Probability in the Engineering and Informational Sciences, **15**, 2001, 519–533. Printed in the U.S.A.

A GENERAL SOFTWARE TESTING MODEL INVOLVING OPERATIONAL PROFILES

SÜLEYMAN ÖZEKICI, İ. KUBAN ALTINEL, AND EBRU ANGÜN
Boğaziçi University
Department of Industrial Engineering
Bebek-İstanbul, Turkey
E-mail: ozekici@boun.edu.tr

An operational profile describes, in a probabilistic way, how a software is utilized by its users. It makes the testing procedure more realistic and efficient. We consider a model where the software is tested sequentially in all of the operations that it is designed to perform. The stochastic and deterministic model parameters involving costs and failures all depend on the operations. In particular, the failure process generated by each fault is quite general and debugging is not necessarily perfect. Our aim is to find the optimal testing durations in all of the operations in order to minimize the total expected cost. This problem leads to an interesting nonlinear programming formulation that can be solved using well-known procedures in convex optimization.

1. INTRODUCTION

In a recent article, Özekici et al. [9] introduced an interesting model to test software with an operational profile. It involves the determination of optimal durations of testing in a number of test cases where each fault creates failures depending on the operation that is performed. There is perfect debugging so that the fault causing a failure is removed with certainty. Moreover, the distribution of the failure time for each fault is exponential, as in the Jelinski and Moranda [5] model. This leads to a nonlinear programming formulation with an explicit solution under reasonable conditions. A Bayesian perspective on this model is provided by Özekici and Soyer [12]. Our aim in this article is to extend this line of research in two directions. We will have imperfect debugging and, more importantly, the failure process is not necessarily Poisson.

© 2001 Cambridge University Press 0269-9648/01 $12.50

https://doi.org/10.1017/S0269964801154069 Published online by Cambridge University Press

The determination of the optimal release time is perhaps the most challenging decision problem during the development and testing phase of software. It is particularly appealing to researchers and practitioners in operations research and management sciences. In life-critical applications, the most important attribute of software is its reliability. However, in commercial applications, cost is as important as reliability. Before release, software is tested to decrease the number of inherent faults and, hence, increase its reliability. The testing phase is time-consuming and costly. On the other hand, if software is released without sufficient testing, this will result in a high debugging cost after release. This fact constitutes a trade-off between costs incurred during testing and costs incurred after release.

There is abundant literature on software testing and release. They involve a variety of models. Forman and Singpurwalla [3] provide a statistical perspective and Goel and Okumoto [4] propose a static optimization approach. Ross [13] uses a decision rule that is similar to acceptance sampling in quality control. The decision to release the software is based on the number of faults corrected in Bai and Yun [1] and a dynamic model is discussed in Özekici and Çatkan [11]. This list is by no means exhaustive, as reflected by the review in Özekici and Çatkan [10]. An undesirable restriction in almost all of these models is that the parameters of the software failure process and the cost parameters are independent of the operation performed. This issue is discussed in detail by Musa [7,8], who introduced the concept of an operational profile. The operational profile simply consists of the set of all operations that a system is designed to perform and their occurrence probabilities. It reduces the system risk by making it more realistic. Furthermore, it makes the testing procedure faster and more efficient.

Our model is one that incorporates the operational profile in the testing problem when there is imperfect debugging and the failure time distributions are not necessarily exponential. In Section 2, we present the testing model, and the optimization problem is discussed in Section 3. A special case of our model, namely the exponential model, will be considered in Section 4, together with an illustration. Section 5 will be devoted to solving the general nonexponential model.

2. THE OPTIMAL TESTING PROBLEM

We consider the optimal testing problem of software with an operational profile that consists of K well-defined operations. Before release, it is to be tested sequentially in all of these operations and the decision problem is to determine the testing durations. The objective of testing is to decrease the number of faults in the software before releasing it. All stochastic and economic parameters of the model depend on the operations as given by the following notation:

c_i: testing cost per unit time during operation i

t_i: decision variable representing the duration of testing for operation i

e_i: debugging cost after release due to failure during operation i

https://doi.org/10.1017/S0269964801154069 Published online by Cambridge University Press

d_i: debugging cost before release due to failure during operation i

π_i: probability that the software will fail while performing operation i after release

μ: expected number of inherent faults in the software before testing

$m_i(t)$: expected number of failures caused by a fault until time t during operation i

$F_i(t)$: distribution of the time to failure during operation i

p_i: probability of successfully removing a fault after failure during operation i.

The model to determine the optimal testing strategy is based on the following assumptions:

ASSUMPTION 1: *(1) Whenever a software failure occurs, the fault that caused the failure will be removed instantaneously with probability* $0 \leq p_i \leq 1$ *during operation i. (2) No new faults are introduced during debugging. (3) The total number of faults in the software is a random variable with some known mean* μ. *(4) The failures caused by each fault occur independently and randomly in time according to a counting process with some mean value function* m_i *during operation i. (5)* $c_i \geq 0$, $e_i \geq d_i \geq 0$. *(6) A fixed sequence of testing is given.*

The first assumption implies that debugging is imperfect. Each fault is removed with probability p_i after a failure, or it remains in the program with probability $q_i = 1 - p_i$ during operation i. The purpose of the second assumption is to ensure that there is reliability growth so that testing is beneficial. The third assumption indicates that the total number of faults in the program is a function of many factors such as program size, complexity, and human performance. Hence, it is reasonable to consider the total number of faults in the program as a random variable with a known mean μ.

The fourth assumption describes how failures are caused by a fault. The faults are assumed to be identical and independent in the sense that they generate independent and identical failure processes. This is a common assumption made to model failures in most of the literature on software reliability engineering. If the faults are not identical, then they can be classified into different types, where each type of fault generates its own failure process. The discussions and results presented in this article still apply to any type with a random number of initial faults. The assumption on independence may not hold, especially if a fault prevents another fault from being discovered. For example, a fault in the user interface may prevent some codes or algorithms from being accessed during testing. The inherent faults there may only be accessed once the fault in the user interface is eventually removed. In such cases, it is necessary to classify the faults into different types with a model that describes the dependence among the different types. The dependence is specific to the software in question because it can only be described through its own internal structure. For

https://doi.org/10.1017/S0269964801154069 Published online by Cambridge University Press

example, a software that has a modular structure with substantial interaction among the modules is more likely to possess dependent faults where fault classification can be done with respect to the modules. Models with dependent faults are outside the scope of this article since we basically assume that there is a random number of initial faults of only one type that behave identically and independently.

The fifth assumption states that all costs are nonnegative and it is less costly to debug during testing than after release. The fixed sequence of testing presumed in the sixth assumption indicates that testing is to be performed in a prescribed order $1,2,3,\ldots,K$ of operations with durations $t_1, t_2, t_3, \ldots, t_K$.

The testing model considered in this article provides a generalization of Özekici et al. [9] on two important issues. The failure process is not necessarily Poisson with exponential interfailure times, and debugging is not perfect. The model in Özekici et al. [9] also includes a number of test cases during which each operation creates failures with different rates. In the present setting, we set a one-to-one relationship between cases and operations so that a case is designed to test only one operation. Moreover, our model is the more attractive and realistic "common faults" model, where each fault can be accessed by a failure caused by any operation. Finally, we want to emphasize that we do not deal with finding the optimal sequence of operations to be tested and assume that a sequence is given. Our aim is to determine the optimal testing durations for the given sequence. The optimal sequencing problem is indeed very intriguing. Since there are a finite number of sequences, the best one could be picked after solving for the optimal durations for each one.

We will now derive some computational formulas on the expected number of faults removed during testing for a given operation. This will be done in the context of a generic counting process $N = \{N_t; t \geq 0\}$ with some mean value function $m(t) = E[N_t]$. This process represents the failure process caused by a fault if there is no debugging. If, however, there is debugging with success probability p, then the failure process should be stopped since there can be no more failures once the fault is removed. We will count these failures by the process $M = \{M_t; t \geq 0\}$. The conditional probability distribution of the number of failures generated by a fault can be derived as

$$P[M_t = k | N_t = n] = \begin{cases} 0 & \text{if } k = 0 \\ pq^{k-1} & \text{if } 1 \leq k \leq n-1 \\ q^{n-1} & \text{if } k = n \end{cases} \quad \textbf{(1)}$$

when $n \geq 1$. For $n = 0$, $P[M_t = 0 | N_t = 0] = 1$. From this conditional distribution, we can obtain the conditional expectation of the number of failures generated by a fault as

$$E[M_t | N_t] = \frac{1}{p}[1 - q^{N_t}] \quad \textbf{(2)}$$

https://doi.org/10.1017/S0269964801154069 Published online by Cambridge University Press

whenever $p \neq 0$. If $p = 0$, $E[M_t|N_t] = N_t$ trivially since the fault cannot be removed after a failure. Note that for $p = 1$, the expression in (2) will be equal to 1 when $N_t \geq 1$ and 0 when $N_t = 0$.

We can use (2) to obtain the expected number of failures generated as

$$E[M_t] = E[E[M_t|N_t]] = \frac{1}{p}[1 - E[q^{N_t}]] = \frac{1}{p}[1 - G(t)], \tag{3}$$

where we define $G(t) = E[q^{N_t}]$. Recalling that the expected number of inherent faults in the software at time 0 is μ, the expected total number of failures experienced up to time t is given by

$$\frac{\mu}{p}[1 - G(t)]. \tag{4}$$

After the occurrence of each failure, a debugging activity takes place to remove the fault that caused it. Each debugging activity, independent of others, is successful with probability p. Hence, given that the expected number of failures experienced is given in (4), the expected number of faults removed up to time t is

$$\mu[1 - G(t)] \tag{5}$$

and the expected number of faults that still remain is

$$\mu - \mu[1 - G(t)] = \mu G(t). \tag{6}$$

Expressions (5) and (6) are more meaningful when we note that $G(t)$ is, in fact, the probability that a fault is not successfully removed until time t. This function plays a crucial role in our analysis. If the failure times are given by a renewal process with some interrenewal distribution F, then we can write

$$G(t) = E[q^{N_t}] = \sum_{n=0}^{+\infty} q^n P[N_t = n] = \sum_{n=0}^{+\infty} q^n [F^n(t) - F^{n+1}(t)], \tag{7}$$

where F^n is the cumulative distribution function for the n-fold convolution of F by itself. Moreover, the mean value function is now the renewal function given by $m(t) = E[N_t] = \sum_{n=1}^{+\infty} F^n(t)$.

A more tractable model is obtained when N is a nonstationary Poisson process with expectation function m such that

$$G(t) = E[q^{N_t}] = \sum_{n=0}^{+\infty} q^n \frac{e^{-m(t)} m(t)^n}{n!} = e^{-pm(t)}. \tag{8}$$

The reader should note that our results apply to both renewal and nonstationary Poisson models. In fact, the counting process N is quite general and our analysis only

https://doi.org/10.1017/S0269964801154069 Published online by Cambridge University Press

requires that the function G be known. By (7) and (8), we provide two stylish choices with plausible probabilistic interpretations.

If $p = 0$, then $G(t) = 1$ and no faults are removed. In this case, (4) reduces trivially to $\mu E[N_t]$. It is also clear that $G(t) = E[q^{N_t}]$ is defined over $R_+ = [0, +\infty)$ and it is not defined for negative values of t. Therefore, in the following analysis, we suppose that properties of such functions like logarithmic convexity and monotonicity are defined over nonnegative values of the variables.

It is also clear that $G(0) = 1$ and $G(t)$ decreases in t with $G(+\infty) = 0$, provided that $p > 0$. As a matter of fact, G is a survival function and $1 - G$ is a cumulative distribution function. If T is the time at which a fault is removed, then $G(t) = P[T > t]$. This also implies that the expected time the fault is removed and failures stop is $E[T] = \int_0^{+\infty} G(t)\, dt$. These observations provide a further verification on the validity of the model in (8). Note that any survival function can be expressed as

$$G(t) = P[T > t] = e^{-H(t)}, \tag{9}$$

where H is the hazard function. So, the nonstationary Poisson model simply takes the hazard as $H(t) = pm(t)$, among many other possible choices.

Recall now that testing is done sequentially with durations $\{t_1, t_2, \ldots, t_K\}$. It is obvious that no testing should be done in operation i if $p_i = 0$ so that $t_i = 0$ optimally. We therefore assume without loss of generality that $p_i > 0$ for all i. Using the above derivations, we see that the expected number of faults remaining before testing in operation i is

$$\mu \prod_{j=1}^{i-1} G_j(t_j), \tag{10}$$

where $G_j(t_j)$ is as given in (7), with q_j and F_j inserted in place of q and F, respectively, for the renewal model, or in (8), with p_j and m_j inserted in place of p and m, respectively, for the nonstationary Poisson model. More generally, $G_j(t_j)$ is the probability that no fault is successfully removed after testing t_j time units in operation j.

The expected number of failures experienced during the ith operation is

$$\frac{\mu}{p_i}[1 - G_i(t_i)] \prod_{j=1}^{i-1} G_j(t_j) \tag{11}$$

and the expected number of faults removed during the ith operation is

$$\mu[1 - G_i(t_i)] \prod_{j=1}^{i-1} G_j(t_j). \tag{12}$$

We will now use these explicit formulas to obtain a well-defined nonlinear programming formulation.

https://doi.org/10.1017/S0269964801154069 Published online by Cambridge University Press

3. THE OPTIMIZATION PROBLEM

It is clear that for any strategy $\{t_1, t_2, \ldots, t_K\}$, the total testing cost is

$$\sum_{i=1}^{K} c_i t_i \tag{13}$$

and (11) enables us to write the expected total cost of debugging during testing as

$$\mu \sum_{i=1}^{K} \frac{d_i}{p_i} [1 - G_i(t_i)] \prod_{j=1}^{i-1} G_j(t_j). \tag{14}$$

After the testing period, we know that the software will be released and each fault will create a failure during operation i with probability π_i. Using (12), the expected number of faults still remaining in the software after testing is

$$\mu \left\{ 1 - \sum_{i=1}^{K} [1 - G_i(t_i)] \prod_{j=1}^{i-1} G_j(t_j) \right\}, \tag{15}$$

so that the expected total cost of debugging after release is

$$\mu \sum_{n=1}^{K} \pi_n e_n \left\{ 1 - \sum_{i=1}^{K} [1 - G_i(t_i)] \prod_{j=1}^{i-1} G_j(t_j) \right\}. \tag{16}$$

Combining (13), (14), and (16), the expected total cost is

$$\sum_{i=1}^{K} c_i t_i + \mu \sum_{n=1}^{K} \pi_n e_n \left\{ 1 - \sum_{i=1}^{K} [1 - G_i(t_i)] \prod_{j=1}^{i-1} G_j(t_j) \right\}$$
$$+ \mu \sum_{i=1}^{K} \frac{d_i}{p_i} [1 - G_i(t_i)] \prod_{j=1}^{i-1} G_j(t_j). \tag{17}$$

To simplify this cost function, we set $e = \sum_{n=1}^{K} \pi_n e_n$ and $b_i = \mu(e - (d_i/p_i))$. There are some interesting economic interpretations for these terms. Each remaining fault is eventually debugged successfully after a failure of operation n at cost e_n and with probability π_n. It is, therefore, clear that e is, in fact, the expected cost of debugging a fault after release. During testing operation i, it takes, on the average, $1/p_i$ trials to successfully remove a fault. The expected cost of removing a fault during testing in operation i is then d_i/p_i and the difference $e - (d_i/p_i)$ represents the benefit of testing operation i. Finally, b_i is the total benefit obtained through testing and removing all faults in operation i as opposed to removing them after release.

https://doi.org/10.1017/S0269964801154069 Published online by Cambridge University Press

After some manipulations, (17) can be rewritten equivalently as

$$\sum_{i=1}^{K}\left[c_i t_i + (b_i - b_{i+1})\prod_{j=1}^{i} G_j(t_j)\right] + \mu e - b_1, \tag{18}$$

where we set $b_{K+1} = 0$.

This leads to the nonlinear programming formulation $\min_{t_1, t_2, \ldots, t_K \geq 0} L(t_1, t_2, \ldots, t_K)$ with

$$L(t_1, t_2, \ldots, t_K) = \sum_{i=1}^{K}\left[c_i t_i + (b_i - b_{i+1})\prod_{j=1}^{i} G_j(t_j)\right], \tag{19}$$

and the constant term $\mu e - b_1$ is left out because it does not affect the decision variables. Since the expected total cost is μe if no testing is done, $b_1 - L(t_1, t_2, \ldots, t_K)$ is the total benefit obtained through the testing strategy $\{t_1, t_2, \ldots, t_K\}$.

Although we do not necessarily have an analytical closed-form expression for the optimal solution, the special form of the objective function can be exploited through convexity arguments to compute an optimal solution efficiently. The following proposition states why. Recall that each G_j is defined on R_+ and takes values in $[0,1]$.

PROPOSITION 2: *Suppose that $b_1 \geq b_2 \geq b_3 \geq \cdots \geq b_K$ and G_j is log-convex for all $j = 1,2,\ldots,K$, then L is convex.*

PROOF: The linear function $c_i t_i$ in (19) is convex. The term $(b_i - b_{i+1})\Pi_{j=1}^{i} G_j(t_j)$ is also convex since $b_i \geq b_{i+1}$, the product of log-convex functions is log-convex, and log-convexity implies convexity. The convexity of L now follows since the sum of convex functions is also convex. The reader is referred to Klinger and Mangasarian [6] for details of these arguments. ■

If the failure process is the nonstationary Poisson process in (8), then $\log G_j(t_j) = -p_j m_j(t_j)$. It follows that G_j is log-convex if and only if the mean value function m_j is concave. This implies that the intensity, or failure arrival rate, should be decreasing in time. A similar observation holds for the more general characterization (9) using the hazard function. This implies $\log G_j(t_j) = -H_j(t)$, where H_j is the hazard function during operation j corresponding to the distribution of T, the time at which a fault is successfully removed. Therefore, log-convexity holds if and only if T has a decreasing failure rate (DFR) distribution.

As a consequence of Proposition 2, the nonlinear programming formulation reduces to the minimization of a convex function with nonnegativity constraints. This can be solved very efficiently by one of the known methods discussed, for example, in Bazaraa et al. [2]. The objective function L in (19) has explicit parameters, given by the costs $\{c_n\}$ and total benefits $\{b_n\}$, and implicit parameters, which are specific to the survival functions $\{G_j(t_j)\}$. Their relation, as well as the form of $\{G_j(t_j)\}$, are of great importance and the structure of the optimal solution depends on

https://doi.org/10.1017/S0269964801154069 Published online by Cambridge University Press

their properties. We present some useful results along this direction in the following sections.

4. THE EXPONENTIAL MODEL

Suppose now that the time to failure has an exponential distribution with rate λ_i during operation i so that the failure process is an ordinary Poisson process. This implies that

$$G_j(t_j) = e^{-\lambda_j p_j t_j} \tag{20}$$

and the objective function simplifies to

$$L(t_1, t_2, \ldots, t_K) = \sum_{i=1}^{K} \left[c_i t_i + (b_i - b_{i+1}) \exp\left(-\sum_{j=1}^{i} \lambda_j p_j t_j \right) \right]. \tag{21}$$

This is, in fact, almost identical to the one given by (23) in Özekici et al. [9], where there is perfect debugging (i.e., $p_i = 1$). The difference is that the failure rates $\{\lambda_i\}$ are replaced by fault removal rates $\{\lambda_i p_i\}$. It is also clear that the G_j is not only log-convex but also convex for all j. Therefore, if $b_1 \geq b_2 \geq b_3 \geq \cdots \geq b_K$, convexity of L holds and Theorem 2 in Özekici et al. [9] can be used to obtain the explicit optimal solution

$$t_i^* = \frac{1}{\lambda_i p_i} \ln \left[\left(\frac{b_i - b_{i+1}}{a_i - a_{i+1}} \right) \left(\frac{b_{i-1} - b_i}{a_{i-1} - a_i} \right)^{-1} \right], \tag{22}$$

where $a_i = c_i / \lambda_i p_i$ for all $i = 1, 2, \ldots, K$ provided that $a_1 > a_2 > a_3 > \cdots > a_K$ and

$$1 < \frac{b_1 - b_2}{a_1 - a_2} < \frac{b_2 - b_3}{a_2 - a_3} < \cdots < \frac{b_K - b_{K+1}}{a_K - a_{K+1}}. \tag{23}$$

The boundary values are taken to be $a_0 = a_1 + 1$, $b_0 = b_1 + 1$, and $a_{K+1} = b_{K+1} = 0$ to complete expression (22) for $i = 1$ and K. The reader is referred to the proof, discussions, and economic interpretations presented in Özekici et al. [9]. The results outlined in (20)–(23) extend their results when debugging is not perfect. This generalization is achieved simply through the use of fault removal rates rather than the failure rates.

We will now give a numerical illustration of the exponential model. We can assume, without loss of generality, that our model consists of $K = 3$ operations so that $\lambda_i p_i = 1$ for simplification and $c_1 = 3$, $c_2 = 2$, $c_3 = 1$, $b_1 = 15$, $b_2 = 12$, and $b_3 = 8$. The objective function becomes

$$L(t_1, t_2, t_3) = 3t_1 + 2t_2 + t_3 + 3e^{-t_1} + 4e^{-(t_1 + t_2)} + 8e^{-(t_1 + t_2 + t_3)}. \tag{24}$$

https://doi.org/10.1017/S0269964801154069 Published online by Cambridge University Press

All conditions required above are satisfied and the optimal solution is given by the formulas

$$t_1^* = \ln\left[\frac{b_1 - b_2}{a_1 - a_2}\right], \qquad t_2^* = \ln\left[\left(\frac{b_2 - b_3}{a_2 - a_3}\right)\left(\frac{b_1 - b_2}{a_1 - a_2}\right)^{-1}\right],$$
$$t_3^* = \ln\left[\left(\frac{b_3 - b_4}{a_3 - a_4}\right)\left(\frac{b_2 - b_3}{a_2 - a_3}\right)^{-1}\right], \tag{25}$$

so that

$$t_1^* = \ln\left[\frac{15 - 12}{3 - 2}\right], \qquad t_2^* = \ln\left[\left(\frac{12 - 8}{2 - 1}\right)\left(\frac{15 - 12}{3 - 2}\right)^{-1}\right],$$
$$t_3^* = \ln\left[\left(\frac{8}{1}\right)\left(\frac{12 - 8}{2 - 1}\right)^{-1}\right] \tag{26}$$

or

$$t_1^* = 1.0986, \qquad t_2^* = 0.2877, \qquad t_3^* = 0.6932. \tag{27}$$

Using (21), we compute $L^* = 7.5643$, and the total benefit obtained through testing is $b_1 - L^* = 15 - 7.5643 = 7.4357$. Özekici et al. [9] discussed how the optimal solution can be computed for models that do not satisfy the conditions on $\{a_n, b_n\}$.

5. THE GENERAL NONEXPONENTIAL MODEL

If the failure time distribution is not exponential, then we do not necessarily have a closed-form optimal solution, as in the previous section. However, we suppose that the conditions of Proposition 2 are true, so that the objective function L is convex. The final convex program can be solved easily. In addition, further assumptions on the relations between the parameters of the objective function L results in a simple search method. Suppose that each G_j has a derivative denoted by G_j' and a second derivative denoted by G_j''.

The Karush–Kuhn–Tucker conditions state that if there are values $t_1^* \geq 0$, $t_2^* \geq 0, \ldots, t_K^* \geq 0$ of the decision variables $\{t_2, t_2, \ldots, t_K\}$ and values $\alpha_1^* \geq 0$, $\alpha_2^* \geq 0, \ldots, \alpha_K^* \geq 0$ of Lagrange multipliers $\{\alpha_1, \alpha_2, \ldots, \alpha_K\}$ such that

$$c_n + \sum_{j=n}^{K} (b_j - b_{j+1}) G_n'(t_n^*) \prod_{m=1,\, m\neq n}^{j} G_m(t_m^*) - \alpha_n = 0 \tag{28}$$

and

$$\alpha_n t_n^* = 0 \tag{29}$$

https://doi.org/10.1017/S0269964801154069 Published online by Cambridge University Press

for all $n = 1,2,\ldots,K$, then $\{t_1^*, t_2^*, \ldots, t_K^*\}$ is a stationary policy. In fact, it is an optimal policy because of the convexity of the objective function L in (19) as a consequence of the log-convexity assumption on G_j.

If the optimal policy is such that $t_n^* > 0$ for all operations, then $\alpha_n^* = 0$ and the condition (28) can be expressed as

$$\sum_{j=n}^{K} (b_j - b_{j+1}) \prod_{m=1}^{j} G_m(t_m^*) = h_n(t_n^*) \tag{30}$$

for $n = 1,2,\ldots,K$, where

$$h_n(t_n) = -\frac{c_n G_n(t_n)}{G_n'(t_n)}. \tag{31}$$

This implies that if the system of equations in (30) has a positive solution, then it must be optimal. Rewriting (30) for $n + 1$, we have

$$\sum_{j=n+1}^{K} (b_j - b_{j+1}) \prod_{m=1}^{j} G_m(t_m^*) = h_{n+1}(t_{n+1}^*) \tag{32}$$

and the difference of (30) and (32) gives the recursive relation

$$h_{n+1}(t_{n+1}^*) = h_n(t_n^*) - (b_n - b_{n+1}) \prod_{m=1}^{n} G_m(t_m^*) \tag{33}$$

for $n = 1,2,\ldots,K-1$. For $n = K$, condition (30) becomes

$$h_K(t_K^*) = b_K \prod_{m=1}^{K} G_m(t_m^*) \tag{34}$$

or

$$\Delta = h_K(t_K^*) - b_K \prod_{m=1}^{K} G_m(t_m^*) = 0. \tag{35}$$

Note from the special form of (33) that if $\{t_1^*, t_2^*, \ldots, t_n^*\}$ are known, then we can solve for t_{n+1}^*. This procedure can then be implemented as an interval bracketing method which searches for a positive value for t_1 such that condition (35) is satisfied. In the determination of the new interval, the facts that Δ is a nondecreasing function of t_1 and $h_j(t_j)$ is a nondecreasing function of t_j are used. The first property follows directly since each $G_j(t_j)$ is decreasing in t_j. The second property is shown next.

PROPOSITION 3: *Suppose that if G_j is log-convex, then $h_j = -c_j G_j / G_j'$ is nondecreasing.*

https://doi.org/10.1017/S0269964801154069 Published online by Cambridge University Press

PROOF: Since G_j is log-convex, the second derivative

$$\frac{d^2 \log G_j(t_j)}{dt_j^2} = \frac{G_j''(t_j)G_j(t_j) - (G_j'(t_j))^2}{(G_j(t_j))^2} \tag{36}$$

is nonnegative. This implies that $G_j''(t_j)G_j(t_j) \geq (G_j'(t_j))^2$, which completes the proof since

$$h_j'(t_j) = -c_j \frac{(G_j'(t_j))^2 - G_j(t_j)G_j''(t_j)}{(G_j'(t_j))^2} \geq 0 \tag{37}$$

and h_j is nondecreasing. ■

The pseudocode of the algorithm, which bisects the search intervals, is given below. The algorithm assumes that the initial search interval of t_1 contains $t_1^* > 0$ for which Δ is 0 and forces $\{t_n; n = 1,2,\ldots,K\}$ to remain nonnegative during the iterations by simply setting it to 0 when it has a negative value.

Algorithm

Input: Initial interval $[a,b]$ *for* t_1, *final precision* ϵ.
Output: $t_1^* > 0$, $t_2^* > 0, \ldots, t_K^* > 0$.
begin
  repeat
    Set $t_1 = \max\{0,(a+b)/2\}$;
    For $n = 1$ *to* $K-1$, *solve (33) to compute* t_{n+1} *and set* $t_{n+1} = 0$ *if* $t_{n+1} < 0$;
    Compute Δ *by using (35);*
    If $\Delta < 0$ *then set* $a = t_1$, *otherwise set* $b = t_1$;
  until $|\Delta| < \epsilon$
end;

Observe the use of Proposition 3 when combined with recursive relations in (33); an increase (decrease) in the values of t_1 results in an increase (decrease) in the values of $t_2, t_3, \ldots, t_K$. In other words, the remaining variables follow t_1 because of the propagating change through relations in (33).

To demonstrate the use of these results, we will discuss the following nonexponential model with

$$G_j(t_j) = e^{-\lambda_j p_j t_j^{1/n}} \tag{38}$$

for $n \geq 1$. In view of (8), this is a nonstationary Poisson model with $m_j(t_j) = \lambda_j t_j^{1/n}$, where the expected time to fault removal is

$$E[T] = \int_0^{+\infty} e^{-\lambda_j p_j t^{1/n}}\, dt = \frac{n!}{(\lambda_j p_j)^n}. \tag{39}$$

https://doi.org/10.1017/S0269964801154069 Published online by Cambridge University Press

Note that if $n = 1$, we obtain the ordinary Poisson or exponential model in Section 4. It is clear that $G_j(t_j)$ is log-convex and

$$h_j(t_j) = \frac{nc_j}{\lambda_j p_j} t_j^{(n-1)/n} \tag{40}$$

is increasing in t_j.

Consider a numerical problem that has the same cost parameters as in Section 4 with $n = 2$ and $K = 3$ operations so that $\lambda_j p_j = 1$ for simplification and $c_1 = 3$, $c_2 = 2$, $c_3 = 1$, $b_1 = 15$, $b_2 = 12$, and $b_3 = 8$. The objective function now becomes

$$L(t_1, t_2, t_3) = 3t_1 + 2t_2 + t_3 + 3e^{-\sqrt{t_1}} + 4e^{-(\sqrt{t_1}+\sqrt{t_2})} + 8e^{-(\sqrt{t_1}+\sqrt{t_2}+\sqrt{t_3})}. \tag{41}$$

The optimality conditions (33) and (34) can be written as

$$4\sqrt{t_2} = 6\sqrt{t_1} - 3e^{-\sqrt{t_1}}, \tag{42}$$

$$2\sqrt{t_3} = 4\sqrt{t_2} - 4e^{-(\sqrt{t_1}+\sqrt{t_2})}, \tag{43}$$

$$2\sqrt{t_3} = 8e^{-(\sqrt{t_1}+\sqrt{t_2}+\sqrt{t_3})}, \tag{44}$$

and solving this system of equations by Maple, we obtain the optimal solution

$$t_1^* = 0.43043, \qquad t_2^* = 0.35395, \qquad t_3^* = 0.38124. \tag{45}$$

The optimal objective value of (41) is $L^* = 6.3168$ and the total benefit of testing is $b_1 - L^* = 15 - 6.3168 = 8.6832$. Note that although the parameters are identical, the optimal solutions for the exponential and nonexponential models are substantially different due to the different distributions.

The recursive procedure suggested by (33) can also be used. Given an initial search interval including $t_1^* = 0.43043$, one can easily generate t_2^* and t_3^* by solving (42) and (43) iteratively. This is essentially what the Algorithm does. If we find such a nonnegative solution that also satisfies (44), then it will be optimal. Note that such a procedure can be applied to any arbitrary case since (33) and (34) are general results.

As an illustration, we generated a sequence $\{t_1(k), t_2(k), t_3(k); k = 1,2,\ldots,10\}$ of solutions using the Algorithm with precision $\epsilon = 0.001$ by taking $[0, 0.5]$ as the initial search interval for t_1 and defining

$$\Delta(k) = 2\sqrt{t_3(k)} - 8e^{-(\sqrt{t_1(k)}+\sqrt{t_2(k)}+\sqrt{t_3(k)})} \tag{46}$$

as the difference in (35) or (44). The results are summarized in Table 1.

Although it is not possible to formulate the interaction between the parameters explicitly, such as inequalities (23) given for the exponential case, it is still possible

https://doi.org/10.1017/S0269964801154069 Published online by Cambridge University Press

Table 1. Iteration Results of the Algorithm

k	$t_1(k)$	$t_2(k)$	$t_3(k)$	$\Delta(k)$
1	0.375000	0.262155	0.140113	−1.038728
2	0.437500	0.366116	0.417995	0.112001
3	0.406250	0.313106	0.265101	−0.414543
4	0.421875	0.339371	0.338419	−0.140774
5	0.429688	0.352686	0.377465	−0.011942
6	0.433594	0.359387	0.397549	0.050620
7	0.431641	0.356033	0.387461	0.019489
8	0.430664	0.354358	0.382452	0.003811
9	0.430176	0.353522	0.379956	−0.004056
10	0.430420	0.353940	0.381203	−0.000120

to make some observations. First, as a consequence of relations (33) and (34), it is possible to write

$$G_1(t_1) = \frac{h_1(t_1) - h_2(t_2)}{b_1 - b_2} \tag{47}$$

and

$$G_j(t_j) = \frac{h_j(t_j) - h_{j+1}(t_{j+1})}{b_j - b_{j+1}} \frac{b_{j-1} - b_j}{h_{j-1}(t_{j-1}) - h_j(t_j)} \tag{48}$$

for $j = 2,3,\ldots,K$ with $b_{K+1} = h_{K+1}(t_{K+1}) = 0$. The combination of these inequalities with the fact that $0 < G_j(t_j) < 1$ for $t_j > 0$ gives

$$1 < \frac{b_1 - b_2}{h_1(t_1) - h_2(t_2)} < \frac{b_2 - b_3}{h_2(t_2) - h_3(t_3)} < \cdots < \frac{b_K - b_{K+1}}{h_K(t_K) - h_{K+1}(t_{K+1})}, \tag{49}$$

which is identical to (23) for $G_j(t_j) = e^{-\lambda_j p_j t_j}$. In addition, $h_j(t_j) > h_{j+1}(t_{j+1})$ for $j = 1,2,\ldots,K-1$ follows since $b_1 \geq b_2 \geq \cdots \geq b_K$ and $G_j(t_j) > 0$.

Coming back to our numerical example, for $t_1^* = 0.43043$, $t_2^* = 0.35395$, and $t_3^* = 0.38124$, we have $h_1(t_1^*) = 3.93643$, $h_2(t_2^*) = 2.37975$, $h_3(t_3^*) = 1.23489$, and

$$\frac{15-12}{3.93643-2.37975} = 1.92718 < \frac{12-8}{2.37975-1.23489} = 3.49388 < \frac{8}{1.23489} = 6.47831, \tag{50}$$

so that inequalities (49) are satisfied. In other words, the Algorithm computes an optimal policy if the parameters guarantee the existence of a policy which satisfies (49) and $h_j(t_j) > h_{j+1}(t_{j+1})$ for $j = 1,2,\ldots,K-1$.

https://doi.org/10.1017/S0269964801154069 Published online by Cambridge University Press

For our numerical example, if $c_1 = 0.1$, $c_2 = 1$, $c_3 = 5$, $b_1 = 29$, $b_2 = 14$, and $b_3 = 3$, then the Algorithm fails to converge. However, there exists an optimal solution. We computed $t_1^* = 13.01933$, $t_2^* = 0.02598$, and $t_3^* = 0.0$ with $L^* = 2.05709$ by using Lingo. This results in $h_1(t_1^*) = 0.72165$, $h_2(t_2^*) = 0.32237$, and $h_3(t_3^*) = 0.0$ and we obtain

$$\frac{29-14}{0.72165-0.32237} = 37.5676 > \frac{14-3}{0.32237-0.0} = 34.1223 < \frac{3}{0.0} = -\infty, \tag{51}$$

which clearly violates conditions (49). This is quite supportive since the Algorithm assumes the existence of positive optimal values. This is not the case here since $t_3^* = 0$ and the corresponding Lagrange multiplier α_3^* is allowed to be positive; for example, Lingo computes $\alpha_3^* = 0.57928$. Similar observations can be made for other datasets which also violate inequalities (49).

Acknowledgments

This research is supported by the National Science Foundation grant INT-9602081 and Boğaziçi University Research Fund grant 97HA303.

References

1. Bai, D.S. & Yun, W.Y. (1988). Optimum number of errors corrected before releasing a software system. *IEEE Transactions on Reliability* 37: 41–44.
2. Bazaraa, M.S., Sherali, H.D., & Shetty, C.M. (1993). *Nonlinear programming.* New York: Wiley.
3. Forman, E.H. & Singpurwalla, N.D. (1979). Optimal time intervals for testing hypotheses on computer software. *IEEE Transactions on Reliability* 28: 250–253.
4. Goel, A. & Okumoto, K. (1980). Optimal release time for software systems based on reliability and cost criteria. *Journal of Systems and Software* 1: 315–318.
5. Jelinski, Z. & Moranda, P. (1972). Software reliability research. In W. Freiberg (ed.), *Statistical computer performance evaluation.* New York: Academic Press, pp. 465–484.
6. Klinger, A. & Mangasarian, O.L. (1968). Logarithmic convexity and geometric programming. *Journal of Mathematical Analysis and Applications* 24: 388–408.
7. Musa, J.D. (1993). Operational profiles in software reliability engineering. *IEEE Software* 10: 14–32.
8. Musa, J.D. (1996). The operational profile. In S. Özekici (ed.), *Reliability and maintenance of complex systems*. NATO ASI Series Volume F154. Berlin: Springer-Verlag, pp. 332–343.
9. Özekici, S., Altınel, İ.K., & Özçelikyürek, S. (2000). Testing of software with an operational profile. *Naval Research Logistics* 47: 620–634.
10. Özekici, S. & Çatkan, N.A. (1992). Software release models: A survey. In A.E. Harmancı and E. Gelenbe (eds.), *Proceedings of the 5th International Symposium on Computer and Information Sciences (ISCIS V)*, pp. 203–210.
11. Özekici, S. & Çatkan, N.A. (1993). A dynamic software release model. *Computational Economics* 6: 77–94.
12. Özekici, S. & Soyer, R. (2001). Bayesian testing strategies for software with an operational profile. To appear in *Naval Research Logistics*.
13. Ross, S.M. (1985). Software reliability: The stopping rule problem. *IEEE Transactions on Software Engineering* 11: 1472–1475.

https://doi.org/10.1017/S0269964801154069 Published online by Cambridge University Press