Optimal and Adaptive Testing with Cost Constraints†

Kai-Yuan Cai, Yong-Chao Li and Wei-Yi Ning
Department of Automatic Control
Beijing University of Aeronautics and Astronautics
Beijing 100083, China
kycai@buaa.edu.cn

ABSTRACT
This paper generalizes our previous work on optimal and adaptive testing to consider a more general scenario of software testing resource constraints. The assumption is that software testing must be stopped once the allowed testing resources are used up. The contributions of this paper are as follows. First, we show that software testing with fixed resource constraints can be handled in the framework of the controlled Markov chains (CMC) approach to software testing. Second, an algorithm is adopted to reduce the computational complexity of on-line decision making in the optimal testing strategy. Finally, the simulation results presented in this paper further confirm the effectiveness of the idea of adaptive testing in particular, and that of software cybernetics (which explores the interplay between software and control) in general.

Categories and Subject Descriptors
D.2.5 [Software Engineering]: Testing and Debugging

General Terms
Experimentation, Verification

Keywords
Software testing, optimal testing, adaptive testing, cost constraint, controlled Markov chain, software cybernetics

1 INTRODUCTION
The controlled Markov chains (CMC) approach to software testing proposed in our previous work [1-3] treats software testing as an optimal control problem, in which software under test serves as a controlled object modeled as a controlled Markov chain, and testing strategy serves as a controller. The underlying motivation is to formalize, quantify, and optimize the feedback mechanisms in software testing. The software under test and the corresponding testing strategy constitute a closed-loop feedback control system as shown in Figure 1.1. In this way the testing strategy is designed in accordance with some a priori given optimization criterion involving software testing/reliability objectives and costs. It has been shown that both the optimal software reliability growth problem and the optimal software reliability assessment problem can be formulated in the CMC framework [1-3].Reference [1] presents a simple CMC model of software testing. Reference [3] extends this model to deal with the scenario that software testing must be stopped once a given number of tests is executed. In this paper we generalize the work of reference [3]. We consider a more general scenario: software testing must be stopped once a given amount of software testing resources is used up, where each executed test consumes a certain amount of the resources. An algorithm is also applied to reduce the computational complexity of on-line decision making in the optimal testing strategy and the adaptive testing strategy.

Figure 1.1 Software testing as a control problem

Treating software testing as a problem of feedback and adaptive control defines a topic of software cybernetics, which is aimed at exploring the interplay between software and control. Although the idea of software cybernetics is not new [4-6], rigorous research work in this area is rather new and limited. A continuous-time state model for management of the software test process is reported in [7, 8]. Studies that show the theory of supervisory control of discrete event systems can be used to synthesize the required safety controller to guarantee the safe operation of the ConnectedSpace under consideration appear in [9, 10]. The synthesis of reactive software can also be formulated in the framework of the supervisory control of discrete event systems [11, 12]. Related work also includes anti-random testing [13] and the so-called “adaptive random testing” [14]. Reference [14] presents more related works in the emerging area of software cybernetics. In general, software cybernetics addresses issues and questions on (1) the formalization and quantification of feedback mechanisms in software processes and systems; (2) the adaptation of control theory principles to software processes and systems; (3) the application of the principles of software theories and engineering to control systems and processes; and (4) the integration of the theories of software engineering and control engineering.

The rest of the paper is organized as follows. Section 2 formulates the software testing problem examined in the paper and presents the theoretical results that define an optimal testing strategy. Section 3 explains an adaptive approach to the software testing problem. Section 4 reports simulation results to compare the performance of the optimal, adaptive, and random testing strategies. An algorithm is adopted in Section 5 to reduce the computational complexity in the optimal testing strategy and the adaptive testing strategy. Concluding remarks and future work are in Section 6.

† Supported by the National Natural Science Foundation of China (Grants Nos. 60474006, 60473067, 60233020)
2 THE SOFTWARE TESTING PROBLEM AND OPTIMAL TESTING STRATEGY

2.1 Methodology of Adaptive Software Testing

In software testing, test cases are selected and applied to the software under test one by one. Selection and execution of each test case consumes time and incurs cost. In practice, software testing resources (time, budget, etc.) are often limited and not allowed to exceed specified bounds. Software testing must terminate if the given testing resources are used up. In this paper we assume that the total cost of selection and execution of test cases cannot exceed a given value. Our goal is to reveal as many defects as possible within the cost constraint. To this end we confine ourselves to a discrete time domain for modeling of the software testing process. The ith test case is selected and applied to the software under test at time t = i. Let x0 be the given amount of testing resource in terms of cost, and Xt the available testing resource at time t for the rest of the testing process (including the cost of selecting and executing the test case at time t). This means that X0 = x0, X1 = x0 - w, where w denotes the cost of selecting and executing the first test case at time 0.

Let Yt = j if the software contains j defects at time t, Zt = 1 if the action taken at time t detects a defect, 0 if the action taken at time t does not detect a defect.

\[ \xi = (X_t, Y_t) \]

j = 0, 1, 2, ..., N; t = 0, 1, 2, ...

In this way \( \xi \) can be treated as the state of the software at time t. In state \( \xi \) the software contains Xt (excluding the possibly detected defect) by the test case applied at time t, and the rest of the software testing process (including selection and execution of the test case at time t) can use up at most Xt units of testing resource. We have the following assumptions:

1) The software contains N defects at the beginning (t = 0).
2) The testing process can use up at most x0 units of testing resource in total.
3) An action taken at one time detects at most one defect.
4) If a defect is detected, then it is removed immediately and no new defects are introduced; that is, Yi = j and Zi = 1 mean Yi+1 = j - 1.
5) If no defect is detected, then the number of remaining software defects remains unchanged; that is, Yi = j and Zi = 0 mean Yi+1 = j.
6) At every time there are always m admissible actions; the action set is \( A = \{1, 2, ..., m\} \).
7) Action \( A_i \) taken at time t incurs a cost of \( w_i(A_i) \), i.e., it consumes \( w_i(A_i) \) units of testing resource, whether or not it detects a defect; there holds \( X_{t+1} = X_t - W_i(A_i) \) if \( t = 0, 1, 2, ... \) (2.1)
8) The cost of removing a detected defect is ignored.
9) A defect detected by action \( A_i \) in state \( \xi \) and removed accordingly generates rebate \( \sigma_i \) (A)
10) There are m+1 possible and different actions in total, i.e., the action set is \( A = \{1, 2, ..., m+1\} \) (action m+1 is to set the software under test to the target state \( \xi = (0, 0) \) while all test resource are used up); however there holds

\[ W_i(A_i) = \begin{cases} \sup_{(x,y)} \{w_i(A_i) (x,y) \} & \text{if } x > 0, y > 0 \\ \text{otherwise} & \end{cases} \]

\[ W_i(m+1) = \begin{cases} \infty & \text{if } y > 0, \min\{w_i(1), w_i(2), ..., w_i(m)\} < x_i \\ x_i & \text{if } y = 0, \min\{w_i(1), w_i(2), ..., w_i(m)\} > x_i \\ \text{otherwise} & \end{cases} \]

(2.2) (2.3)

11) In any state, action m + 1 detects no defect, and \( \sigma_i(m+1) = 0 \) (2.4)
12) \( \xi = (X_t, Y_t) = (0, 0) \) is an absorbing state; it is the target state.
13) The software state transitions depend on the current state \( \xi \) and the action taken at time t.

\[ q_{i,j}(t) = \Pr[X_{t+1} = (x_{t+1}, y_{t+1}) | \xi = (x_t, y_t), A_i = i] \]

\[ \begin{align*}
&= 0 \quad \text{if } y_i = 0, x_i = 0, y_{t+1} > 0, x_{t+1} > 0 \\
&= 1 \quad \text{if } y_i = 0, x_i = 0, y_{t+1} = 0, x_{t+1} = 0 \\
&= \text{undefined} \quad \text{otherwise}
\end{align*} \]

(2.5)

\[ q_{i,j}(m+1) = \Pr[X_{t+1} = (x_{t+1}, y_{t+1}) | \xi = (x_t, y_t), A_i = m+1] \]

\[ \begin{align*}
&= 1 \quad \text{if } y_i = 0, x_i = 0 \\
&= 0 \quad \text{otherwise}
\end{align*} \]

(2.6)

Let \( \tau \) be the first-passage time to state (0,0), and

\[ J_p(x_0, N) = E_{x_0} \sum_{t=0}^{\tau} \theta^t \mathbf{1}(A_i) \]

(2.7)

where \( \theta \) denotes a control policy (testing strategy). Our problem is to find the control policy (testing strategy) that maximizes \( J_p(x_0, N) \). Such a policy detects and removes as many defects as possible within the cost constraints (x0 units of testing resource). From the above assumptions we have the following observations:

- The software testing process makes up a controlled Markov chain.
- The term \( \mathbf{1}(A_i) \) represents the expected reward generated by the action \( A_i \) in state \( \xi \).
- The action can be treated as the probability that each remaining defect is detected by action \( A_i \).
- Suppose \( X_i \neq 0 \), there certainly holds \( X_{i+1} = X_i - W_i(A_i) ; t = 0, 1, 2, ... \)
- If the total testing resource left cannot afford any additional test \( (x < \min\{w_i(1), w_i(2), ..., w_i(m)\} \) , then both software under test moves to the absorbing state (0,0), no matter how many (zero or non-zero) defects are remaining in the software. The software testing process terminates accordingly.
- If all the N defects have been detected and removed, then the software under test no longer needs further testing and moves to the absorbing state (0,0).
- Action m+1 moves the software under test to state (0,0). For convenience of mathematical treatment, we assume that a cost of \( x_i \) is incurred when it is taken. This assures equation (2.1) always holds. However action m+1 is actually a dummy action. In practice it means that no test case is selected or no action is taken. The software testing terminates accordingly.
Proposition 2.1

Proof: From Equations (2.1) to (2.10). Just note that in state (x,0) with x ≠ 0, state (0,y) with y ≠ 0, and state (x,y) with y ≠ 0 and x < \min\{w_{(x,y)}(1), w_{(x,y)}(2), \ldots, w_{(x,y)}(m)\}, action m+1 is the only admissible one. In state (x,y) with y ≠ 0 and x ≥ \min\{w_{(x,y)}(1), w_{(x,y)}(2), \ldots, w_{(x,y)}(m)\}, the set of admissible actions is \( i \in \{ \min\{w_{(x,y)}(k)\} ≤ x \} \).

Proposition 2.1 gives a clear picture of how to test software and defines the required optimal testing strategy. More specifically, in state (x,0) with x ≠ 0, state (0,y) with y ≠ 0, and state (x,y) with y ≠ 0 and x < \min\{w_{(x,y)}(1), w_{(x,y)}(2), \ldots, w_{(x,y)}(m)\}, action m+1 must be taken. In other states, the action

\[ \arg \min_{i \in \{ \min\{w_{(x,y)}(k)\} ≤ x \}} \left\{ w^*_i(v) + y \cdot \theta_i \right\} (x - w_{(x,y)}(i), y - 1) \]

should be taken. In general, in order to determine the optimal action in state (x,y), we must take account of its possible following states (x-w_{(x,y)}(i),1) (x-w_{(x,y)}(i),y), (x-w_{(x,y)}(i)-w_{(x,y)}(j),y-1) , (x-w_{(x,y)}(i)-w_{(x,y)}(j),y-2), (x-w_{(x,y)}(i)-w_{(x,y)}(j),y), and so forth.

Example 2.1

Suppose \( x_0 = 10 \) and \( N = 2 \), that is, at the beginning the software under test contains two defects and at most 10 units can be used to complete the whole test. Assume also the action set consists of four different actions, and they are not all admissible each time. Action 4 (i.e., test case 4) can be used only in state (x,0) with x ≠ 0, state (0,y) with y ≠ 0, and state (x,y) with y ≠ 0 and x < \min\{w_{(x,y)}(1), w_{(x,y)}(2), \ldots, w_{(x,y)}(3)\}. In fact, it is the only admissible action under those conditions. The corresponding defect detection rates are \( \theta_1 = 0.20 \), \( \theta_2 = 0.15 \), and \( \theta_3 = 0.1 \); the defect detection rate for action 4 (test case 4) is 0. The cost of removing detected defects is ignored. The costs of applying test cases and the rebates of detecting defects are listed in Tables 2.1 and 2.2, respectively. These values are only for illustrative purposes. In real-life testing, the possible states and the cost and rebate under each state are defined by the testers as a cost constraint of the testing process. Due to space limits, we have only specialized some of the possible states to explain the software testing process.

Table 2.1 Costs of applying test cases for Example 2.1

<table>
<thead>
<tr>
<th>Action</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
</tr>
</thead>
<tbody>
<tr>
<td>(4.5,1)</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>(7.5,1)</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>(0,1)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(2.5,1)</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>(5.2,1)</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>(7.5,1)</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>(0,1)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 2.2 Rebates of detecting defects for Example 2.1

<table>
<thead>
<tr>
<th>Action</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
</tr>
</thead>
<tbody>
<tr>
<td>(4.5,1)</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>(7.5,1)</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>(0,1)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(2.5,1)</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>(5.2,1)</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>(7.5,1)</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>(0,1)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

From Equation (2.11) we have \( v(x,0) = 0 \) and \( v(x,y) = 0 \), if \( x < 1 \). The values of \( v(x,y) \) in other states appear in Table 2.3. We can see in state \( \lambda = (10,2) \) action (test case 2) should be taken because it generates the highest expected rebate (refer to Table 2.3). Also, from Table 2.1, the cost of applying test case 2 is 1 in state \( \lambda = (10,2) \). If the test case detects a defect and the software under test moves to state \( \lambda = (9,1) \), then action (test case) 1 should be taken. If the test case does not detect a defect and the software under test moves to state \( \lambda = (9,2) \), then action (test case) 2 should be taken again. In short, we may have \( (10, 2) \rightarrow (9,1) \rightarrow (7.5,1) \rightarrow (4.5,1) \rightarrow (3.5,0) \rightarrow (0,0) \) or \( (10, 2) \rightarrow (9,2) \rightarrow (6.2) \rightarrow (5.1) \rightarrow (0.5,1) \rightarrow (0,0) \).

Table 2.3 Expected rebates with various initial states for Example 2.1

<table>
<thead>
<tr>
<th>Action</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
</tr>
</thead>
<tbody>
<tr>
<td>(4.5,1)</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>(7.5,1)</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>(0,1)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(2.5,1)</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>(5.2,1)</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>(7.5,1)</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>(0,1)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

73
3 THE ADAPTIVE TESTING STRATEGY

In Section 2 we implicitly assume that the parameters of concern such as \( N \) and \( \theta_1, \theta_2, \ldots, \theta_6 \) are known. In practice they are unknown and need to be estimated in the course of software testing. What we can observe directly are the actions taken \( \{A_i, t = 0,1,2,\ldots\} \) and the corresponding outputs \( \{Z_i, t = 0,1,2,\ldots\} \). As a result we need to estimate or update the related parameters on-line as new observations come up, and determine the corresponding adaptive actions based on the estimates of these parameters. This leads to an adaptive testing strategy, as depicted in Figure 3.1. We can simply say that adaptive testing is the counterpart to adaptive control in software testing. By adaptive control we mean that the controller should be adjusted on-line in accordance with the changes taking place in the controlled object. A non-adaptive control in software testing. By adaptive control we mean that the controller should be activated by the new observations coming up, and determine the corresponding adaptive actions based on the estimates of these parameters. This leads to an adaptive testing strategy, as depicted in Figure 3.1. We can simply say that adaptive testing is the counterpart to adaptive control in software testing.

**Figure 3.1 Diagram of adaptive software testing**

Suppose that assumptions (1) through (13) presented in Section 2 are still valid and the random variables \( \{Z_i, t = 0,1,2,\ldots\} \) are independent. Denote \( z_i \) as the realization of \( Z_i \). In this way \( r_i \) represents the probability that action \( a_i \) detects a defect. We have

\[
Pr\{Z_i = z_i\} = (r_1)^{i-1}(1-r_1)^{z_i-1}, \quad t = 0,1,2,K
\]  

(3.2)

Here \( r_i \) is determined by equation (3.1) and represents the expected value of \( Z_i \), whereas \( z_i \) is the realization of \( Z_i \) in accordance with the statistical law of equation (3.4). Write

\[
Pr\{Z_i = z_i\} = \left[ \frac{N - \sum_{j=1}^{z_i} \theta_j}{N - \sum_{j=1}^N \theta_j} \right]^{z_i} \left[ 1 - \frac{N - \sum_{j=1}^N \theta_j}{N - \sum_{j=1}^N \theta_j} \right]^{N-z_i}, \quad z_i \in \{0,1,2,K\}
\]

(3.4)

Suppose each of the \( m \) actions has been selected at least once until time \( t \). Then the \( m+1 \) parameters \( N, \theta_1, \theta_2, \ldots, \theta_6 \) can be estimated by minimizing the function \( L(z_1,z_2,\ldots,z_m; a_0, a_1, \ldots, a_m) \) (at least in theory). Denote the resulting estimates as \( N^{(i)}, \theta_1^{(i)}, \theta_2^{(i)}, \ldots, \theta_6^{(i)} \). or

\[
N^{(i)} = \sum_{j=1}^{z_i} \theta_j, \quad \theta_1^{(i)} = \frac{z_1}{N^{(i)}}, \ldots, \theta_6^{(i)} = \frac{z_m}{N^{(i)}}, \quad i = 0,1,2,m
\]

(3.5)

By using the certainty-equivalence principle or the method of substituting the estimates into adaptive stationary controls [18], we treat \( N^{(i)}, \theta_1^{(i)}, \theta_2^{(i)}, \ldots, \theta_6^{(i)} \) as the true values of the corresponding parameters at time \( t+1 \) and take the adaptive action based on these values. Consequently, we obtain the following adaptive control policy (adaptive software testing strategy):

**Step 1** Initialize parameters. Set \( x = x_0 \) and \( z_i = 0, N^{(0)} = N, \theta_0^{(0)} = \theta_0, i = 1,2,\ldots,m \). \( Y_0^{(0)} = 0, t = 0 \). If \( N^{(0)} = 0, \) or \( x < \min\{w_{(1)}, w_{(2)}, \ldots, w_{(m)}(m)\} \), action \( m+1 \) should be taken.

For other cases:

\[
\begin{align*}
A_i &= \arg \min_{a \in \{0,1,2,\ldots\}} \left[ w_i(t) + y \theta_i x - \min_{y \in \{0,1,2,\ldots\}} \left[ (1-\theta_i) w_i(t) + y \theta_i x - w_{(i)}(t) \right] \right]
\end{align*}
\]

(3.6)

Alternatively, randomly choose an action as \( A_0 \).

**Step 2** Observe the testing result \( z_i = z_i \) activated by the action \( A_0 \).

**Step 3** Estimate parameters by minimizing \( L(z_0,z_1,\ldots,z_m; a_0, a_1, \ldots, a_m) \) (refer to equation (3.3)) and obtain \( N^{(i)} = \sum_{j=1}^{z_i} \theta_j, \theta_1^{(i)}, \theta_2^{(i)}, \ldots, \theta_6^{(i)} \).

**Step 4** Update the current software state by setting

\[
x = x^*, y = Y_0^{(i)} = \sum_{j=1}^{z_i} z_j
\]

(3.7)

**Step 5** Decide the adaptive action.

Action \( m+1 \) should be taken, if \( Y_i = 0 \) or \( x < \min\{w_{(1)}, w_{(2)}, \ldots, w_{(m)}(m)\} \).

For other cases,

\[
A_i = \arg \min_{a \in \{0,1,2,\ldots\}} \left[ w_i(t) + y \theta_i x - w_{(i)}(t) \right]
\]

(3.8)

**Step 6** Observe the testing result \( z_{i+1} = z_{i+1} \) activated by the action \( A_{i+1} \).

**Step 7** Set \( t = t+1 \).

**Step 8** If \( x = 0 \), then stop testing; otherwise go to Step 3.

4 SIMULATION RESULTS

In order to justify the effectiveness of the optimal testing strategy presented in Section 2 and the adaptive testing strategy presented in Section 3, we perform simulations and comparisons for the optimal, adaptive, and random testing strategies. In the random testing strategy a uniform probability distribution is used to randomly select an action (test case) each time; that is, each admissible action has an equal chance of being selected. In the optimal testing strategy we assume that the true values of all the software parameters of concern are known a priori. In the adaptive testing strategy the true values of the software parameters of concern are unknown and must be estimated on-line during software testing as described in the methods of Section 3. The simulation examples presented in this section use a genetic algorithm for this purpose.

**Example 4.1** Suppose \( x_0 = 18, N = 4, A = \{1,2,3\}, \theta_1 = 0.1, \) and \( \theta_0 = 0.01 \). Further, \( w_1(t) = 2.5, \quad w_2(t) = 1.5, \quad \sigma_1(t) = 1.0, \) and \( \sigma_2(t) = 2.0 \). We perform simulation for 8 runs with respect to the optimal, adaptive, and random testing strategies, respectively. Tables 4.1 and 4.2 give the corresponding simulation results. We notice that the optimal testing strategy performs better than the adaptive testing strategy, which in turn performs better than the random testing strategy. This is because the optimal testing strategy assumes that \( N = 4, \theta_1 = 0.1, \) and \( \theta_0 = 0.01 \) are accurately known a priori. The adaptive testing strategy uses the testing data...
collected during testing to estimate $N$, $\theta_1$, and $\theta_2$ and tries to improve these estimates. The random testing strategy ignores all such information (i.e., does not take the advantages of $N=4$, $\theta_1=0.1$, $\theta_2=0.01$, $w_1(1)=2.5$, $w_2(2)=1.5$, $\sigma_1(1)=1.0$, and $\sigma_2(2)=2.0$). We assume each testing action has an equal probability of being selected. By using up the total 18 units of testing resource, the optimal testing strategy detects 2.63 defects and generates 7.88 units of rebate on average in the eight runs of simulation. In comparison, the adaptive testing strategy detects 2.13 defects and generates 6.25 units of rebate, whereas the random testing strategy detects 1.50 defects and generates 4.38 units of rebate. The optimal testing strategy detects about 6% more defects and generates about 21% more rebates than the random testing strategy. In comparison with Example 4.1, the changes in rebates of detecting defects (from $\sigma_1(1)=1.0$, $\sigma_2(2)=2.0$) do not affect the performance of the testing strategies much.

### Table 4.2 Statistics of optimal, adaptive, and random testing for Example 4.1

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_1(1)$</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1.50</td>
</tr>
<tr>
<td>$n_2(2)$</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>3.50</td>
</tr>
<tr>
<td>$n_3(3)$</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>4</td>
<td>6</td>
<td>5.25</td>
</tr>
<tr>
<td>$t_{o}(1)$</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>6</td>
<td>4.38</td>
</tr>
<tr>
<td>$t_{a}(1)$</td>
<td>6</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6.25</td>
</tr>
<tr>
<td>$t_{r}(1)$</td>
<td>3</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>12</td>
<td>9</td>
<td>9</td>
<td>7.88</td>
</tr>
</tbody>
</table>

### Example 4.2

Suppose $x_0=18$, $N=4$, $A=\{1,2,3\}$ and $\theta_1=0.1$, $\theta_2=0.05$. Further, $w_1(1)=2.5$, $w_2(2)=1.5$, $\sigma_1(1)=1.0$, and $\sigma_2(2)=2.0$. In comparison with Example 4.1, the difference between $\theta_1$ and $\theta_2$ becomes smaller. This implies that the performance discrepancies among the optimal testing strategy, the adaptive testing strategy, and the random testing strategy should tend to be narrow. We do simulation for 8 runs for each of the three testing strategies. Table 4.3 summarizes the corresponding simulation results. The optimal testing strategy detects about 10% more defects and generates about 12% more rebates than the adaptive testing strategy, which in turn detects about 6% more defects and generates about 13% more rebates than the random testing strategy.

### Table 4.3 Statistics of optimal, adaptive, and random testing for Example 4.2

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_1(1)$</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2.25</td>
</tr>
<tr>
<td>$n_2(2)$</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2.50</td>
</tr>
<tr>
<td>$n_3(3)$</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>6</td>
<td>4.75</td>
</tr>
<tr>
<td>$t_{o}(1)$</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>6.00</td>
</tr>
<tr>
<td>$t_{a}(1)$</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9.50</td>
</tr>
</tbody>
</table>

### Example 4.3

Suppose $x_0=18$, $N=4$, $A=\{1,2,3\}$ and $\theta_1=0.1$, $\theta_2=0.05$. Further, $w_1(1)=2.5$, $w_2(2)=1.5$, $\sigma_1(1)=1.0$, and $\sigma_2(2)=2.0$. We do simulation for 8 runs with respect to the optimal, adaptive, and random testing strategies, respectively. Table 4.4 summarizes the corresponding simulation results. The optimal testing strategy detects about 18% more defects and generates about 18% more rebates than the adaptive testing strategy, which in turn detects about 21% more defects and generates about 21% more rebates than the random testing strategy. In comparison with Example 4.1, the changes in rebates of detecting defects (from $\sigma_1(1)=1.0$, $\sigma_2(2)=2.0$) do not affect the performance of the testing strategies much.

### Table 4.4 Statistics of optimal, adaptive, and random testing for Example 4.3

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_1(1)$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1.75</td>
</tr>
<tr>
<td>$n_2(2)$</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2.13</td>
<td></td>
</tr>
<tr>
<td>$n_3(3)$</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>14</td>
<td>7</td>
<td>14</td>
<td>14</td>
<td>12.25</td>
</tr>
<tr>
<td>$t_{o}(1)$</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>14</td>
<td>7</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14.88</td>
</tr>
<tr>
<td>$t_{a}(1)$</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>14</td>
<td>7</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14.88</td>
</tr>
</tbody>
</table>

Suppose $x_0=18$, $N=4$, $A=\{1,2,3\}$ and $\theta_1=0.1$, $\theta_2=0.05$. Further, $w_1(1)=2.5$, $w_2(2)=1.5$, $\sigma_1(1)=1.0$, and $\sigma_2(2)=2.0$. We do simulation for 8 runs with respect to the optimal, adaptive, and random testing strategies, respectively. Table 4.5 summarizes the corresponding simulation results. The optimal testing strategy detects about 6% more defects and generates about 10% more rebates than the adaptive testing strategy, which in turn detects about 6% more defects and generates about 24% more rebates than the random testing strategy. In comparison with Example 4.1, the changes in defect detection rates (from $\theta_1=0.1$, $\theta_2=0.05$) substantially affect the performance of the testing strategies. In conjunction with Examples 4.1 and 4.2, we suggest that software defect detection rates play a dominant role in evaluating the performance of various software testing strategies.

Overall, the adaptive testing strategy is feasible and works well for the above four examples. In comparison with the random testing strategy, it detects more defects and generates more rebates, although it cannot compete with the optimal testing strategy. However we should note that the optimal testing strategy is infeasible in practice since accurate values of software detection rates are not known a priori.
testing.

In terms of the performance of optimal, adaptive, and random case, as shown in Example 5.1, we still have the same conclusion. The consequence of using Equation (5.1) is that each action taken takes into account. For example, let \( x_0=30, N=8, A=1,2,3, \), \( \varepsilon_1 (1)=2.0 \) and \( \varepsilon_1 (2)=1.5 \). In calculating \( v(30,8) \), we need to calculate \( v(28,8), v(28,7), v(28,5,8) \), and \( v(28,5,7) \), which in turn require four more calculations each. This implies to calculate \( v(28,8) \), four additional calculations (namely, \( v(26,8), v(26,7), v(26,5,8), \) and \( v(26,5,7) \)) are required. This process continues until all the resources are used up. This may lead to a potential combinatorial explosion problem in calculation.

In order to reduce the computational complexity posed on Equation (2.11), we do not adhere to considering all possible subsequent states of a given state \((x,y)\). Rather, only those states that can possibly be reached within 10 actions are considered. As a result, only a reasonable amount of calculations is required in order to decide the action that should be taken in any state. More specifically, for given \( x \) and \( y \), let \( x_1=x-10(\min(n^*(i))) \). The following equation is used to replace Equation (2.11) for the optimal testing and the adaptive testing strategy.

\[
v(x,y) = \begin{cases} 
0 & \text{if } x=0 \text{ or } y=0 \text{ or } x<y < \min\{\varepsilon_1 (1), \varepsilon_1 (2), K, \varepsilon_1 (m)\} \\
\min[\psi(i)] & \text{if } x \leq x_1, y \neq 0 \\
\min \left[ \psi (i) + \frac{1}{1-\theta} \xi \left( x - \varepsilon_1 (i), y \right) \right] & \text{if } y=0, x \geq x_1, \text{and } x \neq \min\{\varepsilon_1 (1), \varepsilon_1 (2), K, \varepsilon_1 (m)\} \\
\min \left[ \psi (i) + \frac{1}{1-\theta} \xi \left( x - \varepsilon_1 (i), y \right) \right] & \text{if } y>0, x \geq x_1, \text{and } x \neq \min\{\varepsilon_1 (1), \varepsilon_1 (2), K, \varepsilon_1 (m)\}
\end{cases}
\]

The consequence of using Equation (5.1) is that each action taken may not necessarily be optimal. However, even that might be the case, as shown in Example 5.1, we still have the same conclusion in terms of the performance of optimal, adaptive, and random testing.

Example 5.1 Suppose \( x_0=300, N=10, A=1,2,3, \), \( \theta_1=0.1, \) and \( \theta_2=0.002 \). Further, \( \varepsilon_1 (1)=2.5, \varepsilon_1 (2)=1.5, \varepsilon_2 (1)=3.0, \) and \( \varepsilon_2 (2)=1.0 \). In comparison with the examples presented in Section 4, the problem size is substantially increased (from \( x_0=18, N=4 \) to \( x_0=300, N=10 \)). We do simulation for eight runs with respect to the optimal, adaptive, and random testing strategies, respectively. Tables 5.1, 5.2, and 5.3 give the corresponding simulation results. The optimal testing strategy detects about 8% more defects and generates about 8% more rebates than the adaptive testing strategy, which in turn detects about 21% more defects and generates about 31% more rebates than the random testing strategy. The effectiveness of the optimal testing strategy and the adaptive testing strategy is once again confirmed even if an approximate algorithm (with Equation (2.11) being replaced by Equation (5.1)) is used.

### Table 5.1 Number of test cases used for optimal, adaptive, and random testing for Example 5.1

<table>
<thead>
<tr>
<th>( n )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n_o(1) )</td>
<td>14</td>
<td>28</td>
<td>50</td>
<td>53</td>
<td>64</td>
<td>66</td>
<td>68</td>
<td>102</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>( n_o(2) )</td>
<td>14</td>
<td>28</td>
<td>54</td>
<td>65</td>
<td>83</td>
<td>102</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>( n_o(3) )</td>
<td>14</td>
<td>28</td>
<td>49</td>
<td>57</td>
<td>73</td>
<td>80</td>
<td>94</td>
<td>115</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>( n_a(1) )</td>
<td>19</td>
<td>28</td>
<td>47</td>
<td>69</td>
<td>122</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>( n_a(2) )</td>
<td>21</td>
<td>30</td>
<td>51</td>
<td>64</td>
<td>80</td>
<td>94</td>
<td>93</td>
<td>98</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>( n_a(3) )</td>
<td>5</td>
<td>9</td>
<td>10</td>
<td>13</td>
<td>24</td>
<td>29</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>( n_r(1) )</td>
<td>3</td>
<td>9</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>23</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>( n_r(2) )</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>136</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>( n_r(3) )</td>
<td>4</td>
<td>7</td>
<td>9</td>
<td>12</td>
<td>15</td>
<td>18</td>
<td>21</td>
<td>23</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

\( \xi \) denote the rest of the software testing process detects no more defects.

### Table 5.2 Testing rebate generated for optimal, adaptive, and random testing for Example 5.1

<table>
<thead>
<tr>
<th>( \sigma )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma_o(1) )</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>5.50</td>
<td></td>
</tr>
<tr>
<td>( \sigma_o(2) )</td>
<td>6</td>
<td>9</td>
<td>9</td>
<td>6</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>7.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \sigma_o(3) )</td>
<td>4</td>
<td>7</td>
<td>7</td>
<td>9</td>
<td>6</td>
<td>7</td>
<td>8.50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \sigma_a(1) )</td>
<td>18</td>
<td>27</td>
<td>30</td>
<td>27</td>
<td>12</td>
<td>18</td>
<td>21</td>
<td>23.25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \sigma_a(2) )</td>
<td>19</td>
<td>27</td>
<td>30</td>
<td>27</td>
<td>12</td>
<td>18</td>
<td>21</td>
<td>23.25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \sigma_a(3) )</td>
<td>21</td>
<td>27</td>
<td>30</td>
<td>27</td>
<td>12</td>
<td>18</td>
<td>21</td>
<td>23.25</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The notation \( ** \) denotes the rest of the software testing process detects no more defects.

### Table 5.3 Statistics of optimal, adaptive, and random testing for Example 5.1

<table>
<thead>
<tr>
<th>( \tau )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \tau_o(1) )</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>5.50</td>
</tr>
<tr>
<td>( \tau_o(2) )</td>
<td>6</td>
<td>9</td>
<td>9</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>7.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \tau_o(3) )</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>9</td>
<td>6</td>
<td>7</td>
<td>7.50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \tau_a(1) )</td>
<td>18</td>
<td>27</td>
<td>30</td>
<td>27</td>
<td>12</td>
<td>18</td>
<td>21</td>
<td>21.38</td>
<td></td>
</tr>
<tr>
<td>( \tau_a(2) )</td>
<td>19</td>
<td>27</td>
<td>30</td>
<td>27</td>
<td>12</td>
<td>18</td>
<td>21</td>
<td>21.38</td>
<td></td>
</tr>
<tr>
<td>( \tau_a(3) )</td>
<td>21</td>
<td>27</td>
<td>30</td>
<td>27</td>
<td>12</td>
<td>18</td>
<td>21</td>
<td>21.38</td>
<td></td>
</tr>
</tbody>
</table>

The notation \( ** \) denotes the rest of the software testing process detects no more defects.
6 CONCLUDING REMARKS AND FUTURE WORK

We have successfully demonstrated that software testing with fixed resource constraints can be modeled by the controlled Markov chains (CMC) approach. We report an algorithm to reduce the computational complexity of on-line decision making in optimal testing and adaptive testing. Our simulation results confirm the effectiveness of the idea of adaptive software testing in particular, and that of software cybernetics in general. We also observe that how much better the optimal testing strategy and the adaptive testing strategy perform than the random testing strategy is largely dependent on the discrepancies among the software defect detection rates of various classes of test cases.

The importance of the theoretical and simulation results presented in this paper stems from the observation that the proposed adaptive testing strategy can effectively apply to a wide class of software testing problems with testing resource constraints. The adaptive testing strategy serves as the counterpart to adaptive control in software testing. In contrast to conventional software testing strategies that normally define a test suite off-line [19-21], the adaptive testing strategy selects test cases one by one on-line according to some specific optimization criteria. This should improve the current status of software testing, which is arguably the least understood part of software development and heavily depends on tester experience and rules of thumb [22].

Several topics deserve further investigation in the future including a theoretical analysis of the performance of the proposed adaptive testing strategy, case studies of the adaptive testing strategy on real-life industry applications, and more widely applicable models that take account of the architecture of the software under test.

7 REFERENCES