Reliability-Estimation & Stopping-Rules for Software Testing, Based on Repeated Appearances of Bugs

Mark C. K. Yang
University of Florida, Gainesville
Anne Chao
National Tsing Hua University, Hsin Chu

Key Words — Software reliability, software testing, stopping rules in testing

Summary & Conclusions — Software-testing (debugging) is one of the most important components in software development. An important question in the debugging process is, when to stop. The choice is usually based on one of two decision criteria:

1. when the reliability has reached a given threshold,
2. when the gain in reliability cannot justify the testing cost.

Various stopping rules and software reliability models are compared by their ability to deal with these two criteria. Two new stopping rules, initiated by theoretical study of the optimal stopping rule based on cost, are more stable than other rules for a large variety of bug structures. The 1-step-ahead stopping rules based on the Musa et al basic execution and logarithmic Poisson models, as well as the stopping rule by Dalal & Mallows (1990), work well for software with many relatively small bugs (bugs with very low occurrence rates). The comparison was done by simulation.

1. INTRODUCTION

Acronyms

LP Musa et al logarithmic Poisson model
SR stopping rule

Software testing, or debugging, is one of the most important components in software development. In many projects, the time used in debugging can be around 50% of the total development effort [17]. An important question is when to stop the debugging process. Two obvious criteria are:

1. when the reliability has reached a given threshold,
2. when the gain in reliability cannot justify the testing cost.

To use criterion #1, we need to predict the remaining (future) failure rate of the software if testing is stopped. To use criterion #2, in addition to the prediction requirement, we need to identify a monetary cost equivalent for failures encountered by the user. Cost estimation is presented in Sherer [23] and Boehm [2]. This paper assumes that the monetary equivalent of future failure has been established.

Prediction is always risky, especially when humans are involved. There are models that count the lines of code to predict the number of bugs in a software program (Lipow [12] and Brown, Maghsoodloo, Deason [4]). However, a well accepted formula has not been found, and it is unlikely that one bug-prediction formula, without referring to the nature of the software and the programmer skills, can cover the large variation in software development. A bug represents a defect component that will cause some type of failure or incorrect output with certain inputs. Sometimes fault or error is used for the term bug. The definitions can be different, but it is difficult to make them precise. They are approximations to a similar phenomenon and cause no difference in the mathematical aspect of software reliability.

Reliability prediction usually relies on the testing history of the software. In order to predict the future from the past, one needs a model to connect them. But so far, there is no simple way to identify a correct model for this connection. Nevertheless, some of the models fit a particular piece of software very well. When all the bugs are assumed to be the same size (measured by the failure rate), it becomes the Musa et al basic model [16] or the Goel-Okumoto model [9]. This same model is also known as the Jelinski & Moranda [11] model in statistical literature. When the bugs are not assumed to be the same size, a structure of the bug failure rates can be modeled, eg, the Littlewood-Verrall model [13, 14], in which a Gamma-prior is assigned to the failure distribution. So far, neither the equal size nor the Gamma-prior models have any theoretical justification. Thus, some researchers would rather fit the testing history (total faults encountered vs time) directly by reasonable curves. Among these empirical fits are the Musa et al logarithmic Poisson (LP) model [16], the Yamada et al Weibull model [24], and many other models (see [16: chapter 11]).

Once a model is preassigned, one can predict the future reliability if there are enough past data to estimate the parameters in the model. However, the choice of the best model usually fluctuates as observations continue. Thus, it is of both theoretical & practical interest to develop reliability estimation methods and SR that do not depend on a particular model or curve. This paper constructs two SR without assuming any distribution of the bug sizes. The idea is based on the similarity between the undiscovered bugs in software and the undiscovered species in an ecological system. This analogy has been used by Nayak [18] who proposed a recapture debugging process to get extra information for estimating the total number of bugs and for constructing a SR. While the Nayak procedure still depends on the equal failure rate assumption, our new SR are very general. Moreover, our new methods depend less on the recapture requirement. Here, recapture means that a bug can be observed more than once in the testing process. If a software bug, once discovered, is removed immediately, recapture debugging...
becomes impossible. But in many testing schemes, testing & debugging are not performed simultaneously. Thus, in the testing phase, the same bug can cause several failures, ie, the same bug can be observed repeatedly. Some recent developments in software engineering also advocate the separation of code writing, testing, and debugging. This method, usually referred to as the cleanroom procedure, has been shown to be better than the traditional simultaneous writing, testing, and debugging scheme in many small software development experiments; see Mills, Dyer, Linger [15] and Selby, Basili, Baker [22]. Recapture debugging also occurs when the software is already released to the public or to a group of trial customers. None of the traditional testing models can use the information of the repeated bug appearances. It seems now necessary to investigate the merit of recapture debugging in software testing.

The testing cases in the cleanroom procedure are chosen as a random sample for possible future inputs. Testing which uses this type of input is usually referred to as random testing. Its advantages & limitations have been discussed by Duran & Ntafos [8]. This is the only type of testing considered in this paper.

Nomenclature

• Encounter. A bug is encountered (in a case in a round) iff it: 1) causes a failure in that case in that round, and 2) is identified as the cause of that failure.
• Case. A single test of the software under specified conditions.
• Round. A set of a specified number of contiguous cases.
• Singleton. A bug that has been encountered exactly once in a specified period.
• Doubleton. A bug that has been encountered exactly twice in a specified period.
• Test size. Total number of cases tested.

Notation

\( S_n, B_n \) number of [singletons, doubletons] discovered during rounds 1 through \( n \) inclusive
\( s_n, b_n \) number of [singletons, doubletons] discovered during round \( n \)
\( c_1 \) cost of testing 1 case
\( c_2 \) cost to the software producer for 1 failure after the release of the software
\( c \) \( M \cdot c_2/c_1 \)
\( S(\cdot) \) indicator function: \( S(\text{True}) = 1, S(\text{False}) = 0 \)
\( M \) \( s \)-expected number of cases to be used by the customers
\( m \) initial number of bugs in a software
\( N \) number of cases in 1 round; the stopping decision is made only at the end of each round, not during the round
\( n \) implies round \# \( n \)
\( T_q, R \) [initial, remaining] bug-encounter-probability of the software
\( X_i(n) \) number of times that bug \( i \) is encountered at the end of round \( n \)

\( q_i, p_i \) [encounter, non-encounter] probability (per case) for bug \( i \), \( i=1,2,...,m; p_i + q_i = 1 \)
\( N_0 \) initial test size

\[ \sum_i \sum_{i=1}^m \]

Other, standard notation is given in "Information for Readers & Authors" at the rear of each issue.

Assumptions

1. Testing is performed in rounds; in each round, equal numbers of cases are tested. Failures are recorded, but not investigated, during each round.
2. The \( q_i \) is a constant from case-to-case and from round-to-round. For section 2.1 only, the \( q_i \) are known.
3. For sections 2.2 & 2.3 only, there are an unknown number, \( m \), of bugs, each with unknown \( q_i \), in the software before testing.
4. The bugs act s-independently.
5. Non-encountered bugs are never removed. When bugs are removed, removal is perfect, ie, a) removal is certain, b) no new bugs are inserted, and c) the \( q_i \) for non-removed bugs are not affected in any way.
6. Either of the following 2 debugging models can be used.
   a. Recapture debugging model: encountered bugs are not removed until the release of the software.
   b. Usual debugging model: encountered bugs are removed after each round.

A consequence of these assumptions is that the failures are modeled by a discrete-time non-homogeneous Poisson process with a step-function failure intensity.

2. COST LOWER-BOUND AND NEW STOPPING RULES

2.1 Perspective

Assumption #2, in more detailed mathematical terms, is:

\( q_1 \geq q_2 \geq ... \geq q_m > 0 \)

(1)

\[ T_q = \sum_i q_i; \]

\( R < T_q \) if & when bugs are removed.

Though there is little difference in measuring the "time" axis by execution duration or cases, we use cases as the "time" unit. Assumption #2, in (1), is very general and our first goal is to find the lower bound for testing cost under any SR. It is well known that the optimal SR depends on the prior knowledge of (1) [21]. For example, if we know that \( m=0 \), then any testing is a waste. Because we have argued previously that no modeling of (1) can be justified, assumption #2 states that all the \( q_i \) are known; but it takes testing to find these bugs and to remove them. This section shows that under assumption #2, the optimal...
SR and its cost can be found. This cost is the lower bound. If the cost of any SR is close to this bound, then improvement becomes unnecessary for this SR. All three numbers, \( c_1, c_2, M \), are necessary in order to balance the testing cost and failure penalty.

Appendix A shows that the optimal SR is to stop at the first \( n \) such that:

\[
\sum_i q_i \cdot (1-p_i^n) \cdot g(X_i(n)=0) \leq N/c. \tag{2}
\]

There is no simple formula for the optimal cost, but it can be found easily by simulation based on the optimal SR.

### 2.2 Recapture Debugging (Assumptions #3 & #6a)

Since the \( q_i \) are unknown, the SR in (2) cannot be used in practice. But (2) tells us that if there were a good estimate of the l.h.s of the inequality, we could construct a close-to-optimal SR. Since the \( q_i < < 1 \), a good approximation for (2) is:

\[
\sum_i q_i^2 \cdot g(X_i(n)=0) \leq 1/c. \tag{3}
\]

#### Notation

\( \theta(n) = E\left\{ \sum_i q_i^2 \cdot g(X_i(n)=0) \right\} \).

\( \theta(n) = \sum_i q_i^2 \cdot p_i^n \cdot N. \tag{4a} \)

It is known [3, 5] that:

\[
E\left\{ \sum_i g(X_i(n)=2) \right\} = \binom{n^*}{2} \cdot \sum_i q_i^2 \cdot p_i^{n^*-2}. \tag{4b}
\]

Since \( p_i^{n^*-2} = p_i^{n^*} \),

\[
\theta = \left[ \sum_i g(X_i(n)=2) \right] / \binom{n^*}{2} = B_n / \binom{n}{2}. \]

Thus, a reasonable adaptive SR for recapture debugging is — Stop at the first \( n \) such that:

\[
B_n / \binom{n}{2} \leq 1/c. \tag{5}
\]

#### 2.3 Usual Debugging (Assumptions #3 & #6b)

The number of doubletons needs to be estimated. By the simple equality: \( B_n = \text{Bugs in } B_{n-1} \text{ are not encountered in round } n \) + \( \text{Bugs in } S_{n-1} \text{ are encountered exactly once in round } n \) + \( b_n \), and by the maximum-likelihood principle, recursive estimates are:

\[
\hat{B}_n = \hat{B}_{n-1} \cdot (1-2p)^N + \hat{S}_{n-1} \cdot (1-p)^{N-1}/(n-1) + b_n; \tag{6}
\]

\[
\hat{S}_n = \hat{S}_{n-1} \cdot (1-2p)^N + s_n; \tag{7}
\]

\( \nu = 1/[N \cdot (n-1)] \).

Thus, an adaptive SR for this model is:

Stop at the first \( n \) such that:

\[
\hat{B}_n / \binom{n}{2} \leq 1/c. \tag{8}
\]

Eq (5) & (8) are our two adaptive SR.

### 2.4 General Discussion

Data with recapture information can also be used to estimate \( R \). According to the undiscovered species theory [5], the maximum-likelihood estimates for \( R \) under the recapture-debugging SR and usual-debugging SR are, respectively:

\[
\hat{R}_{\text{rec}} = S_n / (n \cdot N), \tag{9a}
\]

\[
\hat{R}_{\text{usa}} = \hat{S}_n / (n \cdot N). \tag{9b}
\]

They are useful because \( R \) is also one of the main concerns in software testing.

Adaptive SR (5) & (8) cannot be as efficient as (3) due to the large gap in prior information on the \( q_i \). For example, if we know that the software has no bugs, then by (2), the optimal SR is not to test it at all. But when this fact is unknown, it would take a lot of testing to find it out. However, the inferiority of adaptive SR (5) & (8) to SR (3) does not imply that they are inferior to any other practical SR, because all of them are in the same blind situation on the distribution of the \( q_i \). Our new adaptive SR and the other old SR are compared in section 3 by a simulation study, but here we investigate the conditions under which the adaptive SR (5) should be close to (3). Adaptive SR (8) is not investigated here due to its mathematical complexity.

The difference between the l.h.s of (3) & (5) is:

\[
\Delta(n) = \sum_i q_i^2 \cdot g(X_i(n)=0) - B_n / \binom{n}{2}. \tag{10a}
\]

Appendix B shows that:

\[
E\{\Delta(n)\} \approx 0, \tag{10b}
\]

\[
\text{Var}\{\Delta(n)\} = \sum_i q_i^2 \cdot p_i^{n-N} \cdot \left[ q_i^2 + 1 / \binom{n}{2} \right], \tag{10c}
\]

\[
\text{StdDev}\{\Delta(n)\} = \sqrt{\text{Var}\{\Delta(n)\}}. \tag{11}
\]

Since \( \text{Var}\{\Delta(n)\} \to 0 \) as \( n \to \infty \), this approximation is asymptotically \( s \)-consistent. For smaller \( n \), we use the coefficient of variation,

\[
\text{CV}(n) = \text{StdDev}\{\Delta(n)\} / \theta(n). \tag{11}
\]
to judge the goodness of the approximation. This criterion is reasonable because if the error is much smaller than the mean value, the error is not serious. Eq (11) is calculated from (4a) & (10b).

Since the decision rule is at time \( \theta(n) = 1/c \), the \( c \) is in a desirable range if,

\[
CV(n) \leq 0.5, \text{ for } \theta(n) = 1/c.
\]

(12)

Though this range cannot be computed without knowledge of the \( q_i \), it can be easily checked for any given distribution of the \( q_i \). As demonstrated in section 3, adaptive SR (5) is usually good for a large range of \( c \).

### 3. COMPARISON OF STOPPING RULES

One of the most convincing methods to compare SR is to use them in real software testing and see which one saves the most money. This is obviously unrealistic, because it is unlikely that any company is willing to release multiple versions of a software product with different SR. Thus, we have to use simulation. There are few existing records on the distribution of bug \( q_i \) in software. Actually, if a bug is immediately removed from the software at its discovery during testing, its \( q_i \) cannot be estimated. The only record we can find is from the nine IBM products in Adams [1: table 2]. All of the distributions are similar. We used the following \( q_i \) distributions and parameter values therein.

A. Rapidly decreasing \( q_i \) (in exponential rate)

\[
q_i = K \cdot e^{-\alpha i}, \text{ for } \alpha \in (0, 1); \alpha = 0.8, 0.7, 0.5;
\]

B. Moderately decreasing \( q_i \) (Zipf's Law)

\[
q_i = K/(\delta + i), \text{ with a constant } \delta; \delta = 0, 10, 50;
\]

C. Constant \( q_i \): \( q_i = K \);

D. Uniform \( q_i \): \( q_i = K \cdot U_i \), where \( U_i \) is a uniformly distributed r.v. in (0, 1);

E. The Adams data in table 1.

For A - D:

\[
T_q = 0.05, 0.01, 0.001;
\]

\[
c = 10^5, 10^6, 10^7.
\]

A & B were chosen because the Adams data falls between them, \( i.e., \) the \( q_i \) in table 1 decrease faster than Zipf's law but slower than the exponential rate. Constant rate is included because many SR are based on this assumption, and the uniform rate is used following Ross [20] in his simulation.

Many simulations were run. Since each method requires a minimum amount of data to apply its SR, \( N_0 \) was chosen as 300, 500, 1000; \( N \) was chosen as 100. The SR compared were Ross [20], Rasmussen & Starr [19], Nayak [18], Goudie [10], Dalal & Mallows [7], and the Musa basic-execution and LP models. The detailed formulation of these SR and several other SR were laid out in [25]. The ability of each SR to estimate the remaining \( R \) is also examined. Since we know the true remaining \( R \) in the simulation, the,

\[
e = |\hat{R} - R|,
\]

(13)
can easily be computed.

1000 simulations were done in each of the possible parameter combinations. Tables 2 & 3 give typical outputs. Only five of those SR are included.

### TABLE 2
Average Testing & Penalty Cost For Six SR

\([N=100, m=100, T_q=0.05, c=10^6, N_0=1000]\)

The body of the table gives: "estimated mean cost — including total-test and penalty" and "standard-error of the estimated mean"; the latter is in ( ) with units of \( 10^2 \)

<table>
<thead>
<tr>
<th>Method</th>
<th>( A, \alpha = 0.7 )</th>
<th>( B, \delta = 0 )</th>
<th>( C )</th>
<th>( D )</th>
<th>( E )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>29.3 (0.24)</td>
<td>104.2 (0.26)</td>
<td>81.7 (0.32)</td>
<td>84.5 (0.30)</td>
<td>118.3 (0.23)</td>
</tr>
<tr>
<td>Recapture (5)</td>
<td>36.0 (0.36)</td>
<td>111.5 (0.59)</td>
<td>84.6 (0.31)</td>
<td>88.5 (0.36)</td>
<td>124.5 (0.42)</td>
</tr>
<tr>
<td>Usual (8)</td>
<td>38.1 (0.29)</td>
<td>108.5 (0.48)</td>
<td>85.2 (0.36)</td>
<td>87.9 (0.36)</td>
<td>123.4 (0.24)</td>
</tr>
<tr>
<td>Dalal/Mallows</td>
<td>39.7 (0.50)</td>
<td>116.0 (0.28)</td>
<td>85.4 (0.30)</td>
<td>90.5 (0.28)</td>
<td>147.8 (0.21)</td>
</tr>
<tr>
<td>Musa Basic</td>
<td>36.1 (0.39)</td>
<td>107.3 (0.24)</td>
<td>85.0 (0.24)</td>
<td>88.2 (0.32)</td>
<td>130.6 (0.25)</td>
</tr>
<tr>
<td>Musa LP</td>
<td>34.9 (0.28)</td>
<td>120.1 (0.97)</td>
<td>148.4 (0.32)</td>
<td>128.5 (2.24)</td>
<td>166.0 (0.51)</td>
</tr>
</tbody>
</table>

All the other SRs, when adapted to our cost structure, were no longer competitive. Since our cost structure was not studied in the Musa nor in the Dalal/Mallows modeling, we put our version of their SR in appendix C. The results here do not
The body of the table gives: \( c = \left| \bar{A} - \bar{R} \right| \) and \( \text{standard error of } c \); the latter is in (0, both with units of \( 10^{-6} \)).

<table>
<thead>
<tr>
<th>Method</th>
<th>A, q=0.7</th>
<th>B, ( \delta=0 )</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>0.00 (0.000)</td>
<td>0.00 (0.000)</td>
<td>0.00 (0.000)</td>
<td>0.00 (0.000)</td>
<td>0.00 (0.000)</td>
</tr>
<tr>
<td>Recapture (5)</td>
<td>1.39 (0.039)</td>
<td>1.23 (0.035)</td>
<td>0.90 (0.021)</td>
<td>1.03 (0.025)</td>
<td>1.54 (0.041)</td>
</tr>
<tr>
<td>Usual (8)</td>
<td>2.07 (0.075)</td>
<td>1.18 (0.064)</td>
<td>2.70 (0.037)</td>
<td>1.74 (0.032)</td>
<td>1.72 (0.033)</td>
</tr>
<tr>
<td>Dalal/Mallows</td>
<td>1.55 (0.052)</td>
<td>1.33 (0.044)</td>
<td>0.83 (0.022)</td>
<td>1.07 (0.030)</td>
<td>2.14 (0.049)</td>
</tr>
<tr>
<td>Musa Basic</td>
<td>4.78 (0.042)</td>
<td>2.03 (0.046)</td>
<td>1.11 (0.029)</td>
<td>1.51 (0.040)</td>
<td>2.91 (0.049)</td>
</tr>
<tr>
<td>Musa LP</td>
<td>0.92 (0.025)</td>
<td>2.70 (0.116)</td>
<td>5.08 (0.135)</td>
<td>3.95 (0.141)</td>
<td>2.03 (0.077)</td>
</tr>
</tbody>
</table>

necessarily imply that the other methods are less effective in the circumstances they were designed for. From the simulation results, we observed the following.

1. As we might anticipate, there is no definite winner in all the \( q_i \) configurations. The Musa LP model does extremely well in rapidly decreasing \( q_i \), but does poorly in others. For Zipf's law, constant \( q_i \), and uniform \( q_i \), our two adaptive SR and the SR by Dalal/Mallows and Musa Basic models have similar performance. Taking the standard error of estimation into consideration, their differences are not significant. But if we consider the Adams data to be typical, then our adaptive SR are better in average cost reduction and in estimation of \( R \).

2. As mentioned in section 2, it is not fair to compare the optimal SR (3) with other SR, because there is a tremendous prior information gap between them. However, the simulation shows that for cases B, C, D, the existing SR are nearly optimal. Further research on improving the testing strategies under these three models does not seem worthwhile.

3. For the same \( T_p \), it is easier to debug "one bug with large \( q_i \)" than "many bugs each with small \( q_i \)". Thus \( T_p \) has little effect on the total cost. Testing is more difficult with low-

4. The range of \( c \) is important in determining the closeness between the adaptive and the optimal SR. The desirable ranges (12) for \( c \), for each case A - E, are:
   - A: none
   - B: \( 3 \cdot 10^{-5} \) to \( 10^{7} \)
   - C: \( 6 \cdot 10^{-6} \) to \( 10^{6} \)
   - D: \( 5 \cdot 10^{-4} \) to \( 6 \cdot 10^{-5} \)
   - E: \( 2 \cdot 10^{-6} \) to \( 1.4 \cdot 10^{-10} \).

Tables 2 & 3 clearly show the relationship between the cost discrepancy from the optimal SR and the closeness of \( c \) to the desirable range. The lengthy desirable range for the Adams data seems to indicate that the adaptive SR will work well in commercial software.

5. We checked the robustness of the existing SR when the pure death assumption of bugs in debugging is slightly violated (see assumption 5). From a mathematical viewpoint, there is no difference between a hidden bug B behind a bug A, and a bug B generated during the correction of bug A. In other words, they share the same consequence; B appears only after A is removed. We simulated this situation by assuming a \( \beta \) birth rate when a bug is removed. The \( q_i \) of the newly generated bug is randomly picked from the original bug pool. We tried \( \beta =0.05 \) and \( 0.10 \); the results were very similar to those in tables 2 & 3.

APPENDIX A

Derivation of the Optimal Stopping Rules (2)

We found no elementary proof for (2). However, if we use the 1-step-ahead SR, then (2) is easy to derive. The 1-step-ahead SR is: "If the next step is not profitable, then stop, regardless of what would happen many steps ahead." To see this, suppose there are still \( k \) bugs with \( \{q_i\}_{i=1}^k \) left in the software after round \( n \). Then if we decide to test one more round, the cost is \( N \cdot c_1 \). The mean reduction of penalty is:

\[
\begin{align*}
\varphi_2 \cdot M & \cdot E \left\{ \sum_{i=1}^{k} q_i \cdot \Pr \{ \text{bug } i \text{ is discovered in the nest } N \text{ tests} \} \right\}, \text{ or} \\
\varphi_2 \cdot M & \cdot \sum_{i=1}^{k} q_i \cdot (1 - p^n_i).
\end{align*}
\]

Thus, the 1-step-ahead SR says that we stop iff \( \varphi_2 \cdot M \cdot \sum_{i=1}^{k} q_i \cdot (1 - p^n_i) \leq N \cdot c_1 \) which is the same as (2).

For a rigorous proof, we follow [6: theorem 3.3].

Notation

\[
\begin{align*}
U(n) & \text{ } \sum_{i} q_i \cdot \mathbb{I}(X_i(n) = 0) \\
\tau & \text{ any SR: stop at } \tau \\
\phi & \text{ optimal SR for } E\{C(\phi)\} \leq E\{C(\tau)\}: \text{ stop at } \phi \\
\mathcal{F}_n & \text{ sigma field generated by all the previous samples up to } n.
\end{align*}
\]

If the program is released after test period \( n \), the cost is:

\[
C(n) = \varphi_2 \cdot M \cdot U(n) + c_1 \cdot n \cdot N.
\]

Our purpose is to find the \( \phi \) for any \( \tau \). A SR is a decision that depends only on the sampling information from the past, not the future. To put it in the usual notation, \( \tau \) is a r.v. such that the event \( (\tau = n) \in \mathcal{F}_n \). Without loss of generality, we let:

\[
\begin{align*}
c_1 = 1 & \text{ and } \varphi_2 \cdot M = c_2; \\
g(n) = -c \cdot U(n) - n \cdot N.
\end{align*}
\]

According to [6: theorem 3.3], we need to show:

i. The set \( A_n = \{ E[g(n+1) | \mathcal{F}_n] \leq g(n) \} \) is monotonically increasing in \( n \), and
in order for the SR $\phi$, defined as: "stop at the first $n$ such that $g(n) \geq E\{g(n+1)|S_n\}$(same as SR (2))", to be optimal among all the SR $\tau$ satisfying $\lim_n\{\inf_{\tau>n}g^+(n)\ dP\} = 0$. To show #i, note that $E\{g(n+1)|S_n\} \leq g(n)$ implies:

$$-c \cdot \sum_{i:X(n)=0} q_i \cdot Pr\{\delta(X_i(n+1)=0)|X_i(n)=0\}$$

$$+ (n+1) \cdot N \leq -c \cdot U(n) - n \cdot N$$

$\Rightarrow c \cdot \sum_{i} q_i \cdot (1-p_i^N) \cdot \delta(X_i(n)=0) \leq N$

$\Rightarrow c \cdot \sum_{i} (1-p_i^N) \cdot \delta(X_i(n+1)=0) \leq N$

$= A_n \subseteq A_{n+1}$. 

Thus #i is proven. Because $g^+(n)=0$ for all $n$, #ii is true. Since,

$U(n) \leq 1,$

$g^-(n) = c \cdot U(n) + n \cdot N < c + n \cdot N,$

$\int_{r > n} g^-(n) \ dP \leq (c+n \cdot N) \cdot Pr\{r > n\} = 0,$

as $n \rightarrow \infty$, for $E\{\tau\} < \infty$.

Thus, we have shown that $\phi$ is the optimal SR for all SR with finite $s$-expectations. Suppose $E\{\tau\} = \infty$, then $E\{g(\tau)\} \leq -E\{g(\tau)\} = -\infty$. Because the SR that stops with $\phi=1$ has an $s$-expectation greater than $-(c+N)$, this $\tau$ cannot be optimal. We have proven (2) is optimal among all $\tau$.

APPENDIX B

Derivation of the Variance for (10)

Notation

$\Psi_i = q_i^2 \cdot \delta(X_i(n)=0) - B_n\binom{n-2}{2}$. 

Since we have shown that $E\{\Delta(n)\} = 0$,

$\text{Var}\{\Delta(n)\} = E\left\{ \sum_{i} \Psi_i \right\}^2$

$= \sum_{i} E\{\Psi_i\}^2 + \sum_{i,j} E\{\Psi_i,\Psi_j\}$

By $E\{\delta(X_i(n)=2)\} = \binom{n-2}{2} q_i^2 p_i^n (n-2)$, we can show that the a) cross product terms $\approx 0$, and b) squared terms become (10).
is shown in Singpurwalla [24] where only a 2-step testing scheme is considered.

C.2 Dalal/Mallows [7]

Notation

\[ S(t) = \sum_{i=1}^{m_i} T_i \]

\[ \mu = \text{solution of (C-3)} \]

\[ \xi = \text{cost of an unremoved bug.} \]

\[ \mu (\exp(\mu t) - 1)/\left(\exp(\mu t) - 1 - \mu t\right) = m_i/S(t). \quad (C-3) \]

Their SR is to stop when,

\[ m_i \leq (\exp(\mu t) - 1) \cdot c_i/\xi; \mu \]

The \( \xi \) has no direct correspondence in our cost structure. We adjust the \( c_i/\xi \) ratio so that is has the best performance. The \( R \) is estimated as:

\[ \mu \cdot m_i/(\exp(\mu t) - 1). \]

REFERENCES


AUTHORS

Dr. Mark C.K. Yang: Dept. of Statistics; Univ. of Florida; Gainesville, Florida 32611 USA.

Internet (e-mail): yang@stat.ufl.edu

Mark C.K. Yang received the BS (1964) in Electrical Engineering from National Taiwan University, MS (1967) in Mathematics and PhD (1970) in Statistics from the University of Wisconsin, Madison. He has worked with the US Department of Energy, US NASA, Bell Labs, and US Naval Research Lab. He has published more than 50 research articles in refereed statistical & engineering journals and is a professor at the University of Florida. His research interests are sampling, optimal stopping rules, and reliability.

Dr. Anne Chao; Inst. of Statistics; National Tsing-Hua University; Hsin Chu, TAIWAN.

Internet (e-mail): nhut060@gwahcu10.edu.tw

Anne Chao received the BS (1973) in Mathematics from National Tsing-Hua University and PhD (1977) in Statistics from the University of Wisconsin, Madison. She has published research papers in Annals of Statistics, Biometrics, Biometrika, J. Amer. Statistical Assoc., IEEE Trans., and other journals. She is now a professor at the National Tsing Hua University. Her research interests are wildlife sampling, coverage probability, and software reliability. Manuscript received 1994 March 16.