A Case Study of the Recursive Least Squares Estimation Approach to Adaptive Testing for Software Components

Hai Hu[†], W. Eric Wong
Department of Computer Science
University of Texas at Dallas
Richardson, TX 75083
ewong@utdallas.edu

Chang-Hai Jiang, Kai-Yuan Cai
Department of Automatic Control
Beijing University of Aeronautics and Astronautics
Beijing 100083, China
kycai@buaa.edu.cn

Abstract

The strategy used for testing a software system should not be fixed, because as time goes on we may have a better understanding of the software under test. A solution to this problem is to introduce control theory into software testing. We can use adaptive testing where the testing strategy is adjusted on-line by using the data collected during testing. Since the use of software components in software development is increasing, it is now more important than ever to adopt a good strategy for testing software components. In this paper, we use an adaptive testing strategy for testing software components. This strategy (AT_RLSE$_c$ with c indicating components) applies a recursive least squares estimation (RLSE) method to estimate parameters such as failure detection rate. It is different from the genetic algorithm-based adaptive testing (AT_GA) where a genetic algorithm is used for parameter estimation. Experimental data from our case study suggest that the fault detection effectiveness of AT_RLSE$_c$ is better than that of AT_GA and random testing.

Keywords: software testing, adaptive testing, controlled Markov chain, software cybernetics.

1. Introduction

The strategy used for testing a software system should not be fixed, because as time goes on we may have a better understanding of the software under test. This understanding can help us adjust the underlying testing strategy to make it more effective in detecting software defects. Adaptive testing, first proposed in [4][5][8], provides a way to accomplish this by applying software cybernetics and controlled Markov chains (CMC) to software testing. Software cybernetics explores the interplay between software theory/engineering and control theory/engineering, whereas the CMC approach to software testing treats software testing as a control problem. The software under test serves as a controlled object which is modeled by a CMC, and the testing strategy serves as the corresponding controller. The software under test and the testing strategy make up a closed-loop feedback control system. Adaptive testing is the counterpart to adaptive control in software testing. It means that the software testing strategy should be adjusted on-line by using data collected during testing, as our understanding of the software under test is improved. Figure 1 gives a graphical overview of adaptive testing.

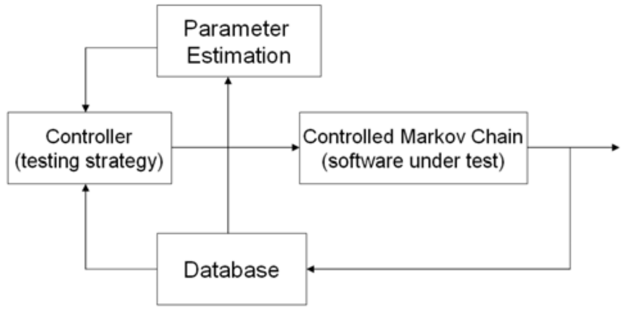
Parameter Estimation
Controller (testing strategy)
Controlled Markov Chain (software under test)
Database

Figure 1 A graphical overview of adaptive testing

The parameter estimation module uses testing history data saved in the database to estimate parameters on-line. Such data are also used by the controller, along with the estimated parameters, to make necessary adjustments to the testing strategy.

The use of software components in software development has increased significantly since the last decade. It is necessary to have a good testing strategy in response to this new development paradigm. Our previous experience in using adaptive testing shows that better fault detection effectiveness can be achieved than with random testing. This motivates us to explore the possibility of applying a modified adaptive testing strategy to testing software components [6]. There are two

[†]Mr. Hai Hu currently is a visiting scholar from Beijing University of Aeronautics and Astronautics.

major differences between the modified approach and the original adaptive testing proposed in [4][5][8]. First, the controlled object (refer to Figure 1) in the modified approach represents an individual component rather than an entire software system as in the original approach. Second, the modified approach uses a recursive least squares estimation (RLSE) method, rather than a genetic algorithm (GA) as in the original adaptive testing, to estimate failure detection rates for different test classes, where each test class contains a number of test cases randomly generated based on the specifications of the program being tested. Hereafter, we use AT_RLSE$_c$ to represent the above modified adaptive testing with c indicating it is applied to testing software components, and AT_GA for the original adaptive testing. A case study on a SPACE program received from the European Space Agency [1] was conducted to compare the fault detection effectiveness of AT_RLSE$_c$ with that of AT_GA and random testing (hereafter denoted as RT).

The rest of this paper is organized as follows: In Section 2, we explain the general mechanism of adaptive testing in the CMC model and describe the problem of how to test software components more effectively as a problem of optimal software testing with resource constraints. The mathematical deduction of RLSE and its corresponding control policy is presented in Section 3. Our case study and fault detection effectiveness comparison among AT_RLSE$_c$, AT_GA, and RT appear in Section 4. Finally, in Section 5, we offer our conclusions and future directions.

2. Methodology of Adaptive Software Component Testing

Software testing is often limited by resources (time, budget, etc.) which have to be kept within specified bounds. In this paper we choose the number of tests as the constraint by assuming software testing will be terminated after a pre-defined maximum number of tests have been executed. If the same input is used more than once, it is counted as more than one test. For example, if an input is used for the second and the fifth executions, then we say the second and the fifth tests have the same input. A testing action refers to an execution of one test case on the software under test.

Let the first and the i^{th} tests be applied to the software under test at time $t = 0$ and $t = i$ -1, respectively; M be the maximum number of tests; and x_t be the remaining number of test cases at time t (inclusive) that can be executed. This implies $x_0 = M$, $x_1 = M$-1, etc. Let also

$$y_t = \begin{cases} 1 & \text{if the action taken at time } (t-1) \text{ reveals a failure} \\ 0 & \text{if the action taken at time } (t-1) \text{ does not reveal a failure} \end{cases}$$

$$Z_t = \begin{cases} 1 & \text{if the action taken at time } t \text{ reveals a failure} \\ 0 & \text{if the action taken at time } t \text{ does not reveal a failure} \end{cases}$$

$$\xi_t = (x_t, y_t);\, t = 0,1,2,\ldots$$

where ξ_t describes the state of the software component under test at time t, and Z_t indicates whether a failure is observed by the testing action at time t. For a given ξ_t, the testing action at time t-1 reveals y_t failure, and the remaining testing resource allows the software to be tested for at most x_t tests (including the one applied at time t).

We also make the following assumptions:

1. One and only one action is taken at each time.
2. At any given instant there are always m admissible actions; the action set[2] is $A = \{1, 2, \ldots, m\}$.
3. Action A_t taken at time t incurs a cost of $W_{\xi_t}(A_t)$, no matter whether it triggers a failure or not;
$$W_{\xi_t}(A_t = i) = \begin{cases} w_{(x_t,y_t)}(i) & \text{if } x_t \neq 0 \\ 0 & \text{if } x_t = 0 \end{cases} \quad (1)$$
where $w_{(x_t,y_t)}(i)$ indicates the cost of taking the i^{th} action at state ξ_t.
4. Actions can be taken to test the software for exactly M times.
5. An action taken at one time triggers at most one failure.
6. The component under test remains unchanged during testing, i.e., no debugging is performed for the component under test even if a failure is observed.
7. $\xi_t = (x_t, y_t) = (0, 0)$ or $(0, 1)$ is an absorbing state; it is the target state.
8. The state transitions depend on the current state ξ_t and the action taken at time t,
$$q_{\xi_t\xi_{t+1}}(i) = \Pr\{\xi_{t+1} = (x_{t+1}, y_{t+1}) \mid \xi_t = (x_t, y_t), A_t = i\}$$
$$= \begin{cases} 1-\theta_i & ;\text{ if } y_{t+1} = 0, x_{t+1} = x_t - 1, x_t \neq 0 \\ \theta_i & ;\text{ if } y_{t+1} = 1, x_{t+1} = x_t - 1, x_t \neq 0 \\ 1 & ;\text{ if } y_{t+1} = 0, x_{t+1} = 0, x_t = 0 \\ 0 & ;\text{ otherwise} \end{cases} \quad (2)$$
where $q_{\xi_t\xi_{t+1}}$ is the transition probability between state ξ_t and ξ_{t+1}, and θ_t is the probability of detecting a failure by taking the i^{th} action.

[2] Do not mistreat the action set as the testing resource. The action set means that there are m different actions that can be taken at any given time. The testing resource refers to how many tests are allowed to apply to the software.

9. Action A_t taken at state ξ_t gives a rebate $\sigma_{\xi_t}(A_t)$ if it triggers a failure. A rebate is defined as the "benefit" that a tester may receive due to the detection of a failure.

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

$$J_\omega(M) = E_\omega \sum_{t=0}^{\tau} \left[W_{\xi_t}(A_t) - \theta_{A_t} \sigma_{\xi_t}(A_t) \right] \tag{3}$$

where $J_\omega(M)$ is the total cost for all M actions and ω is a control policy (i.e., a testing strategy). Our objective is to find a testing strategy that minimizes $J_\omega(M)$. Such a strategy uses M tests to maximize the overall rebate while minimizing the total cost incurred by executing these test cases. Let $W^*_{\xi_t}(A_t) = W_{\xi_t}(A_t) - \theta_{A_t}\sigma_{\xi_t}(A_t)$ where $W^*_{\xi_t}(A_t)$ is the net cost for each action taken (i.e., net cost = cost - rebate). The term $\theta_{A_t}\sigma_{\xi_t}(A_t)$ is the expected reward generated by the action A_t at state ξ_t. Note that $W^*_{\xi_t}(A_t)$ can be positive or negative. However $W^*_{(x_t, y_t)}(A_t) = 0$ when $x_t = 0$. We can rewrite Equation (3) as

$$J_\omega(M) = E_\omega \sum_{t=0}^{\tau} \left[W^*_{\xi_t}(A_t) \right] \tag{4}$$

The problem of minimizing $J_\omega(M)$ is highly related to the so-called test case prioritization problem [3]. This problem is concerned with how to select a permutation of a given test suite such that a given quantitative function (e.g., rate of defect detection) is optimized. An extreme case is that each test case in the given test suite corresponds to an action and the corresponding set of admissible actions consists of those test cases not selected yet. We can also partition the test suite into a number of classes and each action corresponds to one class.

From the theory of controlled Markov chains [2][9] we can conclude that there exists a deterministic stationary that minimizes $J_\omega(M)$. According to the method of successive approximation, let

$$v_{n+1}(\xi) = \min_{1\le i\le m} \left\{ w^*_\xi(i) + \sum_{\eta \ne (0,0)} q_{\xi\eta}(i) v_n(\eta) \right\} \tag{5}$$

Let $$v(\xi) = \lim_{n\to\infty} v_n(\xi) \tag{6}$$

We have the following proposition.

Proposition: Under assumptions 1 to 9, there holds

$$v(x,y) = \begin{cases} 0 & ;\text{if } x=0 \\ \min\limits_{1\le i\le m} \left\{ w^*_\xi(i) \right\} & ;\text{if } x=1 \\ \min\limits_{1\le i\le m} \left\{ w^*_\xi(i) + \theta_i v(x-1,1) + (1-\theta_i) v(x-1,0) \right\} & \\ & ;\text{if } x>1 \end{cases} \tag{7}$$

where $v(x, y)$ is the minimum possible cost at state (x, y).

Proof: Trivial from Equations (1) to (6).

The above proposition gives a clear picture of how to test software. At state (0, y), any action can be taken. At state (1, y), the action $\arg \min\limits_{1\le i\le m} \left\{ w^*_{(1,y)}(i) \right\}$ should be taken. At other states the action $\arg \min\limits_{1\le i\le m} \left\{ w^*_\xi(i) + \theta_i v(x-1,1) + (1-\theta_i) v(x-1,0) \right\}$ should be taken. In general, in order to decide the optimal action at state (x, y), we also need to take account of its possible following states, (x-1, 1), (x-1, 0), (x-2, 1), (x-2, 0), etc. Note that the optimal testing strategy determined by the aforementioned proposition can apply only if the true values of θ_1, θ_2,…, θ_m are accurately known a priori. Otherwise, we have to follow an adaptive testing strategy as described in Section 3.

3. Least Squares Estimation & RLSE

3.1 Least Squares Estimation

Suppose that assumptions 1 through 9 in Section 2 are still valid and the random variables $\{Z_t, t = 0, 1, 2,\ldots\}$ are independent. Denote $z_{-1}=0$. Let $\{z_t, t = 0, 1, 2,\ldots\}$ be a realization of $\{Z_t, t = 0, 1, 2,\ldots\}$ and

$$r_t = \theta_{a_t} \tag{8}$$

where a_t is a realization of A_t. In this way r_t represents the probability that action a_t triggers a failure. We have

$$\Pr\{Z_t = z_t\} = (r_t)^{z_t} (1-r_t)^{1-z_t} ; t = 0,1,2,... \tag{9}$$

Given z_0, z_1,…, z_t and a_0, a_1,…, a_t, following the least squares method of parameter estimation, we have

$$L(z_0, z_1, ..., z_t; a_0, a_1, ..., a_t) = \sum_{j=0}^{t} (z_j - r_j)^2 \tag{10}$$

Here r_j is determined by Equation (8) and represents the expected value of Z_j, whereas z_j is the realization of Z_j in accordance with the statistical law of Equation (9). Suppose each of the m actions has been selected at least once up to time t; then the m parameters θ_1, θ_2,…, θ_m can be estimated by minimizing the function L (z_0, z_1,…, z_t; a_0, a_1,…, a_t) (at least in theory). Denote the resulting estimates as θ_1^{t+1}, θ_2^{t+1} ,..., θ_m^{t+1}. We have[3]

$$\theta_i^{(t+1)} = \theta_i(z_0, z_1, \ldots, z_t; a_0, a_1, \ldots, a_t); i = 1,2, ..., m \tag{11}$$

By using the certainty-equivalence principle or the method of substituting the estimates into optimal stationary controls [9], we treat θ_1^{t+1}, θ_2^{t+1} ,..., θ_m^{t+1} as the true values of the corresponding parameters at time t+1 and take the optimal action based on these values.

[3] A commonly used way of parameter estimation is to differentiate $L(z_0, z_1, \ldots, z_t; a_0, a_1, \ldots, a_t)$ with respect to each parameter of concern and obtain a system of nonlinear equations. However, sometimes the system of these nonlinear equations may not have a solution. So, a more general way of doing parameter estimation is to minimize $L(z_0, z_1, \ldots, z_t; a_0, a_1, \ldots, a_t)$ directly.

Consequently, we obtain the following adaptive control policy (adaptive software testing strategy):

Step 1 Initialize parameters. Set $x = M$ and

$$z_{-1} = 0, , \theta_i^{(0)} = \theta_{0i} \quad ; i = 1, 2, \ldots, m$$

$$Y_0 = 0, \quad t = 0;$$

If $M = 1$, then $A_0 = \arg \min_{1 \le i \le m} \left\{ w^*_{(1,0)}(i) \right\}$.

For other cases,

$$A_0 = \arg\min_{1\le i\le m} \left\{ w^*_{(M,0)}(i) + \theta_i v(M-1,1) + (1-\theta_i) v(M-1,0) \right\}$$

Alternatively, randomly choose an action as A_0.

Step 2 Observe the testing result $Z_0 = z_0$ activated by the action A_0.

Step 3 Estimate parameters by minimizing[4] L ($z_0, z_1, \ldots, z_t$; $a_0, a_1, \ldots, a_t$) (Equation (10)) and obtain

$$\theta_i^{(t+1)} = \theta_i(z_0, z_1, \ldots, z_t; a_0, a_1, \ldots, a_t) \quad ; i = 1, 2, \ldots, m$$

Step 4 Update the current software state by setting $x = x - 1$.

Step 5 Decide the optimal action. If $x = 1$, then

$$A_{t+1} = \arg \min_{1\le i\le m} \left\{ w^*_{(x,y)}(i) \right\}.$$

For other cases,

$$A_{t+1} = \arg \min_{1\le i\le m} \left\{ w^*_{(M,y)}(i) + \theta_i v(M-1,1) + (1-\theta_i) v(M-1,0) \right\}$$

Step 6 Observe the testing result $Z_{t+1} = z_{t+1}$ activated by the action A_{t+1}.

Step 7 Set $t = t + 1$.

Step 8 If $x = 0$, then stop testing[5]; otherwise go to *Step 3*.

3.2 Recursive Least Squares Estimation (RLSE)

To estimate $\theta_1, \theta_2, \ldots, \theta_m$ from Equation (10), we can apply a genetic algorithm as shown in our previous work [4][8]. However the genetic algorithm is computationally intensive since it needs to employ the whole history of testing data to do global optimization on-line for each test case. To alleviate this problem, here we use the recursive least squares estimation method to estimate $\theta_1, \theta_2, \ldots, \theta_m$. Parameters are no longer re-estimated for each test case by using the whole history of testing data. Instead, only new observed testing data are used to update the existing estimates of the parameters.

Note that a test case or action may or may not reveal a failure. For the convenience of mathematical notation, we write $u_i = 1$ if the i^{th} action is taken and $u_i = 0$ otherwise. The expected output corresponding to $u_i = 1$ should be θ_i before the M tests are used up. We write the expected output as $g = \theta_i$. Consequently, we have

$$g = \theta_1 f_1(u) + \theta_2 f_2(u) + \ldots + \theta_m f_m(u) \tag{12}$$

where $u = [u_1, u_2, \ldots, u_m]^T$ is a column vector, T denotes the transpose of a matrix, and

Let
$$f_i(u) = u_i; i = 1, 2, \ldots, m \tag{13}$$

where $f_i(u)$ is a function for estimation calculation.

Equation (12) identifies the ideal function we should have. However, in practice the observed value of g should either be 1 or 0, depending on whether a failure is observed. Suppose the observed value of g is $g^{(l)}$ if $u = u^{(l)}$.

$$\begin{aligned} &\theta_1 f_1\left(u^{(1)}\right) + \theta_2 f_2\left(u^{(1)}\right) + \cdots + \theta_m f_m\left(u^{(1)}\right) = g^{(1)} \\ &\theta_1 f_1\left(u^{(2)}\right) + \theta_2 f_2\left(u^{(2)}\right) + \cdots + \theta_m f_m\left(u^{(2)}\right) = g^{(2)} \\ &\ldots \\ &\theta_1 f_1\left(u^{(n)}\right) + \theta_2 f_2\left(u^{(n)}\right) + \cdots + \theta_m f_m\left(u^{(n)}\right) = g^{(n)} \end{aligned} \tag{14}$$

That is, n actions are applied and we have n pairs of testing data $\left\{\langle a_t, z_t \rangle; t = 0, 1, \ldots, n-1\right\}$ (in the notation of Section 2) or $\left\{\left\langle u^{(l)}, g^{(l)} \right\rangle; l = 1, 2, \ldots, n\right\}$.

Write Equation (14) in a matrix form,

$$D\theta = g \tag{15}$$

Where

$$D = \begin{bmatrix} f_1\left(u^{(1)}\right) & f_2\left(u^{(1)}\right) & \ldots & f_m\left(u^{(1)}\right) \\ f_1\left(u^{(2)}\right) & f_2\left(u^{(2)}\right) & \ldots & f_m\left(u^{(2)}\right) \\ \vdots & & & \\ f_1\left(u^{(n)}\right) & f_2\left(u^{(n)}\right) & \ldots & f_m\left(u^{(n)}\right) \end{bmatrix}$$

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \ldots \\ \theta_m \end{bmatrix}, \qquad g = \begin{bmatrix} g^{(1)} \\ g^{(2)} \\ \ldots \\ g^{(n)} \end{bmatrix} \tag{16}$$

where D is a matrix for calculation that contains a history of previous actions and g is a vector that contains a history of testing results (whether a failure has been observed).

In order to have an exact solution of Equation (15), there should hold $n = m$. However, in practice we usually have $n > m$, and thus we can only have an approximate solution.

[4] If θ_i doesn't appear in $L(z_0, z_1, \ldots, z_t; a_0, a_1, \ldots, a_t)$, or action i has not been taken in the previous actions up to time t, then we let $\theta_i^{(t+1)} = \theta_i^{(t)}$.

[5] A theoretical stopping criterion is $Y_{t+1} = 0$. However in practice $Y_{t+1} = 0$ can seldom be satisfied since $N^{(t+1)}$ is just an estimate of N.

Let

$$D\theta + e = g \qquad (17)$$

where e is an error column vector. A desired approximate solution should minimize e in some sense. Denote

$$\begin{aligned} E(e) &= \sum_{l=1}^{n}\left(g^{(l)} - \theta_1 f_1\left(u^{(l)}\right) - \theta_2 f_2\left(u^{(l)}\right) - \ldots - \theta_m f_m\left(u^{(l)}\right)\right)^2 \\ &= e^T e = \left(g - D\theta\right)^T\left(g - D\theta\right) \end{aligned} \qquad (18)$$

where e^{T} is the transpose of e. Our goal is to minimize $E(e)$. Let the resulting approximate solution of θ be $\theta^{(k)}$ by using $\left\{\left\langle u^{(l)}, g^{(l)}\right\rangle; l = 1, 2, \ldots, k\right\}$. The recursive least squares method employs $\theta^{(k)}$ and $\left\langle u^{(k+1)}, g^{(k+1)}\right\rangle$ to obtain $\theta^{(k+1)}$. Specifically, we have

$$\begin{cases} P^{(0)} = \left(D^T D\right)^{-1} \\ P^{(n+1)} = P^{(n)} - \dfrac{P^{(n)} d_{n+1} d_{n+1}^T (P^{(n)})^T}{1 + d_{n+1}^T P^{(n)} d_{n+1}} \\ \theta^{(n+1)} = \theta^{(n)} + P^{(n+1)} d_{n+1}\left(g^{(n+1)} - d_{n+1}^T \theta^{(n)}\right) \end{cases} \qquad (19)$$

where D is a matrix of dimensions $n \times m$ as specified in Equation (16), "$^{-1}$" denotes the inverse of a matrix, and

$$d_{n+1} = \begin{bmatrix} f_1\left(u^{(n+1)}\right) \\ f_2\left(u^{(n+1)}\right) \\ \ldots \\ f_2\left(u^{(n+1)}\right) \end{bmatrix}$$

4. Case Study

In this section we present a case study to demonstrate the benefits of using AT_RLSE$_c$. Data collected from four different experiments are used to compare fault detection effectiveness among AT_GA, AT_RLSE$_c$, and RT. Recall that AT_GA uses a genetic algorithm (GA) for parameter estimation, whereas AT_RLSE$_c$ uses an RLSE method.

4.1 Experiment Setup

We use the SPACE program received from the European Space Agency as the software being tested. It provides a language-oriented user interface that allows the user to describe the configuration of an array of antennas by using an array definition language (ADL) [1]. The program reads a file that contains several ADL statements, and checks the contents of the file for adherence to the ADL grammar and to specific consistency rules. If the ADL file is correct, SPACE outputs an array data file containing a list of array elements, positions, and excitations; otherwise, it outputs an error message. Its purpose is to prepare a data file in accordance with a predefined format and characteristics from a user, given the array antenna configuration described in a language-like form. The SPACE program consists of 9,564 lines of C code (6,218 executable).

Recall that one of the assumptions as discussed in Section 2 is that no debugging is performed for the component under test even if a failure is observed. Under this assumption, we should carefully select defects to avoid having too many failures. In other words, defects used in our experiments should not be those which can be easily detected by many test cases. A very important point worth noting is that these defects are *real* defects obtained from the error-log maintained during the development (in particular its testing and integration phases) of the SPACE program.

A test pool of 13466 test cases is generated using the test generator provided by the European Space Agency. This pool is then partitioned into four disjoint classes at random containing 3580, 5200, 2315, and 2371 test cases, respectively.

The maximum number of test cases (i.e., M) is set to 2000. We assume the cost for each action is the same, i.e., $w(i) = 5$ (refer to Equations (1), (5), and (7)). Different rebates (refer to Assumption 9 in Section 2) are used for different experiments. The initial failure detection rate (refer to Equation (11)) is set at 0.5 for every test class, i.e., $\theta_0(1) = \theta_0(2) = \theta_0(3) = \theta_0(4) = 0.5$. This is because we assume the probabilities of detecting a failure for each class are equally unknown.

4.2 Results and Observations

Four different experiments with different defects and rebates are conducted. To avoid introducing any possible bias in our statistical analysis, each experiment is repeated 100 times. Tables 1, 2, 3, and 4 give the average of the total cost defined by Equation (4) and the average number of failures observed.

For example, from column 2 of Table 1 the average total cost (with respect to 100 repetitions) for RT, AT_GA, and AT_RLSE$_c$ is 9682.7, 9668.2, and 9617.3, respectively. This implies AT_GA and AT_RLSE$_c$ have a 0.15% and 0.68% cost reduction, respectively, when compared with RT. From column 3 of Table 1, the average number of failures observed (with respect to 100 repetitions) for RT, AT_GA, and AT_RLSE$_c$ is 31.73, 33.18, and 38.27, respectively. This implies AT_GA and AT_RLSE$_c$ have a 4.57% and 20.61% increase in observed failures, respectively, when compared with RT. A similar observation can be made with respect to Tables 2, 3, and 4.

Table 1 Average total cost and number of failures observed for Experiment 1 with 5 defects and rebate = 10

	Average Total Cost	Average Number of Failures Observed
RT	9682.7	31.73
AT_GA	9668.2	33.18
AT_RLSE$_c$	9617.3	38.27
Improvement over Random Testing		
AT_GA	-0.15%	+ 4.57%
AT_RLSE$_c$	-0.68%	**+ 20.61%**

Table 2 Average total cost and number of failures observed for Experiment 2 with 5 defects and rebate = 100

	Average Total Cost	Average Number of Failures Observed
RT	6811	31.89
AT_GA	6763	32.37
AT_RLSE$_c$	6246	37.54
Improvement over Random Testing		
AT_GA	-0.70%	+ 1.51%
AT_RLSE$_c$	-8.30%	**+ 17.72%**

Table 3 Average total cost and number of failures observed for Experiment 3 with 2 defects and rebate = 10

	Average Total Cost	Average Number of Failures Observed
RT	9835.8	16.42
AT_GA	9823.8	17.62
AT_RLSE$_c$	9712.2	28.78
Improvement over Random Testing		
AT_GA	-0.12%	7.31%
AT_RLSE$_c$	-1.26%	**+ 75.27%**

Table 4 Average total cost and number of failures observed for Experiment 4 with 2 defects and rebate = 100

	Average Total Cost	Average Number of Failures Observed
RT	8280	17.20
AT_GA	8296	17.04
AT_RLSE$_c$	7114	28.86
Improvement over Random Testing		
AT_GA	-0.19%	-0.93%
AT_RLSE$_c$	-14.08%	**+ 67.79%**

For a better visualization, the corresponding pictorial display for the average total cost and the average number of failures observed is presented in Figure 2 and Figure 3, respectively.

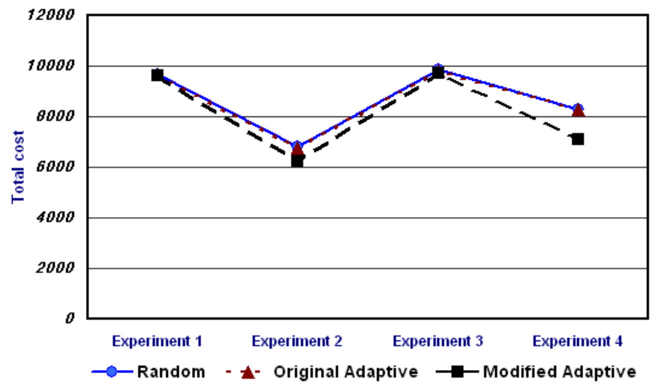

Figure 2 Average total cost for RT, AT_GA, and AT_RLSEc

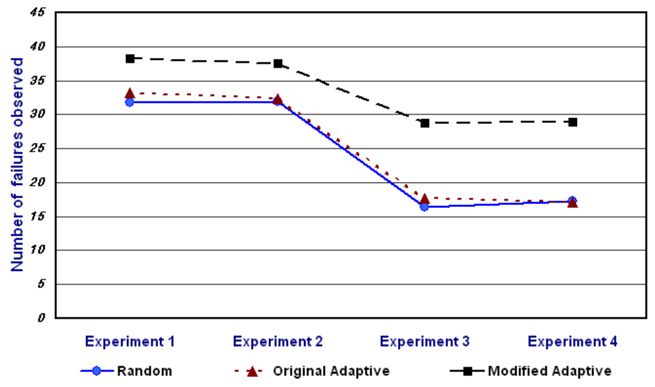

Figure 3 Average number of failures observed for RT, AT_GA, and AT_RLSEc

From the above tables and figures, we make the following observations:

1. AT_RLSE$_c$ (which uses an RLSE method for parameter estimation) significantly outperforms RT in all four experiments. Compared with RT, the average number of failures observed by AT_RLSE$_c$ is increased 20.61%, 17.72%, 75.27%, and 67.79%, respectively, for Experiment 1, 2, 3, and 4; the total cost is reduced by 0.68%, 8.30%, 1.26%, and 14.08%, respectively.
2. The AT_RLSE$_c$ also performs better than AT_GA (which uses a generic algorithm for parameter estimation) in all four experiments.
3. The higher the rebate, the more cost reduction AT_RLSE$_c$ achieves. However, the impact on cost reduction by AT_GA is very small.
4. The impact of rebate on the average number of failures observed is small for both AT_RLSE$_c$ and AT_GA.

5. Conclusions and Future Directions

In this paper, we use a modified adaptive testing strategy for software components. The two major differences between the modified (AT_RLSE$_c$) and the original adaptive testing (AT_GA) are (1) the controlled object in the modified approach represents an individual component, whereas in the original adaptive testing it represents the entire software being tested, and (2) the modified approach uses a recursive least squares estimation method for estimating parameters, whereas the original one uses a genetic algorithm.

From our data we observe that AT_RLSE$_c$ performs better than AT_GA and RT in terms of increasing the number of failures and reducing the total cost. These encouraging observations make it look promising for one to use the modified adaptive testing as described in this paper for testing software components.

Another point worth noting is that in our experiments the time required by AT_RLSE$_c$ to make a decision on how the underlying testing strategy should be adjusted is relatively small when compared to that required by AT_GA.

Finally, since the same cost is assigned for each test action and the same rebate for each failure observed in our experiments, an interesting future study would be to use different costs and rebates to further explore the benefits of using AT_RLSE$_c$ as described in this paper.

References

[1] A. Cancellieri and A. Giorgi, "Array Preprocessor User Manual," Technical Report IDS-RT94/052, 1994.
[2] C. Derman, *Finite State Markovian Decision Processes*, Academic Press, 1970.
[3] G. Rothermel, R. H. Untch, C. Chu and M. J. Harrold, "Prioritizing Test Case for Regression Testing," *IEEE Transaction on Software Engineering*, 27(10):929-948, October 2001.
[4] K. Y. Cai, Y. C. Li and K. Liu, "Optimal and Adaptive Testing for Software Reliability Assessment," *Information and Software Technology* (to appear).
[5] K. Y. Cai, "Optimal Software Testing and Adaptive Software Testing in the Context of Software Cybernetics," *Information and Software Technology*, 44(14):841-855, November 2002.
[6] K. Y. Cai, T. Y. Chen, Y. C. Li, W. Y. Ning and Y. T. Yu, "Adaptive Testing of Software Components," in *Proceedings of The 20th Annual ACM Symposium on Applied Computing* (SAC'05), pp. 1463-1469, Santa Fe, New Mexico, March 2005.
[7] K. Y. Cai, "Towards a Conceptual Framework of Software Run Reliability Modeling," *Information Sciences*, 126(1-4):137-163, July 2000.
[8] K. Y .Cai, Y. C. Li and W. Y. Ning, "Optimal Software Testing in the setting of Controlled Markov Chains," *European Journal of Operational Research*, 162(2):552-579, April 2005O. Hernandez-Lerma, *Adaptive Markov Control Processes*, Springer-Verlag, 1989.
[9] O. Hernandez-Lerma and J. B. Lasserre, *Discrete Time Markov Control Processes: Basic Optimality Criteria*, Springer, 1996.
[10] T. Y. Chen and Y. T. Yu, "On the Expected Number of Failures Detected by Subdomain Testing and Random Testing", *IEEE Transactions on Software Engineering*, 22(2):109-119, February 1996.