(temp,2)==1
                outcome=fail;
            else
                outcome=1-fail;
            end

            outpost=outcome.*prior;
            totoutpost=sum(outpost);
            post=outpost/totoutpost;

            %Store values for Pi and Ci
            eval(['P' num2str(i) '(:,j)=post;']);
            eval(['C' num2str(i) '(1,j)=C(sum(post
                .*logp))+testCost*i;']);

            j=j+1;
        end
        i=i+1;
        j=1;
    end

    %Construct failure transition probabilities
    i=1;
    j=1;
    while i<=numtests
        genvarname('F',num2str(i));
        eval(['F' num2str(i) '=zeros(numt,␣
            pasclength(i));']);
        while j<=pasclength(i)
            if i==1
                prior=P0;
            elseif i>1
                val=eval(['X' num2str(i-1) '(j,1:i
                    -1);']);
                ind=eval(['find(ismember(X'
```

```
                num2str(i-1) ',val,"rows"));']);
                ;
            prior=eval(['P' num2str(i-1) '(:,
                ind);']);
        end
        prior=transpose(prior);

        z=1;
        for k=t
            fail=logncdf(k,theta,sigma);
            failpost=fail.*prior;
            totalfailpost=sum(failpost);
            eval(['F' num2str(i) '(z,j)=
                totalfailpost;']);
            z=z+1;
        end
        j=j+1;
    end
    i=i+1;
    j=1;
end


%Create updated values for the mth test onward
    using heuristic
num_post=length(eval(['X', num2str(numtests)])
    );
h_depth=zeros(num_post,1);
value_rollout=zeros(num_post,1);

z=1;
while z<=num_post

    go=true;
    dep=0;

    while go
```

```
if dep==0
    prior=eval(['P' num2str(numtests)
        '(:,' num2str(z) ')']);
    prior=transpose(prior);
end


fail=logncdf(h,theta,sigma);
pass=1-fail;

outpost_pass=pass.*prior;
outpost_fail=fail.*prior;
probpass=sum(outpost_pass);
probfail=1-probpass;
post_pass=outpost_pass/probpass;
post_fail=outpost_fail/probfail;

fail_cost=min(R,C(sum(post_fail.*logp)
   ))+testCost*(dep+1+numtests);
pass_cost=min(R,C(sum(post_pass.*logp)
   ))+testCost*(dep+1+numtests);


prior=post_pass;

if dep==0
    PROB=[probfail;probpass];
    COST=[fail_cost;pass_cost];
    newcost=probfail*fail_cost+
        probpass*pass_cost;
else

    if dep==3
        lol=1;
    end
    PROB=[PROB, [probfail;probpass]];
    COST=[COST, [fail_cost;pass_cost
```

```
                ]];
            i=dep;
            newPROB=PROB;
            while i>=1
                p=newPROB(2,i);
                newPROB=[newPROB(:,1:i)
                    newPROB(:,i+1:length(PROB))
                    *p];
                i=i-1;
            end
            newcost=sum(newPROB(1,:).*COST
                (1,:))+newPROB(2,length(newPROB
                ))*COST(2,length(newPROB));
        end

        if dep==0
            x=eval(['X' num2str(numtests) '('
                num2str(z) ',:)']);
            oldcost=min(R+testCost*numtests,
                costa(x));
        end

        if oldcost<newcost
            go=false;
        else
            oldcost=newcost;
            dep=dep+1;
        end
    end



    h_depth(z)=dep;
    value_rollout(z)=oldcost;
    z=z+1;
end
```

```
%Di is the decision vector at each round of
    testing
%Vi is the value of the Dith decision

i=numtests;
j=1;
k=1;
while i>=1
    genvarname('D',num2str(i));
    eval(['D' num2str(i) '=zeros(1,pasclength
        (i));']);
    genvarname('V',num2str(i));
    eval(['V' num2str(i) '=zeros(1,pasclength
        (i));']);
    while j<=pasclength(i)
        if i>1
            tout=eval(['X' num2str(i-1) '(j,:)
                ;']);
            tempcost=min(costa(tout),testCost
                *(i-1)+R);
        elseif i==1
            tout=[];
            tempcost=min(C0,R);
        end

        if tempcost == testCost*(i-1)+R
            eval(['D' num2str(i) '(1,j)=-1;'])
                ;
        end

        k=1;
        while k<=numt
            prob=probf([tout k]);
            if i==numtests
                ind1=ismember(eval(['X'
                    num2str(i)]),sort([tout k
```

```
                *2-1]),'rows');
            ind2=ismember(eval(['X'
                num2str(i)]),sort([tout k
                *2]),'rows');
            cost1=prob*value_rollout(ind1)
                ;
            cost2=(1-prob)*value_rollout(
                ind2);
        else
            ind1=ismember(eval(['X'
                num2str(i)]),sort([tout k
                *2-1]),'rows');
            ind2=ismember(eval(['X'
                num2str(i)]),sort([tout k
                *2]),'rows');
            val1=eval(['V' num2str(i+1) '
                (1,ind1);']);
            val2=eval(['V' num2str(i+1) '
                (1,ind2);']);
            cost1=prob*val1;
            cost2=(1-prob)*val2;
        end
        newcost=cost1+cost2;
        if newcost<tempcost
            eval(['D' num2str(i) '(1,j)=k;
                ']);
            tempcost=newcost;
        end
       k=k+1;
    end
    eval(['V' num2str(i) '(1,j)=tempcost;'
        ])
    j=j+1;
    k=1;
end
i=i-1;
j=1;
```

```
    k=1;
end


i=2;
j=1;
d1=D1;
while i<=numtests
    genvarname('d',num2str(i));
    eval(['d' num2str(i) '=zeros(2^(i-1),i);'
        ]);
    while j<=2^(i-2)
        test=eval(['d' num2str(i-1) '(j,i-1);'
            ]);
        outrest=eval(['d' num2str(i-1) '(j,1:i
            -2);']);
        if test<=0
            eval(['d' num2str(i) '(j*2-1,:)=[d
                ' num2str(i-1) '(j,:)-2];']);
            eval(['d' num2str(i) '(j*2,:)=[d'
                num2str(i-1) '(j,:)-2];']);
        else
            out1=test*2-1;
            out2=test*2;
            newout1=[outrest out1];
            newout2=[outrest out2];
            sn1=sort(newout1);
            sn2=sort(newout2);
            ind1=ismember(eval(['X' num2str(i
                -1)]),sn1,"rows");
            ind2=ismember(eval(['X' num2str(i
                -1)]),sn2,"rows");
            val1=eval(['D' num2str(i) '(1,ind1
                );']);
            val2=eval(['D' num2str(i) '(1,ind2
                );']);
            eval(['d' num2str(i) '(j*2-1,:)=[
```

```
                newout1 val1 ]; ']);
            eval(['d' num2str(i) '(j*2,:)=[
                newout2 val2 ]; ']);
        end
        j=j+1;
    end
    i=i+1;
    j=1;
end

l=0;
p=1;
i=1;
j=1;
while i<=length(eval(['d' num2str(numtests)]))
    if eval(['d' num2str(numtests) '(i,
        numtests)>0'])
        while j<=numtests-1
            test=eval(['d' num2str(numtests) '
                (i,j);']);
            tlevel=ceil(test/2);
            ptrans=probf([eval(['d' num2str(
                numtests) '(i,1:j-1)']) tlevel
                ]);
            if mod(test,2)==0
                ptrans=1-ptrans;
            end
            p=p*ptrans;
            j=j+1;
        end
        l=l+p;
    end
    i=i+1;
    j=1;
    p=1;
end
```

```
toc

%FUNCTIONS

%probf takes a vector of test levels/outcomes,
    with the final value being
%the index of the ith test -- with i being the
    length of the vector x
%probf returns the probability that the ith
   test will fail given the test
%level/outcomes.
function f = probf(x)
    i=length(x);
    t=x(i);
    F="F"+num2str(i);
    fi=evalin("base", F);

    if i==1
       f=fi(t,1);
    elseif i>1
        val=x(1:i-1);
        val=sort(val);
        xj="X"+num2str(i-1);
        xj=evalin("base", xj);
        ind=ismember(xj,val,"rows");
        f=fi(t,ind);
    end
end

%costa takes a vector of the test levels/
   outcomes and returns the cost of
%simply approving, ca.
function ca = costa(y)
    i=length(y);
    C="C"+num2str(i);
    ci=evalin("base", C);
```

```
    if  i==1
        ca=ci(1,y);
    elseif  i>1
         val=sort(y);
         xj="X"+num2str(i);
         xj=evalin("base",  xj);
         ind=ismember(xj,val,"rows");
         ca=ci(1,ind);
    end
end
```

Bibliography

[1] Y Abdallah, Shadi Najjar, and George Saad. Reliability-based design of proof load test programs for foundations. *Geotechnical Engineering*, 46:94–101, 2015.

[2] Nammam Ali Azadi. Bayesian sequential experimental design for binary response data with application to electromyographic experiments. *Bayesian Analysis*, 9(2):287–306, 2014.

[3] Wilfrid Joseph Dixon and Am M. Mood. A method for obtaining and analyzing sensitivity data. *Journal of the American Statistical Association*, 43(241):109–126, 1948.

[4] Hovav A. Dror and David M. Steinberg. Sequential experimental designs for generalized linear models. *Journal of the American Statistical Association*, 103(481):288–298, 2008.

[5] Xun Huan and Yousef M. Marzouk. Sequential bayesian optimal experimental design via approximate dynamic programming. *arXiv:1604.08320*, 2016.

[6] Robert Najjar, Shadi B. Gilbert. Importance of proof-load tests in foundation reliability. *Journal of the American Statistical Association*, pages 340–347, 2009.

[7] Barry T. Neyer. A d-optimality-based sensitivity test. *Technometrics*, 36(1):61–70, 1994.

[8] Jianguo Sun. The statistical analysis of interval-censored failure time data. *Springer Science Business Media*, 2007.

[9] Stephen Venter, Gerhard Scotti. Accounting for proof test data in a reliability-based design optimization framework. *AIAA Journal*, 50:2159–2167, 2012.

[10] Lei Wang. Sequential LND sensitivity test for binary response data. *Journal of Applied Statistics*, 40(11):2732–2384, 2013.

[11] CF Jeff Wu and Yubin Tian. Three-phase optimal design of sensitivity experiments. *Journal of Statistical Planning and Inference*, 149:1–15, 2014.

[12] Linda J. Young and Robert G. Easterling. Estimation of extreme quantiles based on sensitivity tests: a comparative study. *Technometrics*, 36(1):48–60, 1994.