Evolutionary Algorithm with Convergence Speed Controller for Automated Software Test Data Generation Problem

Fangqing Liu
School of South China
University of Techology
Software engineering
Guangzhou 510006,China
Email: 564376030@qq.com

Han Huang✉
School of South China
University of Techology
Software engineering
Guangzhou 510006,China
Email: hhan@scut.edu.cn

Zhifeng Hao
School of Mathematics
and Big Data,
Foshan University,
Foshan, P. O. Box, China
Email: zfhao@fosu.edu.cn

***Abstract*—Software testing is an important process of software development. One of the challenges in testing software is to generate test cases which help to reveal errors. Automated software test data generation problem is hard because it needs to search the whole feasible area to find test cases covering all possible paths under acceptable time consumption. In this paper, evolutionary algorithm with convergence speed controller (EA-CSC) is presented for using the least test case overhead in solving automated test case generation problem. EA-CSC is designed as a framework which have fast convergence speed and capability to jump out of the local optimal solution over a range of problems. There are two critical steps in EA-CSC. The adaptive step size searching method accelerates the convergence speed of EA. The mutation operator can disrupt the population distribution and slows down the convergence process of EA. Moreover, the EA-CSC results are compared to the algorithms tested on the same benchmark problems, showing strong competitive.**

***Index Terms*—Evolutionary Algorithm(EA), Convergence Speed Controller(CSC), Test Data Generation**

I. Introduction

It is widely recognized that software testing plays an important role in successful software development process. However, software testing is a kind of labor-intensive work which is not efficient. Almost 50% overhead in software development will use in testing [1]. Bugs exist in each software more or less. We have finite resources in software development lifecycle [2]. One of the challenges to testing software in the effort involved creating test cases which systematically test the whole software and reveal errors. Thus, a reasonable solution is to automate the testing process as much as possible. Test cases generation is naturally a key part of this process. Automated test data generation helps researchers to find more faults and errors.

Software testing can be divided into two kinds: one is black-box testing (namely functional testing), the other is white-box or structure-oriented testing. The criterion of black-box testing is verifying whether the software can respond correctly. White-box testing is based on the fact that the source code of software is known exactly. White-box testing is considered to be an effective approach and have widely application in automated test data generation problem [3]. Joseph et al. [4] have introduce that coverage is a kind of effective way to achieve software quality in 1990s. There are many criteria for structure-oriented testing such as statement coverage, branch coverage and path coverage [1]. The most effective criterion is path coverage. However, this criterion is the most difficult to be satisfied.

A useful strategy which gives great attribution to this area is evolutionary algorithms [5]. Researchers usually use evolutionary algorithms to solve this problem because test case generation problems could be modeled as optimization or search problems. And evolutionary algorithm is a mature global optimization method with high robustness for solving this kind of complex problems. Some researchers considered that we could use the simplest strategies such as random or hill climbing (HC) [6] for automated test data generation problem. However, automated test data generation problem is usually a non-convex problem. Random strategy is full of uncertainty. HC doesn't preform well in complex non-convex problem.

Two kinds of evaluating criteria were proposed for automated test data generation problems [7]. One is based on branch distance between the current position and the target path, the other is using the similarity between two test cases. Nigel Tracy [8] proposed the branch distance terms. The fitness value of input data was evaluated by comparing the difference between actual value and target position in each judgement branch. Many researchers used this criterion to generate test data, such as [2], [9], [10] and [11]. The other criterion is based on the control flow graphs of comparing tested problems. A number of scientists used similarity criteria to generate test data. For example, Yao [12] used GA to generate test data based on path coverage using similarity criterion.

In this paper, we designed an evolutionary algorithm (EA) with convergence speed controller (CSC) [13] to deal with automated test data generation problem. This algorithm follows the basic evolutionary algorithm framework. The propagation of the current population is based on the heuristic information which is provided by current population. However, we find

978-1-5090-4601-0/17/$31.00

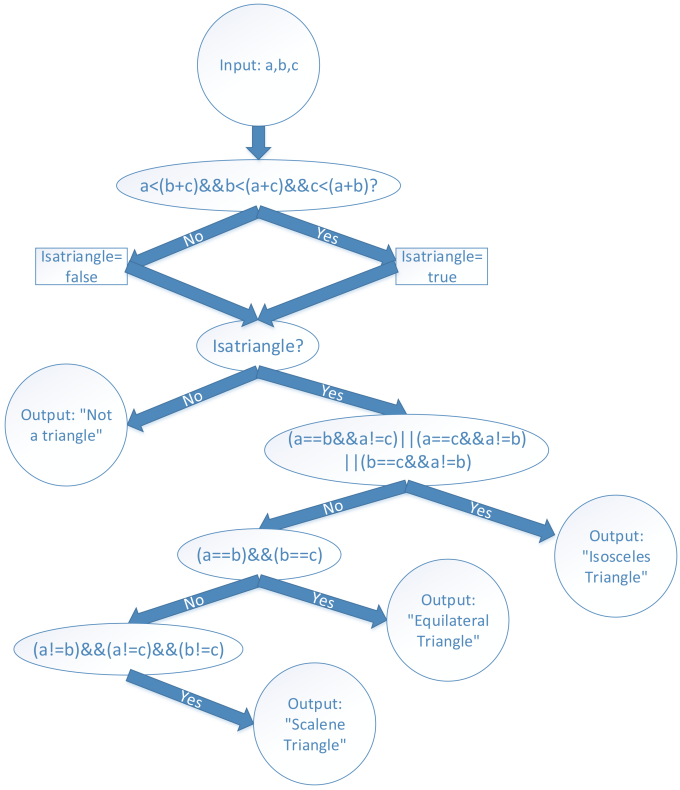


Fig. 1: The process of "Triangle" Problem

EA don't perform well on tested problems. Because EA iterate many times without converging in a small scale. EA will still search in a very large space, and it's very hard to find test cases covering the uncovered paths. Therefore, we applied convergence speed controller which used the adaptive searching length to search the solution for overcome the shortcoming of slow converging speed.

II. Automated Software Test Data Generation Problem

In the section, we will introduce the basic conception and definition about automated software test data generation problem. The test case evaluation function is also included in this part.

A. Problem Definition

We assume that the test problem ϕ contains k paths which can be defined as $\bar{P} = (P_1, P_2, ..., P_k)$. The test problem also contains m input values which can be defined as $x_1, x_2, ..., x_m$. The input domain of those values are $\boldsymbol{S_1, S_2, ..., S_m}$. We can define a test case as a combination about a set of input domain: $X = (x_1, x_2, ..., x_m)$ where $(x_i \in \boldsymbol{S_i}\ , i = 1, 2, ..., m)$. Each of test case X will cover one and only one path. Our purpose is to get a test suit $\bar{X}$ which contains k test cases. Every test case of test suit $\bar{X} = (X_1, X_2, ..., X_k)$ covers different paths at the same time.

As we can see from Fig.1, "Triangle" tested problem has four paths which correspondent to four outputs. This tested problem has three input values. Each group of those (a,b,c) is a test case to this tested problem. Our goal is to find four test cases which cover these four paths. If the input values a, b, c are both in the range of (lb, ub). The search space of this tested problem is $(ub - lb)^3$. When (lb, ub) use the maximum positive integer space, the search space is about 10^{27}. It is very difficult to find a test case covered the target path in such a large search space.

Automated test data generation problem based on path coverage is a kind of combination problem. Each set of input value is a combination of those domain. Those sets of input value are on behave of an problem path. The purpose of this problem is to find at least one of those combinations of input values for all paths.

B. Evaluation Function

Evaluation function is one of the critical part in using evolutionary algorithm to generate test case for path covering. Evaluation function indicates the distinction between current position and the target position. It plays an important role in evolutionary algorithm. In order to cover all possible paths, we need to use the heuristic information to update the current population. In this section we proposed a kind of evaluation function based on branch distance criterion. The proposed fitness function is designed to adaptively update the position of current population according to the found solutions. The equation of fitness function is presented as follows:

$$fitness(X) = \sum_{i=1}^{length(P_X)} f(P_X, i) \tag{1}$$

where $length(P_X)$ is the number of judgement vertexes on path P_X, and $f(P_X, i)$ is part of the fitness value of test case X in its i-th vertex. After accumulating all the fitness values of the judgement vertexes, we obtained the fitness value of the path P_X. The equation of $f(P_X, i)$ is presented in the following paragraph:

$$f(P_X, i) = \frac{1}{cv(v_X^i) + \varepsilon} \tag{2}$$

where v_X^i is the i-th judgement vertex on P_X, and ε is a positive constant value avoiding denominator becoming zero. $cv(v_X^i)$ is the cost value which is described in TABLE I [14], [11], [10].

III. Evolutionary Algorithm With Convergence Speed Controller

Evolutionary Algorithm has an extensive use for searching problem and optimization problem. Numerous researchers are attracted to study in this field. Considering the accuracy and overhead of the problem, we choose positive integer as the input value domain. In the following section, we will introduce the process of our proposed evolutionary algorithm and present the pseudo-code of EA-CSC.

Each individual in the population is on behalf of a test case. The position of individual is a set of input value for tested problems. As we can see from Fig 2, a new test case

Algorithm 1: EA-CSC for automated test data generation problem

Input : a test problem ϕ
Output: a test suite $\bar{X}$ covering all paths in problem ϕ

```
//Step 1: Initialization
Let mutation radio be Mut = 0.1, let the basic step length be step = (ub − lb)/10
for each individual X_i iϵ(1, 2, ..., Pop) do
    Randomly initialize the position of the population
    Set each path of φ to be uncovered
    Add X_i to X̄ if an uncovered path is covered by X_i
end
//End of Step 1
while termination criterion is not met do
    //Step 2: Mutation operation
    for each individual X_i iϵ(1, 2, ..., Pop) do
        for each dimension jϵ{1, 2, ..., n} do
            Generate a random float number r in [0,1]
            if r < Mut then
                Randomly regenerate the x_j of X_i by evenly distributing in the range of [lb, ub]
            end
        end
        Add X_i to X̄ if an uncovered path is covered by X_i
    end
    //End of Step 2
    //Step 3: Adaptive step searching
    for each individual X_i iϵ(1, 2, ..., Pop) do
        Random select a dimension jϵ{1, 2, ..., n} from X_i
        while step ≥ 1 do
            Set x_j as the median value, search 10 values based on the step length step
            Find the point with the best evaluation value, update x_j to this value
            step = step/10
        end
        Add X_i to X̄ if an uncovered path is covered by X_i
    end
    //End of Step 3
end
```

* up and lb are the upper bound and lower bound of input value's domain. * The terminal criterion of this algorithm is that all the paths of ϕ are covered or the test case overhead exceed preset value 10^8.

TABLE I: Criteria of cost value

Predicate of v	cost value
Boolean	if Ture, $cv = 0$ else $cv = K$
$a > b$	if $(b - a) < 0$, $cv = 0$ else $cv = (b - a) + K$
$a \geq b$	if $(b - a) \leq 0$, $cv = 0$ else $cv = (b - a) + K$
$a < b$	if $(a - b) < 0$, $cv = 0$ else $cv = (a - b) + K$
$a \leq b$	if $(a - b) \leq 0$, $cv = 0$ else $cv = (a - b) + K$
$a = b$	if $abc(a - b) = 0$, $cv = 0$ else $cv = (a - b) + K$
$a \neq b$	if $abs(a - b) \neq 0$, $cv = 0$ else $cv = K$
$a \wedge b$	$cv = cost(a) + cost(b)$
$a \vee b$	$cv = min(cost(a), cost(b))$

* K is a constant value.
* a and b are variables that appears in the tested problem's branch node. They may be a known value or a complex value calculated by the preceding part of the program.

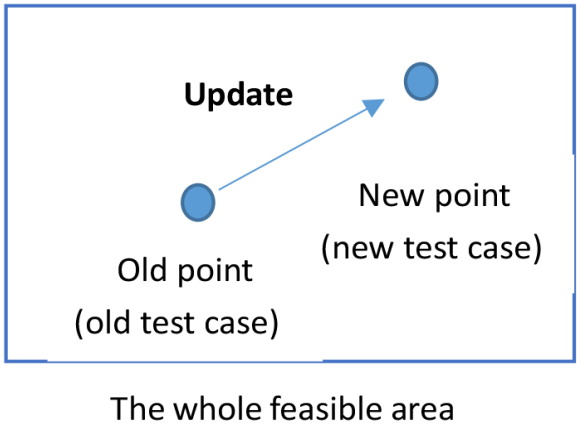


Fig. 2: The process of new test case generation

is generated when EA-CSC update the individual's position. Each individual's position can generate an path. Our purpose is to get at least one set of test cases covering all the paths.

EA-CSC contains three basic steps: Step 1 is initializing the population. Step 2 is the basic evolutionary algorithm's operator named mutation which is commonly used in many evolutionary algorithms [9], [15]. Step 3 is the adaptive step searching operation which we employ to accelerate convergence speed of EA.

Pop is the size of the population which is initialized to be 50. Mut is the probability of mutation in evolutionary algorithm. In order to maintain the fairness of the comparison algorithms, we initialize the population randomly in Step 1. Each individual's position will be randomly set in their range of values. After initializing the first generation population, we will record the new covered path and its test case. There are two basic operators in evolutionary algorithm: crossover and mutation. Both of crossover and mutation are used to searching the feasible region. However, the mutation operator only modify a certain dimension of an individual while the operator of crossover changes the multiple attributes of the individuals. In order to ensure that EA can converge, only the mutation operator is chosen.

In Step 2 of the pseudo-code, EA-CSC will iterate through all the dimensions of each individual and mutate the value

under the probability of Mut. We design this step to slow down the convergence speed.

Step 3 is the main operation of our proposed convergence speed controller: A dimension j will be randomly select for each individual X_i. Line 24 and 25 indicate that EA-CSC will search the dimension j of this individual X_i. EA-CSC will search 10 values in the searching range by step length $step$ in each generation. After finding the value with the best evaluation value between those 10 values, x_j will be updated to this value. The initializing step length $step$ can be calculated by following equation:

$$step = (up - lb)/10 \tag{3}$$

where up and lb are the upper bound and lower bound of input value's domain. The $step$ will be updated as showed in line 26 of Algorithm 1 after a round of searching. The "Adaptive step searching" method will quickly help the individuals search to the local optimum. The Step 2's "Mutation" operator will be essential at this time. "Mutation" operator will help the EA randomly distribute the population.

The convergence speed controller based on adaptive step length can help EA converge to local or global optimum quickly. EA-CSC works under the combination of Step 2 and Step 3 : Step 3 accelerates the converge speed of EA-CSC, and Step 2 allows individuals to jump out of local optimum. We assume that EA converges too slow so the performance is not well. Therefore, we employs convergence speed controller to accelerate the convergence speed of EA.

IV. Experiment

A. Benchmark Problems

For the evaluation, we chose a total of 7 tested problems. The detailed message of our benchmark problems is listed in the TABLE II. There are several kinds of testing problems that we can find in seven tested problems which are commonly used in software testing [10], [15]. To avoid bias in analyzing the experiment result, we will present and discuss the results of the empirical study for testing benchmark problems. In fact, different testing strategies can be applicable to different tested problems and environments. For example, random strategy is effective in testing container classes [16]. However, random testing could be misleadingly advantaged in numerical applications. Therefore, we need to carefully aggregate statistic on all the tested problems.

For each tested problem, three differen value ranges are arranged in our experiment. For example, "Triangle", "Factorial", "GCD", "Middle", "Commission" and "Premium" have three input value ranges: 1)$[1 - 10^2]$, 2)$[1 - 10^6]$ and 3)$[1 - the\ maximum\ integer]$. Similarly, there are also three instances in the tested problem "Quick Sort" with three different sorted number sizes of 5,10 and 20.

B. Experiment Setup

In our experiment, our proposed EA-CSC was compared with Random strategy and three heuristic algorithms which including the immune genetic algorithm (IGA) [9], artificial bee colony algorithm (ABC) [2], and particle swarm optimization (PSO) [13]. All the comparing algorithms' parameters setting was strictly followed their source references.

TABLE II: Benchmark Problems

Program	Loc	Dim	Description	Path
Triangle	21	3	Get the type of triangle of three input length values	4
Factorial	12	1	Produce the factorial for each element of integer form 1 to N	2
Quick Sort	44	5/10/20	Sort a array with quick sort method	4
GCD	16	2	Get the greatest common divisor of two integer numbers	4
Middle	20	3	Get the middle value of array of numbers	4
Commission	32	3	Get the commission of the turnover of a shop	3
Premium	56	2	Calculate the retirement pension for the elderly	11

* Loc, Dim and Path present the line number, input dimension and path number of each testing problem.

Evolutionary algorithms are influenced by its parameters. We find the "best practices" for many of these parameters: For example, we chose a crossover probability Mut of 0.1 based on past experiment. The population size of each strategy was 50, and all the algorithms will be executed 30 times. Different population size of an algorithm would lead to different performance. However, it is not possible to know beforehand which are the best value in a new tested problem. It is fair to choose a same population size for all the comparison algorithms. The stopping condition for all the algorithms is generating test case over 10^8 or getting test suite $\bar{X}$ for covering all the possible paths. The symbol "/" means that the corresponding strategy failed to cover all the possible path of tested problem within the maximum overhead of test cases at once. In order to distinguish the comparing algorithms, we used the Wilcoxon signed rank test [17] with the level of significance $\alpha = 0.05$ over 30 executions to analyze the essential difference between comparison algorithms. The parameters in calculating the fitness value were set as $K = 10$ and $\varepsilon = 0.006$ for each algorithms following the suggestion in [2] [9].

C. Results

The overhead of our proposed EA-CSC and the algorithms under comparsion is presented in TABLE III. As we can see from TABLE III, Average" and "Std" are the sample average and standard deviation of test case consumption for each trial. The symbol of Sig is "*" when there is significant different between the test case cost about EA-CSC and the comparing algorithm. We assume that there is an algorithm called $strategy1$ in our experiment. If the $strategy1$ can cover all the possible paths just under the case of maximum test case overhead, we could also draw a conclusion that $strategy1$ is not appropriate for this kind of tested problem. As there was no essential difference between $strategy1$ and other strategies which could not achieve complete coverage in trial of tested problem.

TABLE III: Comparing Results of Tested Programs

Name	EA-CSC		IGA			ABC			PSO			Random		
	Average	Std	Average	Std	Sig	Average	Std	Sig	Average	Std	Sig	Average	Std	Sig
Triangle1	1.08E03	3.22E02	2.21E03	1.15E03	*	7.72E02	6.25E02		1.91E03	1.23E03	*	2.00E03	1.21E03	*
Triangle2	4.12E03	5.46E02	1.86E07	7.03E06	*	1.58E05	4.06E04	*	1.50E07	6.50E06	*	2.16E07	9.67E06	*
Triangle3	1.77E04	6.70E03	/	/	*	1.62E07	1.56E05	*	/	/	*	/	/	*
Factorial1	1.04E02	3.54E01	1.15E03	9.46E02	*	6.46E02	5.76E02	*	1.90E03	1.73E03	*	1.93E03	1.97E03	*
Factorial2	2.11E03	1.19E03	8.21E06	2.34E06	*	4.95E06	3.38E06	*	1.86E07	1.20E07	*	2.13E07	1.71E07	*
Factorial3	1.17E04	3.51E03	/	/	*	/	/	*	/	/	*	/	/	*
Quick Sort1	3.24E02	3.47E02	1.88E02	1.68E02		1.69E02	1.30E02	*	2.77E02	3.02E02		2.14E02	1.24E02	
Quick Sort2	2.71E04	1.31E04	1.79E03	1.12E03	*	1.05E06	9.41E05	*	3.65E06	2.41E06	*	2.20E03	1.69E03	*
Quick Sort3	/	/	4.65E04	3.72E04	*	/	/		/	/		6.51E04	6.14E04	*
GCD1	4.90E01	3.37E01	1.40E02	1.54E02	*	5.20E01	4.12E01		9.30E01	7.10E01	*	2.16E02	2.78E02	*
GCD2	2.24E03	1.19E03	1.42E06	1.32E06	*	8.01E04	5.62E04	*	1.87E05	1.22E05	*	8.20E05	7.78E05	*
GCD3	1.24E04	5.70E03	/	/	*	6.68E07	3.03E07	*	5.36E07	3.22E07	/	/*		
Middle1	3.30E01	2.43E01	3.60E01	3.09E01		4.20E01	4.12E01		1.00E02	8.17E01	*	5.00E01	4.21E01	*
Middle2	7.72E02	4.62E02	1.04E06	7.58E05	*	2.94E05	2.13E05	*	1.49E06	1.08E06	*	1.35E06	1.00E06	*
Middle3	7.51E03	4.35E03	/	/		/	/	*	/	/	*	/	/*	
Commission1	1.31E03	4.01E02	4.49E04	5.31E04	*	1.86E03	1.35E03		3.01E05	4.54E05	*	6.18E04	7.62E04	*
Commission2	2.52E04	5.52E03	9.21E06	1.41E06	*	/	/	*	6.00E07	4.70E07	*	6.47E07	4.39E07	*
Commission3	1.29E05	3.64E04	/	/		/	/	*	/	/	*	/	/	*
Premium1	1.31E04	2.90E03	9.65E02	3.47E02	*	3.22E02	1.93E02	*	6.78E02	2.91E02	*	9.24E02	3.94E02	*
Premium2	1.48E04	3.78E03	7.42E06	2.25E06	*	6.73E05	5.62E05	*	6.38E06	3.07E06	*	7.45E06	3.12E06	*
Premium3	1.44E04	3.22E03	/	/	*	6.91E07	4.62E08	*	/	/	*	/	/	*
#.of'w-d-l'				16-2-3			14-5-2			18-2-1			18-1-2	

Average:the average cost of use case over the 30 executions
Std:standard deviation over the 30 executions
Sig:if the algorithm there is significant difference with EA-CSC,Sig is *,else Sig is empty
Method of significant analysis:non-parametric estimation, Wilcoxon signed rank test
#.of'w-d-l':the number of 'win-draw-lose' of EA-CSC versus the other algorithms

As we can see from TABLE III, we could see that EA-CSC had the lowest average test case cost in tested problem "Triangle1", "Triangle2", all "Factorial"s, all "GCD"s and all "Commission"s. Artificial Bee Colony algorithm [2] used the lowest test case cost in "Triangle1", EA-CSC used about 40% more overhead. However there was not significant different between the data of EA-CSC and ABC. We may said that the two algorithm hit a draw in "Triangle1". EA-CSC didn't perform well in three "Quick Sort" tested problem by comparing with heuristic algorithm IGA [9]. EA-CSC failed to achieve covering all the possible path at once in "Quick Sort3". This result can be explained by the fact that, each generation EA-CSC would noly find a input value's local optimum. However, tested problem "Quick Sort3" contains 20 input value while EA-CSC could only optimize one variable each generation.

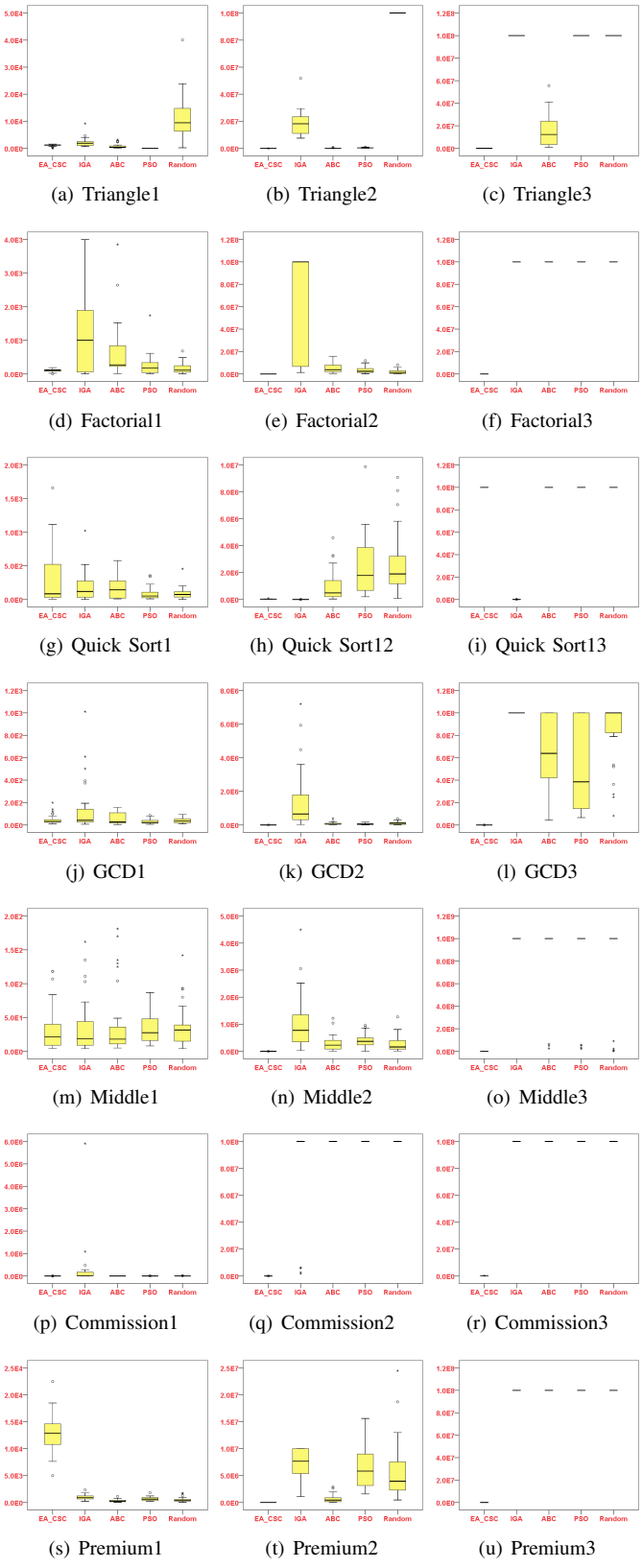
Notice that in three tested problem "Premium"s, the overhead of EA-CSC in all three tested problems are very similar. EA-CSC had a steady performance in those three tested problems. And EA-CSC had the lowest test case overhead in "Premium2" and "Premium3". This did not come as a surprise: For some "easy" tested problems, EA-CSC also use the adaptive step length to search the input domain. In these cases, the mutation operator is essential to help EA-CSC converge.

The box-plots in Fig. 3 compare the test case overhead (averaged out of the 30 runs) of EA-CSC and the comparison algorithms. In total, the data distribution of EA-CSC was centralized except subfigure(g), (m) and (s). Each algorithms had the similar performance in "Middle1" (m). EA-CSC used the maximum overhead of test cases in "Premium" (s). However, in the other 18 tested problems, the test case overhead of EA-CSC is centralized while other algorithms such as artificial bee colony algorithm (ABC), immune genetic algorithm (IGA), particle swarm optimization (PSO) and Random were not achieving all possible path many times especially in "Triangle3" (c), "Factorial3" (f), "GCD3" (l), "Middle3" (o), "Commission2" (q), "Commission3" (r) and "Premium3" (u).

In summary, our proposed EA-CSC used the minimum test cases overhead in most experiment: TABLE III illustrates that EA-CSC achieved using lowest average test case overhead 16 times in total 21 tested problems. The distribution of EA-CSC was centralized in 18 cases of 21 box-plots. However, EA-CSC is not appropriate for generating test cases in the tested problems which contains two many input values. This is the flaw in our algorithm, we will analyze and overcome it in the further work.

V. Conclusion

In this paper, we designed a kinds of convergence speed controller which used adaptive searching step length to improve evolutionary algorithm for solving automated test data generation for path coverage problem. The mutation operator is preserved at the same time, in order to redistribute the population. The mutation operator helps the algorithm get out of the local optimum and slows down the convergence speed. The idea of using adaptive searching step length of input value helps the evolutionary algorithm to accelerate the convergence speed while the mutation operator helps to jump out of local optimal solution. By repeating the above steps, EA-CSC finds all the possible path. The experiment results revolved that EA-CSC can stably used minimum test



(a) Triangle1 (b) Triangle2 (c) Triangle3

(d) Factorial1 (e) Factorial2 (f) Factorial3

(g) Quick Sort1 (h) Quick Sort12 (i) Quick Sort13

(j) GCD1 (k) GCD2 (l) GCD3

(m) Middle1 (n) Middle2 (o) Middle3

(p) Commission1 (q) Commission2 (r) Commission3

(s) Premium1 (t) Premium2 (u) Premium3

Fig. 3: Plot boxes of overhead for benchmark problems

[1]There are Five comparing strategy in those plot boxes, they are EA-CSC, IGA, ABC, PSO and Random from left to right.

[2]Each row below to a test problem. They are Triangle, Factorial, Quick Sort, GCD, Middle, Commission and Premium form (a) to (u).

case overhead in most tested problems. The success using of adaptive searching length CSC to improve evolutionary algorithm's performance for automated test data generation problem impress us the useful of CSC. We will do further work about the CSC and design more suitable CSC framework for software testing problem.

ACKNOWLEDGMENT

This work is supported by National Natural Science Foundation of China (61370102), Guangdong Natural Science Funds for Distinguished Young Scholar (2014A030306050),the Ministry of Education - China Mobile Research Funds (MCM20160206) and Guangdong High-level personnel of special support program(2014TQ01X664).

REFERENCES

[1] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesarwalawege, "A systematic review of the application and empirical investigation of search-based test case generation," *IEEE Transactions on Software Engineering*, vol. 36, no. 6, pp. 742–762, 2010.

[2] D. J. Mala, V. Mohan, and M. Kamalapriya, "Automated software test optimisation framework c an artificial bee colony optimisation-based approach," *Iet Software*, vol. 4, no. 5, pp. 334–348, 2010.

[3] C. C. Michael, G. Mcgraw, and M. A. Schatz, "Generating software test data by evolution," *IEEE Transactions on Software Engineering*, vol. 27, no. 12, pp. 1085–1110, 2001.

[4] J. R. Horgan, S. London, and M. R. Lyu, "Achieving software quality with testing coverage measures," *Computer*, vol. 27, no. 9, pp. 60–69, 1994.

[5] I. H. Osman and J. P. Kelly, "Meta-heuristics theory and applications," *Journal of the Operational Research Society*, vol. 48, no. 6, pp. 657–657, 1997.

[6] G. Gay, M. Staats, M. Whalen, S. Member, M. P. E. Heimdahl, and S. Member, "The Risks of Coverage-Directed Test Case Generation," *IEEE Transactions on Software Engineering*, vol. 41, no. 8, pp. 803–819, 2015.

[7] J. Wegener, A. Baresel, and H. Sthamer, "Evolutionary test environment for automatic structural testing," *Information and Software Technology*, vol. 43, no. 14, pp. 841–854, 2001.

[8] N. Tracey, J. Clark, and K. Mander, "The way forward for unifying dynamic test-case generation: The optimisation-based approach," *International Workshop on Dependable Computing and Its Applications*, pp. 169–180, 1994.

[9] A. Bouchachia, "An immune genetic algorithm for software test data generation," in *International Conference on Hybrid Intelligent Systems*, 2007, pp. 84–89.

[10] C. Mao, L. Xiao, X. Yu, and J. Chen, "Adapting ant colony optimization to generate test data for software structural testing ," *Swarm and Evolutionary Computation*, vol. 20, pp. 23–36, 2015.

[11] J. Kempka, P. Mcminn, and D. Sudholt, "Design and analysis of different alternating variable searches for search-based software testing ," *Theoretical Computer Science*, vol. 605, no. C, pp. 1–20, 2015.

[12] W. W. Xiangjuan YAO, Dunwei GONG, "Test data generation for multiple paths based on local evolution." in *Chinese Journal of Electronics, 2015, 24(CJE-1)*, 2015, pp. 46–51.

[13] B. Y. Shi and R. Eberhart, "A modified particle swarm optimizer, the 1998," in *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, 2010, pp. 69–73.

[14] B. Korel, "Automated software test data generation," *Software Engineering IEEE Transactions on*, vol. 16, no. 8, pp. 870–879, 1990.

[15] D. Gong, X. Yao, and W. Wang, "Test Data Generation for Multiple Paths Based on Local Evolution," *Chinese Journal of Electronics*, vol. 24, no. 1, pp. 46–51, 2015. [Online]. Available: http://digital-library.theiet.org/content/journals/10.1049/cje.2015.01.008

[16] G. Fraser and A. Arcuri, "Whole Test Suite Generation," *IEEE Transactions on Software Engineering*, vol. 39, no. 2, pp. 276–291, 2013.

[17] S. Garcła, D. Molina, M. Lozano, and F. Herrera, “A study on the use of non-parametric tests for analyzing the evolutionary algorithms behaviour: a case study on the cec2005 special session on real parameter optimization,” *Journal of Heuristics*, vol. 15, no. 6, pp. 617–644, 2009.