Robust Dynamic Selection of Tested Modules in Software Testing for Maximizing Delivered Reliability

Ping Cao\textsuperscript{1}, Zhao Dong\textsuperscript{2}, Ke Liu\textsuperscript{2}, Kai-Yuan Cai\textsuperscript{3,4}

\textsuperscript{1}School of Management, University of Science and Technology of China, Hefei, 230026, China

\textsuperscript{2}MADIS and National Center for Mathematics and Interdisciplinary Sciences, Academy of Mathematics and Systems Sciences, CAS, Beijing 100190, China

\textsuperscript{3}Department of Automatic Control, Beijing University of Aeronautics and Astronautics, Beijing 100191, China

\textsuperscript{4}State Key Laboratory of Computer Science, Institute of Software, CAS, Beijing 100190, China

Abstract

Software testing is aimed to improve the delivered reliability of the users. Delivered reliability is the reliability of using the software after it is delivered to the users. Usually the software consists of many modules. Thus, the delivered reliability is dependent on the user’s operational profile which specifies how the users will use these modules as well as the defect number remaining in each module. Therefore, a good testing policy should take the users’ operational profile into account and dynamically select tested modules according to the current state of the software during the testing process. This paper discusses how to dynamically select tested modules in order to maximize delivered reliability by formulating the selection problem as a dynamic programming problem. As the testing process is performed only once, risk must be considered during the testing process, which is described by the tester’s utility function in this paper. Besides, since usually the tester has no accurate estimate of the users’ operational profile, by employing robust optimization technique, we analysis the selection problem in the worst case, given the operational profile’s uncertainty set. By numerical examples, we show the necessity of maximizing delivered reliability directly and using robust optimization technique when the tester has no clear idea of the operational profile. Moreover, it is shown that the risk averse behavior of the tester has a major influence on the delivered reliability.

Keywords: dynamic programming, risk, delivered reliability, unknown operational profile, robust optimization.
1 Introduction

Software testing is the most popular method for improving the software reliability [16]. It is a major paradigm for software quality assurance and is extensively carried out in nearly every software development project [17]. During software testing process, the defects remaining in the software will be detected and removed subsequently, resulting in the improvement of software reliability.

A typical software testing process can be depicted in the following way [8]: At the beginning the test cases are generated and subsequently divided into several classes. For each testing, first select a class of test cases, then a test case will be chosen randomly from this class. Executing this test case will result in either detecting a defect or detecting no defect. In the former case, the detected defect will be removed subsequently. Then a test is completed and the tester moves to the next test. From the testing process we know that we can control the testing process by appropriately selecting the subsets. Based on this special testing pattern, several software testing methods are proposed, including Markov usage model based testing and random testing [20]. Markov usage model based testing assumes that the subset will be selected according to a Markov chain and random testing assumes that the subset will be selected randomly. However, these methods all assume that the selection process will evolve in a specified way, in which it has no relationship with the testing result. Thus, these methods may not be able to detect and remove the defects remaining in the software in an optimal way, which may affect the effectiveness of the testing process. Therefore, a good selection policy should take the current state of the software into account.

A lot of literature in software testing studies how to remove the defects as many as possible during the testing process [3, 4, 5, 6, 7]. However, the actual goal of software testing is not removing the defects but improving delivered reliability. Minimizing the defect number is not exactly equivalent to maximizing delivered reliability. Thus, a testing policy which is optimal in minimizing the defect number may performs badly in maximizing delivered reliability. It is quite necessary to directly set maximizing delivered reliability as the objective. Besides, although there is a qualitative relationship between the remaining defects and delivered reliability, the quantitative relationship is hard to depict. Many papers including [8, 9, 10] discuss how to assess the delivered reliability given a testing policy. However, they assume that the testing policy is a given Markov usage model based testing and ignore the fact that the testing policy may vary with the testing process. As far as the authors know, there is no paper studying how to achieve a high delivered reliability by dynamically selecting the test case during the testing process. But it is a topic worth discussing.

Removing the defects is not the ultimate goal of software testing. We test the software to make it perform well and meet end-users’ needs. Therefore, delivered reliability is what we really care. It is quite necessary to make the testing’s objective be maximizing the delivered reliability rather than minimizing the number of remaining defects. In fact, we shows that minimizing the number of remaining defects can bring a significant small delivered reliability.

Moreover, after the software is delivered, it will be used by various users. It is usually assumed that
the users will use the software in a given operational profile. This profile describes how the users will use
the software. A lot of literature assumes that the operational profile is known before testing[15, 21]. This
assumption seems unreasonable, but it can be explained by that the tester may have the operational profile
of a software with similar function. This profile can be used as the approximated operational profile of the
software under test. It is highly probable that the estimated operational profile is quite different with the real
one. A wrongly supposed operational profile will lead to a selection policy quite deviated from the optimal
one, which will result in a lower delivered reliability. In this paper, we employ robust optimization technique
to find a selection policy in the worst case by assuming that the operational profile lies in an uncertainty set.
Our numerical example shows that the resulting selection policy based on robust optimization has stable
performance, no matter what the true value of the operational profile is.

A significant difference between the software testing process and other stochastic decision processes is
that it is performed only once, i.e., the process is unrepeatable. Thus, the objective which is to maximize
expected reliability at the end of the testing process may not be adequate in this situation as a single poor
realization can have a great impact on the reliability. Thus, the tester may want to control the risk during
the testing process and he may be willing to use selection policies that sacrifice some expected reliability in
return for a less risk, or volatility of the reliability. In other words, the tester is prone to be risk averse
rather than risk neutral. In this paper, we capture the tester’s risk averse behavior by using an increasing
and concave utility function.

There is a lot of literature discussing how to dynamically select the testing subsets. Cai has done a lot
of work on the controlled Markov chains approach to software testing [4, 5, 6, 7]. Those papers study how
to test software such that all the remaining defects are detected and removed at the least expected cost or
how to test software such that defects are detected and removed as many as possible given a cost bound
or the maximal number of tests. Therefore, the objective in those papers is to minimize the number of the
remaining defects rather than maximize the delivered operational reliability. Moreover, those papers did not
incorporate the risk into the testing process. However, these issues cannot be ignored in the practical testing
process.

This paper studies how to dynamically select the tested modules during the testing process to maximize
delivered reliability. Our contribution is three fold. First, the objective becomes maximizing delivered
reliability. Second, we consider the worst case when the tester faces uncertainty of the operational profile.
Usually the tester cannot give the clear value of the user’s operational profile, as different software may have
different operational profile and the tester may not be able to predict accurately the operational profile before
the software is released to the market. However, the tester may have a rough idea by the operational profile of
the software with similar function. Therefore, we can assume that the value of the user’s operational profile
lays in a given uncertainty set. Under this assumption we can discuss this problem in robust optimization
framework. Finally, we consider the risk factor in the selection process. In most of the literature the risk
factor is ignored and the average performance of the selection policy is the only criterion. This may contradict
the common sense that the decision maker is prone to be risk averse as for a specified software the testing
process will be performed only once. Therefore, the expected performance may be quite different with the
realized performance. In this case, the risk must be taken into account.

It is worth mentioning that there are also several papers discussing the risk during the testing, such
as [1, 3]. However, the risk mentioned in this paper is quite different from that in [1, 3]. In [1, 3], risk
means something that might happen. Thus, it can be measured by the probability of a fault. The risk in
those papers is quite similar with the reliability except that it also takes the cost of a fault into account. In
this paper, risk is the deviation of the realized reliability from the expected reliability, which can be called
measure risk. Therefore, they are two different concepts of risk.

This paper is structured as follows. In Section 2, we give the software testing model description and
formulate it as a dynamic programming problem. In Section 3, we analysis the optimal function and the
optimal selection policy. In Section 4, we conduct several numerical examples to examine the impact of testing
objective, the benefit of robust optimization, and the effects of tester’s risk aversion behavior. Section 5
concludes this paper.

2 Model Description and Mathematical Formulation

We take the following assumptions for the software testing process and the software defect removal mecha-
nism.

1. The software under test comprises \( m \) modules \( C_i, i = 1, 2, ..., m \).
2. At the beginning of the testing process, there are \( N_i \) defects in \( C_i, i = 1, 2, ..., m \).
3. We test the software for \( T \) periods; then the software will be released to the market and be used by
the users.
4. At the beginning of each testing period, the tester decides which module should be tested during this
period. The defects in \( C_i \) are independent with each other and each defect in \( C_i \) will trigger a failure
with probability \( \theta_i \).
5. The debugging is perfect and instantaneous, i.e., a defect will be removed immediately upon detected
and no new defect will be introduced into the software.

We have a few remarks about the above assumptions.

1. The initial defect number of module \( C_i \) is in fact unknown. However, since the topic of this paper is
focused on evaluating the impact of operational profile on the selection process, we assume that \( N_i \) is
known for now.
2. For all software projects, time and resources are limited. Thus the number of testing periods $T$ can represent the limited time or resources we have during the testing process.

3. As mentioned in [11], the time to detect a defect seems to be nearly exponentially distributed and the scale parameter is stable across modules. Therefore, it is reasonable to assume that the defects in $C_i$ are independent with each other and each defect in $C_i$ will trigger a failure with a fixed probability. The parameter $\theta_i$ is in fact unknown. A prior on $\theta_i$ may be used and updated according to the testing process, but the mathematical arguments would become quite messy. Since the subject of this paper is not focused on estimating these parameters, we assume that $\theta_i$ is already known.

4. The perfect and instantaneous removal assumption is widely adopted in literature of software testing. It is frequently assumed that after release the software will be used according to an operational profile which can be described by a probability vector $\mathbf{p} = (p_1, p_2, ..., p_m)^T$, where the notation $T$ is used to transpose the matrix, $p_i \geq 0$, $i = 1, 2, ..., m$ and $\sum_{i=1}^{m} p_i = 1$. Under the operational profile $\mathbf{p}$, the users will use module $C_i$ with probability $p_i$, $i = 1, 2, ..., m$. Thus, if there are $x_i$ defects in module $C_i$ after release, $i = 1, 2, ..., m$ (denote $\mathbf{x} = (x_1, x_2, ..., x_m)^T$), then the probability of using the software within a time period without triggering failures for the users is

$$R(\mathbf{x}, \mathbf{p}) = \sum_{i=1}^{m} p_i (1 - \theta_i)^{x_i},$$

(1)

which can be used as a software reliability measure. Since the reliability is computed from the perspective of the users, we call $R(\mathbf{x}, \mathbf{p})$ the delivered reliability.

There are a lot of software reliability measure in literature [19]. In this paper we use the above reliability measure. However, the argument can be obtained similarly for other reliability measures.

For a new software that has not yet been released to the market and delivered to the users, it is hard, if not impossible, to predict the exact value of the operational profile $\mathbf{p}$. However, by the historical record of softwares having similar functions with the software under test, it is reasonable to assume that $\mathbf{p}$ lies in an uncertainty set $P$. Since $\mathbf{p}$ must be a probability vector, we know that $\mathbf{p}$ belongs to the probability vector space

$$\mathcal{PV} = \{\mathbf{x} = (x_1, x_2, ..., x_m)^T \in \mathbb{R}^m : x_i \geq 0, i = 1, 2, ..., m, \sum_{i=1}^{m} x_i = 1\}.$$

There are several forms of the uncertainty set $P$. We assume that $P = \{\mathbf{p} \in \mathcal{PV} : \mathbf{p}_0 \in \mathcal{PV}, \mathbf{p} - \mathbf{p}_0 = \sum_{\ell=1}^{L} \zeta(\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_m)^T, \zeta \in \mathcal{U}\}$. Now we give two types which are tractable. In the case of interval uncertainty, we can normalize the situation by assuming that $\mathcal{U} = \{\zeta \in \mathbb{R}^L : \|\zeta\|_{\infty} \leq 1\}$. In the case of ellipsoidal uncertainty, we can assume that $\mathcal{U} = \{\zeta \in \mathbb{R}^L : \|\zeta\|_2 \leq 1\}$.

The tester’s objective is maximizing the delivered reliability by dynamically selecting the testing modules at the beginning of each testing period. Note that the delivered reliability is a random variable, whose
randomness will evolve during the testing process. A natural way is maximizing the expected delivered reliability instead. However, since the testing process is only performed once, the tester cares more about the realization of the reliability rather than its expectation. It may happen that the reliability induced by a selection policy is high in average while it can probably get a quite low value in its realization. This is what the tester wants to avoid. Therefore, the tester is prone to be risk averse and he wants to control the reliability from below. Thus, we turn to maximize the expected utility of the delivered reliability. A typical risk-averse tester has an increasing concave utility function \( U(x) \). Moreover, since we do not know the exact value of the operational profile, we maximize the expected utility of the delivered reliability in the worst case.

Let the number of defects in module \( C_i \) at the end of period \( t \) is \( N_i(t) \), and let \( \mathbf{N}(t) = (N_1(t), N_2(t), \ldots, N_m(t))^T \). It makes no confusion if we let \( N_i(0) = N_i \), and thus \( \mathbf{N}(0) = (N_1, N_2, \ldots, N_m)^T \). By the above argument, the dynamic selection problem can be formulated as

\[
\max_{u \in U} \min_{p \in P} \mathbb{E}[U(R(\mathbf{N}(T), p)) | \mathbf{N}(0) = \mathbf{N}],
\]

where \( U \) consists of all the non-anticipated selection policies, and \( U(x) \) is a utility function which is increasing and concave in \( x \).

We call the testing process is at state \((x, t)\) if there are \( x_i \) defects in \( C_i \), \( i = 1, 2, \ldots, m \) at the beginning of time period \( t \), where \( x = (x_1, x_2, \ldots, x_m) \).

Denote

\[
J_t(x) = \max_{u \in U} \min_{p \in P} \mathbb{E}[U(R(\mathbf{N}(T), p)) | \mathbf{N}(t) = x].
\]

Obviously,

\[
J_T(x) = \min_{p \in P} U(R(x, p)).
\]  

Note that if the testing process is at state \((x, t)\) and the tester chooses module \( C_i \), then the defect number in module \( C_j \) will remain the same, \( j \neq i \). For each defect in module \( C_i \), with probability \( \theta_i \) it will be detected and removed. Thus, the defect number in module \( C_i \) is a random number, which can be written as \( \sum_{j=1}^{x_i} W_{i,j} \), where \( W_{i,j} \) is independent with each other and follows Bernoulli distribution with parameter \( 1 - \theta_i \).

Therefore, by the principle of optimality, \( J_t(x) \) satisfies the dynamic programming equation

\[
J_t(x) = \max_{1 \leq i \leq m} \mathbb{E}[J_{t+1} \left( x_1, x_2, \ldots, x_{i-1}, \sum_{j=1}^{x_i} W_{i,j}, x_{i+1}, \ldots, x_m \right)]
\]

\[
= \max_{1 \leq i \leq m} \sum_{k=0}^{x_i} \left( \frac{x_i}{k} \right) (1 - \theta_i)^k \theta_i^{x_i-k} J_{t+1} \left( x_1, x_2, \ldots, x_{i-1}, k, x_{i+1}, \ldots, x_m \right),
\]

for \( t = 0, 1, \ldots, T - 1 \).  

By the boundary condition (3) and the optimal equation (4), we can solve \( J_t(x) \) and the optimal selection policy recursively from \( t = T \) to \( t = 0 \).
3 Analysis

In this section, we analysis properties of the optimal function $J_t(x)$ and the optimal selection policy. The properties can help characterize the optimal function and make the tester more clear about the relationship of the value function $J_t(x)$ with the remaining defect numbers $x$ and current testing period $t$.

3.1 Analysis of the terminal function $J_T(x)$

There are several forms of the uncertainty set $P$. We assume that $P = \{p \in \mathcal{P}V : p_0 \in \mathcal{P}V, p - p_0 = \sum_{\ell=1}^{L} \zeta_{\ell}(y_1, y_2, ..., y_m)^T, \zeta \in Z\}$.

Now we give two types of uncertainty sets which are quite common and tractable.

In the case of interval uncertainty, we can normalize the situation by assuming that $Z = \{\zeta \in \mathbb{R}^L : \|\zeta\|_\infty \leq 1\}$. A typical interval uncertainty set is

$$P = \{p \in \mathcal{P}V : p_1 \leq p \leq p_2\}.$$ 

In the case of ellipsoidal uncertainty, we can assume that $Z = \{\zeta \in \mathbb{R}^L : \|\zeta\|_2 \leq 1\}$. A typical ellipsoidal uncertainty set is

$$P = \{p \in \mathcal{P}V : p_0 \in \mathcal{P}V, (p - p_0)^T\Sigma^{-1}(p - p_0) \leq \varepsilon^2\},$$

where $\varepsilon$ denotes the confidential level of the operational profile $p_0$. Small $\varepsilon$ implies that the tester is quite sure about the true value of the operational profile.

From (3) and the fact that $U(x)$ is increasing in $x$, we know that

$$\arg \min_{p \in P} U(R(x, p)) = \arg \min_{p \in P} R(x, p) = \arg \min_{p \in P} \sum_{i=1}^{m} p_i(1 - \theta_i)x_i.$$ 

Therefore, $J_T(x)$ can be obtained by solving

$$\min_{p \in P} \sum_{i=1}^{m} p_i(1 - \theta_i)x_i,$$

subject to $p \in P$

Note that the objective function of the above problem is a linear function in $p$. Thus, if $P$ is an interval uncertainty set, then the above problem is a linear programming problem; if $P$ is an ellipsoidal uncertainty set, then it is a quadratically constrained linear programming problem. Each problem can be solved quite efficiently by using commercial solvers such as CPLEX or MOSEK.

In the case of $m = 2$, (3) can be solved analytically. Note that if $p_1$ is known, the value of $p_2$ is also determined by $p_1 + p_2 = 1$. Thus, the uncertain set $P$ can be written as $P = \{p = (p_1, p_2)^T : \hat{p}_1 - \delta \leq p_1 \leq \hat{p}_1 + \delta, \hat{p}_1 + p_2 = 1\}$. Therefore,

$$J_T(x) = \min_{\hat{p}_1 - \delta \leq p_1 \leq \hat{p}_1 + \delta} U(p_1(1 - \theta_1)x_1 + p_2(1 - \theta_2)x_2)$$
Thus, from (4) we know that
\[ J \]
From Proposition 3.1 we know that
\[ J \]
If
\[ \text{Lemma 3.1.} \]
\[ x \in T \]
number detected and removed, which means that the expected reliability will be greater. Thus, a large testing period
\[ \text{Corollary 3.1.} \]
\[ J \]
\[ P' \]
3.2 Properties of the optimal function \( J_t(x) \)

**Proposition 3.1.** \( J_t(x) \) is decreasing in \( x \).

**Proof.** We prove this proposition by induction on \( t \).

Obviously, from (1) and (3) we know that \( J_T(x) \) is decreasing in \( x \).

Suppose it holds for \( t = k + 1 \). Next we show that it holds for \( t = k \).

By the induction hypothesis, we know that \( \mathbb{E} J_{k+1}(x_1, x_2, ..., x_{i-1}, \sum_{j=1}^{x_i} W_{i,j}, x_{i+1}, ..., x_m) \) is decreasing in \( x_i, i = 1, 2, ..., m \). Thus, from (1) and the maximization of decreasing functions is also a decreasing function we know that \( J_k(x) \) is also decreasing in \( x \).

Therefore, \( J_t(x) \) is decreasing in \( x \).

This result is consistent with our intuition: With more defects remaining, the less the expected reliability will be. Thus, it is beneficial to remove defects as many as possible.

**Corollary 3.1.** \( J_t(x) \) is decreasing in \( t \).

**Proof.** From Proposition 3.1 we know that \( \mathbb{E} J_{t+1}(x_1, x_2, ..., x_{i-1}, \sum_{j=1}^{x_i} W_{i,j}, x_{i+1}, ..., x_m) \geq \mathbb{E} J_{t+1}(x) \).

Thus, from (1) we know that \( J_t(x, y) \geq J_{t+1}(x, y) \), which implies that \( J_t(x) \) is decreasing in \( t \).

This result is also rather intuitive: More testing times left implies with more chance the defects will be detected and removed, which means that the expected reliability will be greater. Thus, a large testing period number \( T \) will result in a greater reliability.

**Lemma 3.1.** If \( f_i(x) \) is convex in \( x, i = 1, 2, ..., m \), then \( f(x) = \max_{1 \leq i \leq m} f_i(x) \) is also convex in \( x \).

**Proof.** For any \( x_1 < x_2 \) and \( 0 < \alpha < 1 \), there exists a number \( i_0 \) such that \( f(\alpha x_1 + (1 - \alpha)x_2) = f_{i_0}(\alpha x_1 + (1 - \alpha)x_2) \). Thus, we have
\[
\begin{align*}
& f(\alpha x_1 + (1 - \alpha)x_2) = f_{i_0}(\alpha x_1 + (1 - \alpha)x_2) \\
& \leq \alpha f_{i_0}(x_1) + (1 - \alpha)f_{i_0}(x_2) \leq \alpha f(x_1) + (1 - \alpha)f(x_2),
\end{align*}
\]
which implies that \( f(x) \) is convex in \( x \).

Note that \( x \) can be a vector in the above lemma. By using this lemma, we have the following result.
Proposition 3.2. In the case that \( U(x) = x \) and \( P = \{ p \} \), \( J_t(x) \) is convex in \( x \).

Proof. We prove this proposition by induction on \( t \).

Obviously, \( J_T(x) = R(x, p) = \sum_{i=1}^m p_i (1 - \theta_i)^{x_i} \) is convex in \( x \).

Suppose it holds for \( t = k + 1 \). We prove it also holds for \( t = k \).

By the induction hypothesis, we know that \( \mathbb{E} J_{k+1} \left( x_1, x_2, \ldots, x_{i-1}, \sum_{j=1}^{x_i} W_{i,j}, x_{i+1}, \ldots, x_m \right) \) is convex in \( x \), \( i = 1, 2, \ldots, m \). Thus, from (4) and Lemma 3.1 we know that \( J_k(x) \) is also convex in \( x \).

Therefore, \( J_t(x) \) is convex in \( x \).

\( U(x) = x \) means that the tester is risk-neutral and \( P = \{ p \} \) means that the tester is quite sure about the exact value of the users’ operational profile. In this case, \( J_t(x) \) is convex in \( x \), which implies that \( J_t(x + e_i) - J_t(x) \geq J_t(x) - J_t(x - e_i) \). Thus, the expected reliability will increase in margin with respect to \( x_i \). Note that \( J_t(x + e_i) - J_t(x) \leq 0 \). Thus, if a defect in module \( C_i \) is removed (the defect number in module \( C_i \) changes from \( x_i \) to \( x_i - 1 \)), the expected reliability can be greatly improved when the defect number in module \( C_i \) (i.e., \( x_i \)) is small. It seems that it is beneficial to testing modules with less defects as removing one defect can significantly improve the expected reliability. However, with less defects remaining in module \( C_i \), one or more defects will be detected and removed with less probability. Thus, it might not be optimal to select module with the minimum number of remaining defects. Similarly, we can argue that it might not be optimal to select module with the maximal number of remaining defects. We will discuss this issue detailly in the following subsection.

However, Proposition 3.2 does not generally hold if the tester is risk averse or he only has a rough estimate of the users’ operational profile. Next we give two simply examples to illustrate this.

Example 3.1. \( m = 1 \), \( U(x) = x - x^2 \).

Obviously, \( p = 1 \). We have that \( J_T(x) = (1 - \theta_1)^x - (1 - \theta_1)^{2x} \). Thus, \( \frac{\partial^2 J_T(x)}{\partial x^2} = \ln(1 - \theta_1)^2(1 - \theta_1)^x \cdot (1 - 2(1 - \theta_1)^x) \), which can not be guaranteed to be nonnegative for all \( x \geq 0 \) and \( 0 < \theta_1 < 1 \). For instance, when \( \theta_1 = 0.1 \) and \( x = 2 \), \( \frac{\partial^2 J_T(x)}{\partial x^2} < 0 \). Thus, \( J_T(x) \) is not convex in \( x \).

Example 3.2. \( m = 2 \), \( U(x) = x \), \( P = \{ (0.2, 0.8)^T, (0.8, 0.2)^T \} \), \( \theta_1 = 0.3 \), \( \theta_2 = 0.2 \).

We have that \( J_T(12, 19) = 0.0114 \), \( J_T(11, 19) = 0.0155 \) and \( J_T(13, 19) = 0.0106 \). Note that \( J_T(12, 19) > 1/2( J_T(11, 19) + J_T(13, 19)) \), which illustrates that \( J_T(x) \) is not convex in \( x \).

By the above two examples, we can see that the properties of the optimal function depend heavily on the structure of the uncertainty set and the utility function.

3.3 Analysis of the Optimal Selection Policy

It is quite intuitive that if it is optimal to select a test case from \( C_i \), it is also optimal when there are more defects in \( C_i \), i.e., the optimal selection policy should have a monotonic property. However, the following discussion shows that the monotonic property of the optimal selection policy fails to hold.
We consider a special case in which \( U(x) = x \) and \( P = \{p\} \). We have

\[
J_T(x) = \sum_{i=1}^{m} \rho_i (1 - \theta_i)^{x_i}.
\]

Therefore, it holds that

\[
EJ_T \left( x_1, x_2, ..., x_{i-1}, \sum_{j=1}^{x_i} W_{i,j}, x_{i+1}, ..., x_m \right)
= \sum_{k=0}^{x_i} \binom{x_i}{k} (1 - \theta_i)^k \theta_i^{x_i-k} J_T(x_1, x_2, ..., x_{i-1}, k, x_{i+1}, ..., x_m)
= \sum_{k=0}^{x_i} \binom{x_i}{k} (1 - \theta_i)^k \theta_i^{x_i-k} \left( \sum_{j=1, j \neq i}^{m} p_j (1 - \theta_j)^{x_j} + p_i (1 - \theta_i)^k \right)
= \sum_{j=1, j \neq i}^{m} p_j (1 - \theta_j)^{x_j} + p_i \theta_i^{x_i} \left( 1 - \frac{(1 - \theta_i)^k}{\theta_i} \right)
= \sum_{j=1, j \neq i}^{m} p_j (1 - \theta_j)^{x_j} + p_i (1 - \theta_i + \theta_i^2)^{x_i}.
\]

From (1) we know that

\[
J_{T-1}(x) = \max_{1 \leq i \leq m} \left\{ \sum_{j=1, j \neq i}^{m} p_j (1 - \theta_j)^{x_j} + p_i (1 - \theta_i + \theta_i^2)^{x_i} \right\}
= \max_{1 \leq i \leq m} p_i ((1 - \theta_i + \theta_i^2)^{x_i} - (1 - \theta_i)^{x_i}) + \sum_{j=1}^{m} p_j (1 - \theta_j)^{x_j}.
\]

Thus, at state \((x, T-1)\), it is optimal to select from module \( C_i \) with \( i^* = \text{arg max}_{1 \leq i \leq m} p_i ((1 - \theta_i + \theta_i^2)^{x_i} - (1 - \theta_i)^{x_i}) \). Note that \( p_i ((1 - \theta_i + \theta_i^2)^{x_i} - (1 - \theta_i)^{x_i}) \) is not increasing in \( x_i \). Thus, it is possible that

\[
p_i ((1 - \theta_i + \theta_i^2)^{x_i} - (1 - \theta_i)^{x_i}) > p_j ((1 - \theta_j + \theta_j^2)^{x_j} - (1 - \theta_j)^{x_j}) \quad \text{for all \( j \neq i \) while \( p_i (1 - \theta_i + \theta_i^2)_j^{x_{j+1}} - (1 - \theta_i)^{x_{j+1}} < p_j ((1 - \theta_j + \theta_j^2)^{x_j} - (1 - \theta_j)^{x_j}) \) for some \( j \neq i \). Therefore, if it is optimal to select a test case from \( C_i \), it might not be optimal to do so when there are more defects in \( C_i \). The following numerical example justifies our conjecture.

**Example 3.3.** \( m = 2, U(x) = x, P = \{(0.2, 0.8)^T\} \), \( \theta_1 = 0.2, \theta_2 = 0.1, T = 2 \).

In the above example, we find that at the beginning of the testing period, it is optimal to select a test case from \( C_2 \) when there are 25 defects remaining in \( C_1 \) and 15 defects remaining in \( C_2 \) while it is optimal to select a test case from \( C_1 \) when there are 25 defects remaining in \( C_1 \) and 15 defects remaining in \( C_2 \). This example tells us that the monotonic property of the optimal selection policy fails to hold.
4 Numerical Study

In this section, we conduct several numerical examples to examine the impact of testing objective, the benefit of robust optimization, and the effects of tester’s risk aversion behavior.

4.1 Impact of Testing Objective

A lot of literature such as [4, 5, 6] discusses how to dynamically select the modules to minimize the number of residual defects while our objective is maximizing delivered reliability. For most of the time these two objectives are not equivalent. Suppose that the operational profile of the users is \( p = (p_1, p_2, \ldots, p_m) \) and the number of defects in module \( C_i \) at the end of the testing process is \( x_i, i = 1, 2, \ldots, m \). The reliability of using the software is \( \sum_{i=1}^{m} p_i (1 - \theta_i)^{x_i} \) while the number of residual defects is \( \sum_{i=1}^{m} x_i \). Therefore, these two objectives will not be equivalent.

In the previous literature, the former objective is frequently used [4, 5], that is, the testing objective is minimizing the number of residual defects. However, the software testing is used to improve the delivered reliability, rather than to minimize the residual defect number. Although the residual defect number can be used to measure the reliability, there may be a substantial gap between minimizing the defect number and maximizing the delivered reliability. Some modules may have many defects, but the user seldom uses these modules and each defect in these modules can trigger a failure with a relatively small probability. In this situation, the defect number may be relatively large while the delivered reliability may be relatively low. Therefore, maximizing the delivered reliability is a more preferred objective from the user’s perspective. The following is a numerical example to illustrate the difference of these two objectives.

**Example 4.1.** Suppose \( m = 2 \), \( N_1 = 40 \), \( N_2 = 50 \), \( T = 40 \), \( \theta_1 = 0.015 \), \( \theta_2 = 0.02 \), \( P = \{(0.2, 0.8)^T\} \).

If the objective is maximizing expected delivered reliability (assuming that the utility function is \( U(x) = x \)), the resulting expected reliability is 0.5382. However, if we use minimizing expected defect number as its objective, the resulting expected delivered reliability will be 0.4722, which is significantly less than 0.5382. Thus, these two objectives can result in quite different value of reliability, which must be taken into account during the testing process.

4.2 Benefit of Robust Optimization

Obviously different softwares also have different operational profiles. The operational profile of a software with similar functionality and the market research may be helpful in obtaining a rough estimate of the operational profile. However, it is difficult, if not impossible to know the exact value of the operational profile before the software is released to the market. It can be easily foreseen that different operational profile might have different optimal selection policy. It may happen that a selection policy is optimal for one operational profile while it performs quite badly for another operational profile. In the case of unknown
Table 1: The gaps of the expected reliability as the true value of \( p \) varies

<table>
<thead>
<tr>
<th>( p )</th>
<th>0.48</th>
<th>0.50</th>
<th>0.52</th>
<th>0.54</th>
<th>0.56</th>
<th>0.58</th>
<th>0.60</th>
<th>0.62</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gap</td>
<td>0.0641%</td>
<td>0</td>
<td>0.11%</td>
<td>0.3642%</td>
<td>0.6738%</td>
<td>1.302%</td>
<td>2.1970%</td>
<td>3.3791%</td>
</tr>
</tbody>
</table>

operational profile, a selection policy that performs robustly in the operational profile might be preferable.

Next we give a numerical example to show whether the robust optimization is more favorable in the case that the exact value of \( p \) is unknown.

**Example 4.2.** Let \( m = 2, N_1 = 40, N_2 = 25, T = 40, \theta_1 = 0.025, \theta_2 = 0.04. \)

We assume that the tester’s objective is maximizing expected delivered reliability, i.e., he is risk neutral and \( U(x) = x \). Suppose that the tester cannot know the exact value of \( p_1 \) (which is 0.5), but he knows that \( 0.48 \leq p_1 \leq 0.62 \). If the tester knows the exact value of \( p_1 \), he can use (3) and (4) to obtain a selection policy with the expected reliability 0.4809. However, he does not know the exact value of \( p_1 \). If the tester uses robust optimization technique to find the testing policy, the resulting reliability is 0.477. The gap which is defined to be the relative error of the two reliabilities is 0.815%. If the tester mistakenly choose the value of \( p_1 \) to be \( p_w \), he will obtain a selection policy which is different with the optimal way and generates a lower expected reliability. Let \( p_w \) vary from 0.48 to 0.62, we obtain the gaps of the expected reliability as shown in Table 1.

From Table 1 we find that the gaps are quite sensitive to the value of \( p_1 \). If the tester mistakenly choose the value of \( p_1 \) to be 0.62, he will obtain a expected reliability which is significantly lower than the optimal expected reliability.

### 4.3 Impact of Risk Aversion Behavior

In this section, we consider a special case of utility function to examine the impact of the tester’s risk aversion behaviors on the reliability. Specifically, the utility function of the tester is assumed to be \( U(x) = 1 - \exp(-x/\gamma) \), where \( \gamma \) is the tester’s risk tolerance. Larger value of \( \gamma \) implies that the tester has more risk tolerance to the delivered reliability. It can be easily seen that the tester tends to be risk neutral as \( \gamma \) goes to infinity. Thus, \( \gamma = +\infty \) corresponds to the case that the tester is risk neutral.

Next we give a numerical example to show the impact of the tester’s risk aversion behaviors on the reliability.

**Example 4.3.** \( m = 2, N_1 = 30, N_2 = 20, P = \{(0.4,0.6)^T\}, \theta_1 = 0.1, \theta_2 = 0.2 \) and \( T = 15 \).

We obtain the optimal selection policies for \( \gamma = 0.001, 0.01, 0.1 \) and 0.1 by solving (1) and (3), respectively. Then we run 10000 simulations according to these selection policies to obtain the distribution of the delivered reliability. For instance, Figure 1 is a histogram recording the frequency of the delivered reliability for \( \gamma = 0.001 \). It shows a relatively uniform distribution of the delivered reliability. In contrast, Figure 3 and Figure 4 show a relatively scattered distribution of the delivered reliability.
Figure 1: $\gamma = 0.001$

Figure 2: $\gamma = 0.01$

Figure 3: $\gamma = 0.1$
Table 2: The average reliability and the variance

<table>
<thead>
<tr>
<th>$\gamma$</th>
<th>0.001</th>
<th>0.01</th>
<th>0.1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>0.3947</td>
<td>0.4346</td>
<td>0.5504</td>
<td>0.5512</td>
</tr>
<tr>
<td>Variance</td>
<td>0.0087</td>
<td>0.0057</td>
<td>0.0099</td>
<td>0.0099</td>
</tr>
</tbody>
</table>

Table 2 shows the average reliability and the variance for $\gamma = 0.001, 0.01, 0.1$ and 1, respectively. It shows that small value of $\gamma$ will result in a small average reliability but the resulting variance is not always small. However, Figure 1 and Figure 2 shows that the reliability is quite centred around the average reliability. Figure 3 and Figure 4 shows that the reliability is quite deviated from the average reliability (noting that the average reliability is around 0.55).

5 Conclusion

Most of the literature on the control problem of the software testing process studies how to detect the defects as many as possible. However, there is a gap between the defect number and the delivered reliability while the tester’s object of testing is improving the reliability rather than reducing the latent defect number. Therefore, it is straightforward to take the users’ operational profile directly into account during the testing process and choose the selection policy based on the users’ operational profile. Since the users’ operational profile is usually unknown at the beginning of the testing process, we analysis the selection problem in the worst case, given that the users’ operational profile lies in an uncertainty set. Moreover, considering that the expected reliability is not equal to the actual reliability, the uncertainty (variability) of the reliability should also be taken into account when we try to find a good selection policy. This kind of variability implies risk for the tester. We use utility function to value the risk.

There are at least fourth topics worthy of future research. First, in this paper the testing times $T$ is fixed. However, during the actual testing activities, the tester might decide to release the software earlier or later based on the testing situation. That is, the tester can also dynamically adjust the testing times.
according the testing process. Second, in this paper we present the testing times $T$ as the testing budget while there are other budgets such as testing expenses which are not considered yet. Third, it is assumed that the defect number in each module at the beginning of the testing process $N_i$ and the detect probability $\theta_i$ are known, which does not meet the reality. These parameters must be estimated online. In order to incorporate parameter estimation into the testing process, adaptive control might be used to model this problem and Bayesian dynamic programming might be a useful analytical tool. Finally, when the number of tested modules $m$ is large, it will take a lot of time in computing (4) directly. This problem is especially important for a complex software system with hundreds of modules. We might need approximate dynamic programming technique to tackle this problem. Heuristic selection policies might be more preferable.

References


