A Genetic Local Search Algorithm for Optimal Testing Resource Allocation in Module Software Systems

Ruimin Gao(✉) and Siyan Xiong

School of Computer Science and Technology,
University of Science and Technology of China (USTC), Hefei, China
gaorm@mail.ustc.edu.cn

Abstract. As modern software systems have been expanded continuously, the problem of how to optimally allocate the limited testing resource during the software testing phase attracted lots of attention. The Optimal Testing Resource Allocation Problems (OTRAPs) involve seeking for an optimal allocation of limited testing resource. There are two major objectives in the OTRAPs: reliability and cost. Since the designers pay more and more attention to reducing the cost, in this paper, we studied OTRAPs with the latter objective. In previous work, approaches based on genetic algorithms have been claimed to be strong alternatives in solving the problem. Hence, in this paper we proposed a new algorithm based on genetic algorithm and local search strategy (GLSA) to solve the OTRAPs. Experimental results show that the algorithm proposed can obtain better performance than some existing approaches for solving the software testing resource problem.

Keywords: Genetic algorithm · Software testing · Optimal allocation · Modular software system

1 Introduction

A software development process consists of four phases [1]: specification, designing, coding and testing. The testing phase consists of several stages including module testing, integration testing, system testing and acceptance testing [2]. During the testing phase the software system is tested to detect and correct faults. Researches show that the testing phase is the most costly phase and approximately 40–50 % of the resources consumed during the software development are testing resources [3]. Each software module is tested independently during the module phase and not all the modules are equally important, therefore how to allocate the limited resource is an important issue.

There are numerous publications on OTRAPs [1–19] which are proposing new formulations of OTRAPs or utilizing problem-solving approaches. To formulate an OTRAP, the relationship of reliability, cost and resources should be defined firstly. The relationship between reliability is described by Reliability Growth Models (SRGMs). The reliability can be described by failure severity, the number of failures, time occurrence and so on [4]. Researchers use different theories, like non-homogeneous

D.-S. Huang et al. (Eds.): ICIC 2015, Part II, LNCS 9226, pp. 13–23, 2015.
DOI: 10.1007/978-3-319-22186-1_2

Poisson process, chaos theory and uncertainty theory, to build a SRGM. Therefore there are publications of maximizing the reliability with constrained testing resource to ensure that the reliability of software systems will be maximized [5–8] or the remaining faults can be minimized [1, 3]. In practical applications, cost means profit. Hence in recent years, researchers pay more attention on the testing cost by minimizing the testing cost with constrained testing resource and reliability [9, 10]. In practical problems designers want to minimum the cost under conditions of achieving a certain level of software reliability. Therefore, in this paper, we studied OTRAPs with the goal to minimize the testing cost with the constraint of reliability and testing resource.

There many optimization approaches have been proposed to solve OTRAPs, like dynamic programming [11], and nonlinear programming [12–14]. When the solution space of the problem is multimodal, these methods may be easily trapped in a local optimal. Genetic algorithms (GA) are inspired by Darwin's theory of evolution. GA has a strong capability of global search which will escape from local optima. In previous work, the best way to solve OTRAPs of the goal to minimize the testing cost with the constraint of reliability and testing resource is GA [9, 10, 19]. To accelerate the genetic algorithm in converging on optimal solutions, we add a local search operator in GA.

The rest of the paper is organized as follows. In Sect. 2, the problem formulation is given. After that, Sect. 3 introduces the genetic local search algorithm in detail. In Sect. 4, empirical studies and discussions are presented to compare the proposed algorithm with other approaches. Finally, in Sect. 5, conclusions are presented.

2 Problem Formulations

Because a software system typically consists of multiple modules, the model that formulates the relationship between the testing resource and reliability (or cost) needs to be addressed. We address this problem with SRGMs, and the SRGMs defined based on the following notations and assumptions [9, 10, 15].

Notations:

W: Total testing resource
C: Total testing cost
$W(t)$: Cumulative testing effort in the time interval $(0, t]$.
$\omega(t)$: Current testing-effort expenditure rate at time t.
$m(t)$: Mean value function in NHPP (Non-Homogeneous Poisson Process).
p: Probability of perfect debugging of a fault.
a: The number of dormant faults in the software at the beginning of testing.
b: Fault detection rate.
α: Error generation rate.
β: Constant in logistic learning function.

Assumptions

(1) The software failure phenomenon can be described by a NHPP.
(2) The software is subject to failures at random times during execution caused by faults remaining in the software.

(3) Each time a failure is observed, an immediate effort takes place to decide the cause of the failure in order to remove it.
(4) During the fault removal process, the fault is reduced by one with probability p or remains unchanged with probability 1-p.
(5) During the fault removal process, new faults can be generated. The fault generation rate is proportional to the rate of fault removal.
(6) Fault removal rate per remaining fault assumed to be non-decreasing inflection S-shaped logistic function.

Under the above assumptions, the removal process can be described with testing resource as follows [10]:

$$\frac{dm(t)}{dt} = pb(W(t))(a(t) - m(t))\frac{dW(t)}{dt} \tag{1}$$

Where

$$b(W(t)) = \frac{b}{1 + \beta e^{-bW(t)}} \tag{2}$$

$$a(t) = (a + \alpha m(t)) \tag{3}$$

Solving Eq. (1) in under the condition $m(t = 0) = 0$ and $W(t = 0) = 0$ we can get:

$$m(t) = \frac{a}{1 - \alpha}\left[1 - \left(\frac{(1 + \beta)e^{-bW(t)}}{1 + \beta e^{-bW(t)}}\right)^{p(1-\alpha)}\right] \tag{4}$$

Based on SRGMs, the mean value function of fault removal process for i-th module can be calculated as:

$$m_i(t) = \frac{a_i}{1 - \alpha_i}\left[1 - \left(\frac{(1 + \beta_i)e^{-b_i W_i(t)}}{1 + \beta_i e^{-b_i W_i(t)}}\right)^{p_i(1-\alpha_i)}\right] i = 1, 2, \ldots\ldots N \tag{5}$$

Where N is the number of modules in a software system.

W_i is cumulative testing resource in the interval $(0, t]$ for i-th module. Based on the above equation, the mean value function of a software system with N modules is calculated as:

$$m(W) = \sum_{i=1}^{N} v_i \frac{a_i}{1 - \alpha_i}\left[1 - \left(\frac{(1 + \beta_i)e^{-b_i W_i}}{1 + \beta_i e^{-b_i W_i}}\right)^{p_i(1-\alpha_i)}\right] i = 1, 2, \ldots\ldots N \tag{6}$$

Where v_i is the weight attached to i-th module.

The measure of defining software reliability at time t as given by Huang et al. [16] is, "the ratio of the cumulative number of detected faults at time t to the expected

number of initial fault content of the software." The initial fault content of the proposed model is $\frac{a}{1-\alpha}$. Then the reliability of i-th module is calculated as:

$$R_i = \frac{m_i(t)}{\frac{a_i}{1-\alpha_i}} = 1 - \left(\frac{(1+\beta_i)e^{-b_i W_i}}{1+\beta_i e^{-b_i W_i}}\right)^{p_i(1-\alpha_i)} \quad i = 1, 2. \ldots . . N \tag{7}$$

Most commonly cost model for each module is defined as [20]:

$$C_i(T) = C_{1i} m_i(T) + C_{2i}(m_i(\infty) - m_i(T)) + C_{3i}(T) \; i = 1, 2. \ldots . . N \tag{8}$$

Where the first and second term is the cost of testing and debugging during the testing phase and operational phase respectively and the last one is the cost of testing up to the release time.

For SRGMs with testing resource the above cost function can be modified to describe the costs per unit testing effort expenditure as:

$$C_i(W(T)) = C_{1i} m_i(W(T)) + C_{2i}(m_i(\infty) - m_i(W(T))) + C_{3i}(W(T)) \; i = 1, 2. \ldots . . N \tag{9}$$

The total cost of the whole system can be calculated as:

$$C = \sum_{i=1}^{N} \left\{ (C_{1i} - C_{2i}) \frac{v_i a_i}{1-\alpha_i} \left[1 - \left(\frac{(1+\beta_i)e^{-b_i W_i}}{1+\beta_i e^{-b_i W_i}} \right)^{p_i(1-\alpha_i)} \right] + \frac{C_{2i} a_i}{1-\alpha_i} + C_3 W_i \right\} \tag{10}$$

Given an optimal testing-resource allocation problem, we aimed at minimizing the testing cost when the reliability of each module is at least R_0. Then the optimal testing-resource allocation problem can be formulated as:

Minimize

$$C = \sum_{i=1}^{N} \left\{ (C_{1i} - C_{2i}) \frac{v_i a_i}{1-\alpha_i} \left[1 - \left(\frac{(1+\beta_i)e^{-b_i W_i}}{1+\beta_i e^{-b_i W_i}} \right)^{p_i(1-\alpha_i)} \right] + \frac{C_{2i} a_i}{1-\alpha_i} + C_3 W_i \right\}$$

Subject to

$$\sum_{i=1}^{N} W_i \leq W$$

$$W_i \geq 0, i = 1, \ldots . . , N$$

$$R_i \geq R_0, i = 1, \ldots . . , N \tag{11}$$

3 GLSA for Testing-Resource Allocation

GA is inspired by Darwin's theory of evolution. Godberg [21] gave the introduction of GA. To accelerate the genetic algorithm in converging on optimal solutions, we add a local search operator in GA. The pseudo-code of GLSA is presented in Algorithm 1.

Algorithm 1. Pseudo-Code of GLSA

```
Randomly generate an initial population with N individuals P = {x_1, x_2 ..... x_N}
The generation counter t=0
while t < t_max  (maximum generation number)
    Evaluate the fitness of all the individuals in the population, the fitness of
    individual x_i is denoted as f(x_i)
      Select parent individuals to do the crossover operation
      if f(offspring)<f(parent)
            Replace parent by offspring
      end if
      for i=1:N
            Perform mutation on individual x_i with probability P_1 to generate
      an offspring x_i^r
                if  f(x_i^r)<f(x_i)
                    x_i=x_i^r
                end if
            Perform local search around each individual with probability P_2 to
      generate an offspring x_i^rr
                if  f(x_i^rr)<f(x_i)
                    x_i=x_i^rr
                end if
          end for
end while
```

3.1 Fitness Function

How to evaluate the quality of a solution is an important issue in a GA. In our testing resource allocation problem, the fitness function is given by:

$$Fitness = \sum_{i=1}^{N} \left\{ (C_{1i} - C_{2i}) \frac{v_i a_i}{1-\alpha_i} \left[1 - \left(\frac{(1+\beta_i)e^{-b_i W_i}}{1+\beta_i e^{-b_i W_i}} \right)^{p_i(1-\alpha_i)} \right] + \frac{C_{2i} a_i}{1-\alpha_i} + C_3 W_i \right\} \quad (12)$$

3.2 The Selection, Crossover and Mutation Operator

Population diversity and selective pressure are two important issues in the evolution process of a genetic search. Strong selective pressure may lead to early convergence to a local optima. If the selective pressure is weak and the population is too diverse, the search will be ineffective. Hence we use tournament selection here.

In the crossover process, two chromosomes are formed by swapping sets of genes of two parent chromosomes, hoping that at least one child will have genes that improve its fitness. In the proposed algorithm, we use arithmetical crossover operator.

In the mutation process, the algorithm may escape from a local optima. We use Gaussian mutation here.

3.3 Local Search Strategy

GA possesses the strong capability of global search and usually not very sensitive to initial solutions. The effectiveness of GA has been demonstrated on OTRAPs, but GA also has shortcomings like poor capability of local search. To accelerate the genetic algorithm in converging on optimal solutions, we combine GA and local search operator to solve OTRAPs.

In this paper, the local search operator is applied to the selected individual. Assume the selected individual **x** corresponds to a system consisting of N components ($N \geq 2$) and the fitness of individual **x** is $f(\mathbf{x})$. The pseudo-code of the local search is presented in Algorithm 2.

Algorithm 2. Pseudo-Code of the local search procedure on an individual **x**

1: Randomly select two components of **x**, denoted as x_1, x_2
2: Initialize an archive $S = \emptyset$. This archive will be used to store the feasible solutions obtained by local search
3: Decrease the resource of the two components by $m * \min(x_1, x_2)$,(ie, $x_1' = x_1 - m * \min(x_1, x_2) and\ x_2' = x_2 - m * \min(x_1, x_2)$) to get a new individual $\mathbf{x}_{i1}$
4: Increase the resource of the two components to get $\mathbf{x}_{i2}$
5: Increase the resource of one component and Decrease the resource of the other component at the same time to get two new individuals $\mathbf{x}_{i3}$ and $\mathbf{x}_{i4}$
6: Decrease the resource of one component to get two new individual $\mathbf{x}_{i5}$ and $\mathbf{x}_{i6}$
7: Decrease the resource of one component to get two new individual $\mathbf{x}_{i7}$ and $\mathbf{x}_{i8}$
8: Include all the new individuals into S
9: i=1;
10: while i<9
11: Randomly select an individual $\mathbf{x}'$ from S
12: if $\mathbf{x}'$ violate all the constraints and f ($\mathbf{x}'$)<f(**x**)
13: **x**=$\mathbf{x}'$
14: break
15: else
16: i=i+1
17: if i=9
18: get the one which has the smallest fitness among the new individuals and parent individual to replace **x**
19: end if
20: end while

The local search operator randomly selects two components. The resources of the two components are modified to generate 8 new individuals. A newly generated individual is reserved in an archive S. Randomly select an individual from S. If the new individual violate all the constraints and the fitness of the new individual is less than $f(\mathbf{x})$, the new individual will replace $\mathbf{x}$. Otherwise, if all the individuals in S do not violate the constraints, get the one which has the smallest fitness among the new individuals and parent individual to replace $\mathbf{x}$.

4 Numerical Example

In this section, the performance of GLSA is evaluated on a numerical example. The simulation is carried out in MATLAB R2014b software under a 64 bit windows 8.1 environment. The processor of my PC is core(TM) i5-4200 M with a CPU frequency at 2.5 GHZ, as well as 8 GB memory.

The results are compared with two algorithms. The first one is Simulated Annealing (SA). The other algorithm, denoted as K-A algorithm, was proposed by Kapur and Aggarwal [10]. K-A algorithm used GA to solve OTRAPs. For K-A, the population size is 50, the crossover probability is 0.8 and the mutation probability is 0.05. For GLSA, the population size is 50, the crossover probability is 0.8, the mutation probability is 0.05 and the local search probability is 0.05. For SA, the initial temperature is 10000 and the temperature update factor is 0.975.

Consider a software system consisting of six modules proposed in [10]. The parameters estimate for each module is shown in Table 1. The total testing resource is 50000. The cost parameters $C_{1i} = C_1 = 2, C_{2i} = C_2 = 10$ and $C_3 = 0.5$.

Table 1. Parameters used for OTRAPs

Module	a_i	b_i	α_i	β_i	υ_i	p_i
M1	218	0.00147	0.0344	0.9736	0.1543909	0.95
M2	105	0.00955	0.6513	0.5321	0.0743626	0.91
M3	231	0.00174	0.2959	0.4129	0.1635977	0.9
M4	286	0.00252	0.1148	0.7276	0.2025496	0.92
M5	260	0.00429	0.1433	0.3697	0.184136	0.91
M6	312	0.00267	0.0123	0.6321	0.2209632	0.99

4.1 The Convergence Behavior Study

In this experiment, the aim is to evaluate the convergence behavior of GLSA, K-A and SA on a case study. The reliability value is set to 0.93. Each algorithm runs 15 times and the result in the table is the average of 15 times. The last row provides results of the Wilcoxon Test, while "w-d-l" indicates GLSA is superior, not significantly different or inferior to the corresponding compared algorithms (Table 2).

The convergence curve of SA, K-A and GLSA are plotted in Fig. 1, where the x-axis represents the number of generations and the y-axis represents the cost. As we

Table 2. Average results of SA, K-A and GLSA with differerent generations

Generation	SA	K-A	GLSA
100	38033.51	35764.42	32765.23
200	37565.20	33604.63	28446.26
300	37270.25	30217.77	25949.91
400	36867.54	28321.97	23937.33
500	35825.89	27293.51	23131.70
600	35227.39	26277.05	22500.39
700	34750.99	25088.98	22494.36
800	33257.69	25009.90	21454.55
900	32904.24	23873.14	21194.72
1000	33034.07	23043.93	21277.61
2000	29648.00	21392.55	20614.45
3000	26162.72	21005.52	20686.10
4000	24757.10	21095.74	20532.21
5000	24317.54	21010.47	20529.96
6000	24100.22	20871.98	20528.93
7000	23246.10	21214.30	20527.14
8000	22630.71	20869.81	20527.81
9000	22162.83	20919.29	20525.88
10000	21959.27	20898.93	20525.88
Wilcoxon Test	18-1-0	19-0-0	–

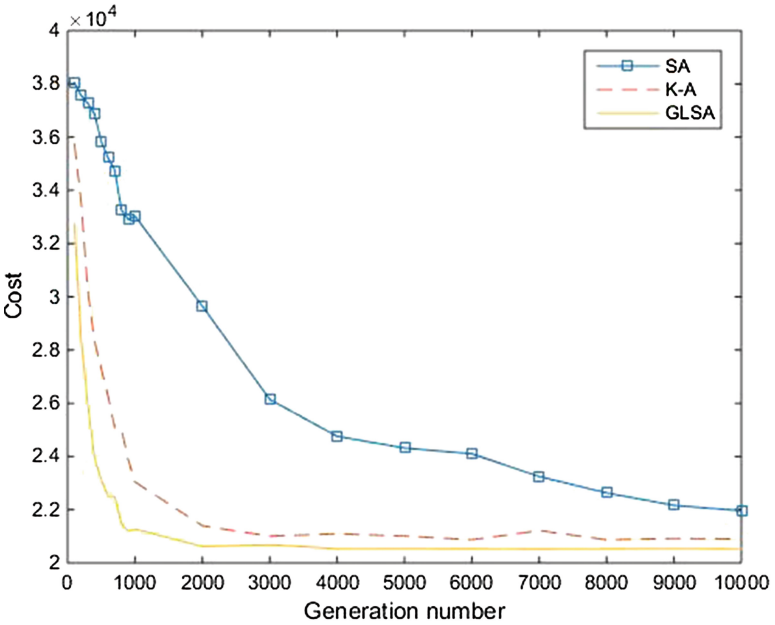

Fig. 1. Convergence Plot of SA, K-A and GLSA

can see, all the methods converged rather fast and the solution obtained by GLSA was significantly better than other two algorithms. Furthermore, the Wilcoxon test (with a significance level 0.05) has been employed to compare the overall performance of the three methods. For most generation numbers, the results of GLSA are significantly than others.

Assume that For K-A, the population size is P, the crossover probability is p_c and the mutation probability is p_m. For GLSA, the population size is P', the crossover probability is p'_c, the mutation probability is p'_m and the local search probability is p_l. For SA, the initial circulation number is u. The generation number of the three algorithms is t_{max}. Therefore, we can calculate that the number of fitness evaluations of K-A is $P + (p_c \times P + p_m \times P) \times t_{max}$, the number of fitness evaluations of GLSA is $P' + (p'_c \times P' + p'_m \times P' + p_l \times 8 \times P') \times t_{max}$, and the number of fitness evaluations of SA is $1 + u \times t_{max}$. In this experiment, the number of fitness evaluations of K-A is $50 + 43\ t_{max}$, the number of fitness evaluations of GLSA is $50 + 63\ t_{max}$ and the number of fitness evaluations of SA is $1 + 200\ t_{max}$.

4.2 Performance Over Different Reliability Values

The experiment above-mentioned only provides a case study on which the GLSA outperformed the other two algorithms, but it is not sufficient to draw a conclusion. Therefore, we take another experiment to compare the algorithms by varying the reliability constraint. Each algorithm runs 15 times and the result in the table is the average of 15 times (Table 3).

Table 3. Average results of SA, K-A and GLSA with differerent reliability constraint values

Reliability constraint value	SA	K-A	GLSA
0.81	19157.31	19290.78	19176.89
0.84	19919.86	19531.35	19397.61
0.87	20144.25	19738.36	19662.84
0.9	20580.31	20075.07	20022.95
0.93	20877.67	20711.46	20526.40
0.96	21564.95	21456.13	21353.93
Wilcoxon Test	4-2-0	5-1-0	–

The results are plotted in Fig. 2, where the x-axis represents reliability constraint value and the y-axis represents the cost. As we can see, the solution obtained by GLSA was significantly better than other two algorithms. Furthermore, the Wilcoxon test (with a significance level 0.05) has been employed to compare the overall performance of the three methods. For most reliability constraint value, the results of GLSA are significantly than others.

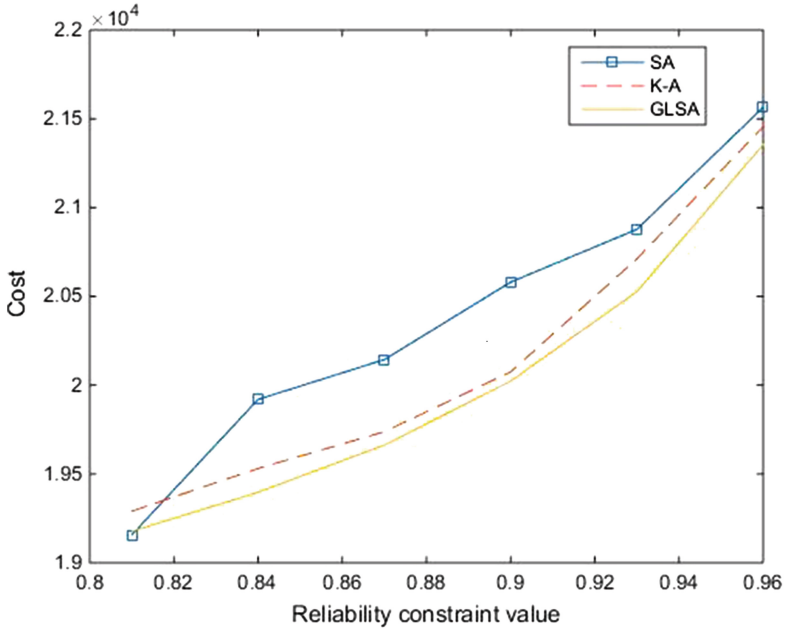

Fig. 2. Comparison among SA, K-A and GLSA under different reliability constraint values

5 Conclusion

In this paper, we proposed a genetic local search algorithm to solve the optimal testing resource problem by minimizing the total software testing cost with the constraints of limited testing resource and to achieve the desired level of reliability for each module. Through the experiments of a numerical example, we can draw a conclusion that the GLSA converged significantly better than the compared algorithms and it can find a better solution.

References

1. Ohtera, H., Yamada, S.: Optimal allocation and control problems for software-testing resources. IEEE Trans. Reliab. **39**, 171–176 (1990)
2. Dai, Y.-S., Xie, M., Poh, K.-L., Yang, B.: Optimal testing-resource allocation with genetic algorithm for modular software systems. J. Syst. Softw. **66**, 47–55 (2003)
3. Yamada, S., Ichimori, T., Nishiwaki, M.: Optimal allocation policies for testing-resource based on a software reliability growth model. Math. Comput. Model. **22**, 295–301 (1995)
4. Lyu, M.R., Rangarajan, S., Van Moorsel, A.P.: Optimal allocation of test resources for software reliability growth modeling in software development. IEEE Trans. Reliab. **51**, 183–192 (2002)
5. Coit, D.W.: Economic allocation of test times for subsystem-level reliability growth testing. IIE Trans. **30**, 1143–1151 (1998)

6. Coit, D.W., Smith, A.E.: Reliability optimization of series-parallel systems using a genetic algorithm. IEEE Trans. Reliab. **45**, 254–260, 266 (1996)
7. Yang, B., Xie, M.: Testing-resource allocation for redundant software systems. In: 1999 Pacific Rim International Symposium on Dependable Computing, Proceedings, pp. 78–83, IEEE (1999)
8. Yang, B., Xie, M.: A study of operational and testing reliability in software reliability analysis. Reliab. Eng. Syst. Saf. **70**, 323–329 (2000)
9. Jha, P., Gupta, D., Yang, B., Kapur, P.: Optimal testing resource allocation during module testing considering cost, testing effort and reliability. Comput. Ind. Eng. **57**, 1122–1130 (2009)
10. Kapur, P., Aggarwal, A.G., Kapoor, K., Kaur, G.: Optimal testing resource allocation for modular software considering cost, testing effort and reliability using genetic algorithm. Int. J. Reliab. Qual. Saf. Eng. **16**, 495–508 (2009)
11. Leung, Y.-W.: Dynamic resource-allocation for software-module testing. J. Syst. Softw. **37**, 129–139 (1997)
12. Huang, C.-Y., Lo, J.-H.: Optimal resource allocation for cost and reliability of modular software systems in the testing phase. J. Syst. Softw. **79**, 653–664 (2006)
13. Hou, R.-H., Kuo, S.-Y., Chang, Y.-P.: Efficient allocation of testing resources for software module testing based on the hyper-geometric distribution software reliability growth model. In: Seventh International Symposium on Software Reliability Engineering, 1996 Proceedings, pp. 289–298, IEEE (1996)
14. Berman, O., Ashrafi, N.: Optimization models for reliability of modular software systems. IEEE Trans. Softw. Eng. **19**, 1119–1123 (1993)
15. Ohba, M.: Software reliability analysis models. IBM J. Res. Dev. **28**, 428–443 (1984)
16. Huang, C.-Y., Lo, J.-H., Kuo, S.-Y., Lyu, M.R.: Optimal allocation of testing-resource considering cost, reliability, and testing-effort. In: 10th IEEE Pacific Rim International Symposium on Dependable Computing, 2004 Proceedings, pp. 103–112, IEEE (2004)
17. Espadas, J., Molina, A., Jiménez, G., Molina, M., Ramírez, R., Concha, D.: A tenant-based resource allocation model for scaling software-as-a-service applications over cloud computing infrastructures. Futur. Gener. Comput. Syst. **29**, 273–286 (2013)
18. Kang, D., Jung, J., Bae, D.H.: Constraint-based human resource allocation in software projects. Softw. Pract. Exp. **41**, 551–577 (2011)
19. Johri, P., Nasar, M., Chanda, U.: A genetic algorithm approach for optimal allocation of software testing effort. Int. J. Comput. Appl. **68**, 21–25 (2013)
20. Kapur, P., Garg, R., Kumar, S.: Contributions to Hardwave and Software Reliability. World Scientific, Singapore (1999)
21. Golberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addion wesley, Boston (1989)

http://www.springer.com/978-3-319-22185-4

Intelligent Computing Theories and Methodologies
11th International Conference, ICIC 2015, Fuzhou, China, August 20-23, 2015, Proceedings, Part II
Huang, D.-S.; Jo, K.-H.; Hussain, A. (Eds.)
2015, XXIX, 755 p. 250 illus., Softcover
ISBN: 978-3-319-22185-4