Research Article **Open Access**

Shi Jianqi, Huang Yanhong*, Li Ang, and Cai Fangda

An optimal solution for software testing case generation based on particle swarm optimization

https://doi.org/10.1515/phys-2018-0048
Received December 9, 2016; accepted April 26, 2017

Abstract: Searching based testing case generation technology converts the problem of testing case generation to function optimizations, through a fitness function, which is usually optimized using heuristic search algorithms. The particle swarm optimization (PSO) optimized testing case generation algorithm tends to lose population diversity of locally optimal solutions with low accuracy of local search. To overcome the above defects, a self-adaptive PSO based software testing case optimization algorithm is proposed. It adjusts the inertia weight dynamically according to the current iteration and average relative speed, to improve the performance of standard PSO. An improved alternating variable method is put forward to accelerate local search speed, which can coordinate both global and local search ability thereby improving the overall generation efficiency of testing cases. The experimental results demonstrate that the approach outlined here keeps higher testing case generation efficiency, and it shows certain advantages in coverage, evolution generation amount and running time when compared to standard PSO and GA-PSO.

Keywords: PSO, testing case, local search, inertia, instrumentation

PACS: 01.40.gf, 01.50.Kw, 06.60.Mr, 07.05.Bx, 29.85.-c

1 Introduction

Software engineering is a subject of Engineering Research on software development and software testing is the key component of software engineering, which directly affects the development prospects of software engineering. Currently, with the extension of the software scale, traditional software testing methods cannot satisfy actual demand, so automated testing of software becomes the emphasis of research today [1–4]. The key of automated testing lies in automated generation. Test case means a scientific organization induction for the activities of software testing, aiming at converting the activities of software testing to manageable modes. Testing case is also verified to be an effective process of quantifying the detailed tests [5–9]. Therefore, the research of testing case generation has great significance on software tests, and on the development of software engineering.

In the field of testing data autogeneration, there exists representative methods such as the symbolic execution method and random number method based on requirement specification, as well as the Korel [10] method etc. Symbolic execution method is often used to generate static testing data and it is not suitable for object-oriented programming. Random number method belongs to dynamic approaches. It generates random number according to the demand. When performing sample testing using a testing case, it can generate testing data with high effectiveness. The defect of such methods is that it is hard to acquire suitable solution sets for the high constraint test method such as path coverage or branch coverage. Korel is designed for the solution of path coverage of high constraint demand. It can adjust input variables continuously to achieve branch function minimization based on the information of branch functions in running programs. But it will also cause local optimal solution and is able to execute needed path when the initial data is bad. Jones et al. [11] adopts branch coverage criterion to propose a GA-based algorithm for testing case generation. They prove that GA-based testing data has a higher speed than random testing method in triangle classification programs. Anand [12] proposes simulated annealing to be implemented on the basic structure of testing generation data and provides empirical evidence for his research. However the algorithm lacks corresponding experiments and analysis. Sadiq [13] adopts variants of integrated learning PSO in structural software testing. It is used in testing for multiple artificial generation programs

Shi Jianqi, Li Ang, Cai Fangda: National Trusted Embedded Software Engineering Technology Research Center, East China Normal University, China; Shanghai 200062, China
***Corresponding Author: Huang Yanhong:** Huang Yanhong: National Trusted Embedded Software Engineering Technology Research Center, East China Normal University, Shanghai 200062, China; E-mail: yhhuang@sei.ecnu.edu.cn

that works as testing objects, and is compared to GA, thus verifying the effectiveness of PSO in software testing case generation. Jee [14] proposes an approximation algorithm like Tracy and he integrates global search to local search in PSO-based testing case generation algorithm. These algorithms also show the defects caused by the disadvantage of PSO, that is, bad local search accuracy and solutions with low quality.

We put emphasis on the searching based generation technologies, and aim at generating the best testing cases with high speed and efficiency, by the analysis and induction of current methods. Then a self-adaptive PSO based testing case generation scheme with optimized local search (SPSOLS) is proposed in this article. According to the actual demand, the dynamic strategy with adaptive adjustment of the inertia weight factor and local research are introduced in this article to balance global and local search of the algorithm, as well as the convergence speed and accuracy. For problems in slow convergence of local extreme rate and oscillation near the extreme value, a local search scheme based on geometrical enhancement is adopted to achieve better speed and performance. Finally the generation model in this article is verified with reaped to the indicators of the number of iterations and running time, compared to standard PSO and GA-PSO. The simulation results also show the improved algorithm has advantages in coverage rate and automatic generation efficiency.

The rest of this paper is organized as follows: The standard PSO algorithm is briefly presented in the next section. Section 3 fully introduces SPSOLS and the whole process of the improved scheme. Simulation and comparison results are provided in section 4. Finally, a conclusion is provided in section 5.

2 Standard PSO and its implementation

2.1 PSO model

PSO [15, 16] is an optimization search technique based on population. It is a new branch of evolutionary computation and the particle swarm can be seen as a simple social system. In PSO, each individual is called a particle and each particle denotes a potential solution, which forms the population set. The particles in searching space influence each other, exchanging information, to adjust the location and speed of itself and approach to the optimal solution. PSO initialize a group of particles, finding he optimal solution by iteration. During each iteration, the particle updates its speed and location by tracing individual extremum and global extremum.

$$v_{id}^{t+1} = v_{id}^{t} + r_1 c_1 (p_{id} - x_{id}^{'}) + r_2 c_2 (p_{gd} - x_{id}^{'}) \quad (1)$$

$$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1} \quad (2)$$

$v_i = \{v_{i1}, v_{i2}, \ldots, v_{id}\}$ denotes the speed of the i_{th} particle; $X_i = \{x_{i1}, x_{i2}, \ldots, x_{id}\}$, $i = 1, 2, \ldots, N$ denotes a vector point of i_{th} particle in d-dimension solution space, t is the iteration number; x_{id}^{t} and $v_{id}^{'}$ denote the location after t_{th} iteration and d_{th} dimensional component of speed vector of i_{th} particle, r_1 and r_2 are random numbers obeying the distribution of $U(0, 1)$; c_1 and c_2 are accelerate factors and usually $c_1 = c_2 = 2$. We use $P_i = \{p_{i1}, p_{i2}, \ldots, p_{id}\}$ to record the optimal point that is searched by the i_{th} particle, as p_{best}. Thus there must exist an optimal point in the population, whose number is g. Then $P_g = \{p_{g1}, p_{g2}, \ldots, p_{gd}\}$ is the optimal value of the population search.

During the process of particle optimization, it is crucial to balance the local developing ability and the global detecting ability. For different problems the balance of these two abilities are not the same. Therefore, Shi [17] introduces inertia weight in equation 1 and it is revised as

$$v_{id}^{t+1} = \omega v_{id}^{'} + r_1 c_1 (p_{id} - x_{id}^{'}) + r_2 c_2 (p_{gd} - x_{id}^{'}) \quad (3)$$

ω, called inertia factor, is a normal number or it may be a linear or nonlinear positive number, with time as a variable. When $\omega = 1$, equation 3 is changed to equation 1, so PSO with inertia factor is an extended form of standard PSO. ω keeps motion inertia of particle and the ability such as trend of extending search space and explore novel regions. It balances global detecting and local development. Therefore, a suitable value of ω implies the improvement of convergence accuracy and speed, which increases the probability of PSO to find the global optimal solution.

When $\omega \geq 1$, the speed is increased with time and the particle will not change direction towards a good region, leading to population divergence; when $0 < \omega < 1$, the particle decelerates and the convergence of the whole population is decided by c_1 and c_2. To get better global search ability at an early stage, and better local development ability at a later stage, ω is set to undergo linear reduction with the evolution.

2.2 Implementation of PSO algorithm on automatic testing data generation

The problem of automatic testing data generation can be expressed as a function minimization problem. The

branches of the program are taken as a function. When we execute the function and approach to a expected location in the code, the function will be expected to reach the desired location of one or more variable values as a parameter of function [18]. As long as a set of input data can be found to achieve the minimum value, they are the requiring input data in need.

Assuming the 324th line in a program contains the condition if (cps≥ 20).

The object of use is to ensure the execution of this branch whose condition is true. Therefore we must find an input to ensure that when the 112th line arrives at a value of variable cps that is greater than or equal to 20. A simple way is to record the value of cps when program comes to 112th line. We use cps_{121} to record this value with input data x. Then the branch problem is converted to a function minimization problem, when 112th execution is true:

$$f(x) = \begin{cases} 20 - cps_{121}(x), & cps_{121}(x) < 20 \\ 0, & otherwise \end{cases} \quad (4)$$

To find the expected input data, it is needed to find a value for x such that $f(x)$ achieves the minimized value, so $f(x)$ indicates the degree that input data approaches the expected result. Generally it is believed that the smaller of $f(x)$ is, the closer of input data approaching expected result is. Thus, the initial input data can be adjusted and $f(x)$ can be used to evaluate the method used to adjust the data for a closer result to the expected object. Test data generator is instructed by the value of $f(x)$ to approach the final object by a series of adjustments. In PSO, $f(x)$ often works as the fitness value to compute the testing case, determining the speed and direction of particles.

3 Testing case optimization algorithm based on SPSOLS

3.1 System model of testing case generation

The core problem of PSO-based testing data generation is ensuring the cooperation of PSO search algorithm and testing process. The system is decomposed into test modules and core algorithm modules, as depicted in figure 1. From the test point of view, it can be divided into the initial preparation of the search algorithm and the interaction part with search algorithm during the execution. The preparation part mainly includes the following work:

(1) Establish fitness function of coverage criterion according to the internal structure of program;

(2) Perform instrumentation on the program structure elements corresponding to the coverage criterion to acquire coverage information;

(3) Extract interface information of program to prepare for the work that describes the input parameter codes for the location vector of the particles.

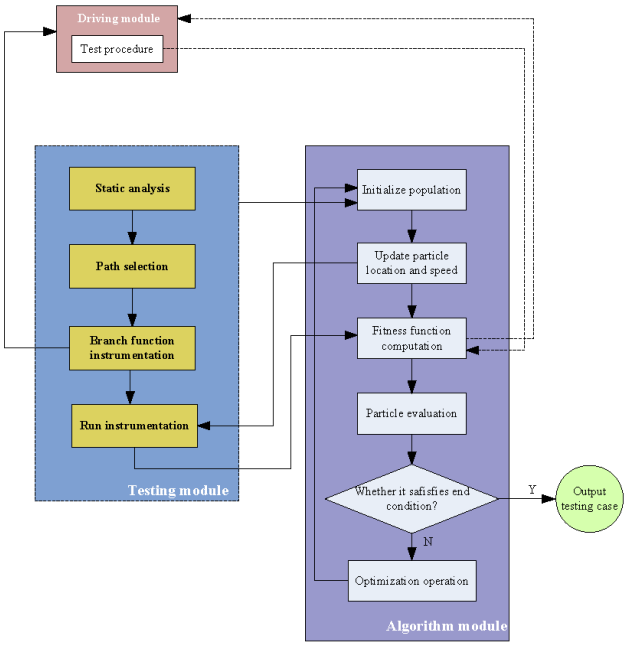


Figure 1: Testing case generation system model

Algorithm module is a key part of the system. It creates an initialized particle swarm according to previous procedures, and instructs the particles to approach the object, finding final solution or arriving at the expected number of iterations, by repeated evaluation and operation of the particles in the population. The testing running environment provides testing programs that can be called in real-time and instrumented in running phase. It evaluates the status of each particle in population and return a fitness value for the core algorithm.

The generation process will observe the following steps:

Step 1: Analyze the source codes of tested program to draw its control flowchart.

Step 2: Choose a needed testing path and specify fitness function according to the conditions in path. Make instrumentation on source programs.

Step 3: Generate testing case set in the definition domain of program randomly in demand.

Step 4: Acquire corresponding fitness with initial testing case executing instrumented programs and check

whether it executes the expected path. If it does, turn to step 6; otherwise, turn to step 5.

Step 5: Run improved PSO algorithm package according to the fitness of particle to generate new testing cases, and turn to step 4.

Step 6: End of run, output appropriate testing cases.

3.2 Self-adaptive adjustment of PSO parameter

Inertia weight factor ω directly influences the convergence of the algorithm and it is crucial to the performance. The adjustment has been performed in literature [19], but the diversity of particles is still an important factor. During the search for a solution, the diversity is gradually decreased, which shows homoplasy. It tends to get to the local extremum before the global solution is found. The approach of chasing diversity of particles will cause bad convergence speed. Therefore, this article proposes a self-adaptive scheme, which integrates the fitness and aggregation degree of particles to set ω with different methods. Then ω can be dynamically adjusted to ensure the global search ability of algorithm without affecting the convergence speed.

The novel adjusting method is based on fuzzy logic and it can improve the performance of the PSO algorithm, First we set two decision variables of ω, the number of current iteration N_t and the average relative speed $\Delta V_{cs}(t)$.

$$\begin{cases} \Delta V_{cs}(t) = |V_{cs}(t) - V_{cs}(t-1)| \\ V_{cs}(t) = \dfrac{\sum\limits_{i}\sum\limits_{j} v_{id}}{ij} \end{cases} \tag{5}$$

The input variables are t and $\Delta V_{cs}(t)$, and the output variable is ω in the fuzzy structure. During the process of fuzzification we choose fuzzy word set (S, M, L): S →small, M →middle, L →large. The diagram of membership is depicted as figure 2.

We establish 9 dynamic fuzzy adjustment rules for ω, as described in table 1. It is obvious to satisfy different demand with adaptive mechanism for the inertia weight at different search stages. It also means a large inertia expectation will change the value of x at the initial stage to a certain extent, whilst a smaller inertia expectation can be taken as the global optimal solution in certain interval in search. The fuzzy rules are based on the following three facts:

- When the algorithm starts, a bigger inertia weight is adopted to increase the diversity of searching space, causing a strong exploration in population. In addition, the last procedure in inertia weight decreasing can lead to a faster convergence of population to find a global solution. Thus it has better performance to search the solution in a small region.
- When the particle approaches the object and $p_{id} \to p_{gd}$, the value of $[p_{id} - x_{id}(t)]$ and $[p_{gd} - x_{id}(t)]$ approaches zero. It means that with the decreasing speed, the decreasing speed of small inertia weight will affect the location of the next particle, so the particle may be converged to a solution up to now. There exists a risk that if a population converges too fast it may not be the global solution for the particle to explore the next attempt. Therefore, larger inertia weight should be chosen to avoid getting into local optimization.
- In PSO, a suitable control of global exploration and local development is to find the optimal solution effectively. To balance exploration and development we can adjust the value of ω dynamically to adapt the change of $\Delta V_{cs}(t)$ for relative speed V. In other words, we expect to maintain the diversity of population by choosing a larger inertia weight value. Though the average relative speed is small, local optimization can be avoided, while smaller inertia weight value also maintain a better convergence ability to get a smaller ΔV_{cs} finally. A larger ΔV_{cs} is suggested to avoid the swarm lies on the boundary of space, with setting of a small ω.

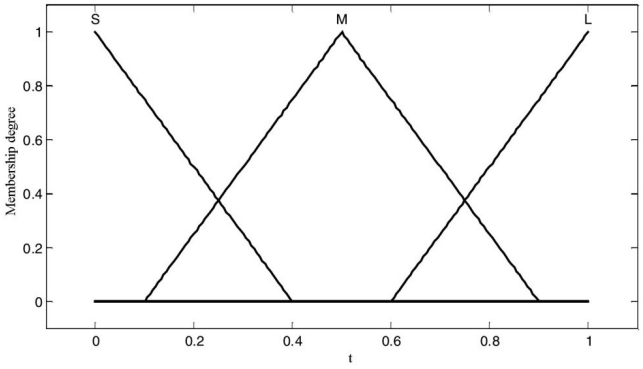


Figure 2: Membership degree change in fuzzy inference system

The detailed steps for fuzzy adjustment are:

Step 1: Initialize the particles with asymmetric method. Repeat step 2 to step 6 until the demand is satisfied.

Step 2: Evaluate the fitness of each particle

Step 3: For each particle i and $p_{gd} = x_{id}$, if $f_i < f_{best}$, $\forall i \le N$

Table 1: Fuzzy rules to compute ω

$V_{cs}(t)$ \ t	S	M	L
S	L	L	M
M	L	M	S
L	M	S	S

Step 4: For each particle i and $p_{gd} = x_{id}$, if $f_i < f_{best}[i]$, $\forall i \le N$

Step 5: Adjust the value of ω based on the normalization of input variables

Step 6: update the speed and location of each particle according to equations 1 and 2.

3.3 Local search strategy based on improved alternating variable method

The major concept of alternating variable method (AVM) is to adjust each dimension of input variable in turn until the fitness cannot be raised any further. Then the value of the next dimensional value will be revised until the fitness value of all the variables cannot be improved any more. The adjusting method adopts a hill-climbing algorithm, that is, finding an optimal solution in the neighbour range of the variable. If a new solution is found and it is better than the current solution, alternate current solution and continue to search the neighbour domain of the new solution. Then the search is iterated to find the optimal solutions so the basis of AVM is a hill-climbing algorithm. The result in literature [20] indicates that for most simple programs AVM is better than GA, but AVM is only suitable for unimodal function. When the test programs are multiple peaks, it will get into local optimization and lose its ability to find the optimal solution.

We propose an improved local search based on AVM to make a more effective search in optimal peak function with geometric hierarchical progression, which is a variant of binary search. This idea executes pattern search and uses binary search until the object in found. It can be seen that the optimization of any peak function depends on the discovery of the logarithm of the initial time distance, which is called geometric research since the pattern search adopts a number of geometric sequences. The following parts offer the procedure of research process:

Geometrical hierarchical progression differentiates the same geometry to perform a search as IPS, while it provides two optimal solutions by variant binary search. If we adopt pattern search and the search points x_{j-1}, x_j, x_{j+1},

Input: The location of $x[]$ of the bet particle
Output: The location of $x[]$ of the bet particle in P_t

```
1  if f(x − 1) ≥ f(x) and f(x + 1) ≥ f(x) then
2  |  return x
3  end
4  if f(x − 1) ≤ f(x + 1) then
5  |  k = −1
6  else
7  |  k = 1
8  end
9  while f(x + k) < f(x) do
10 |  Let l = min(x − k/2, x + k),
   |    r = max(x − k/2, x + k)
11 end
12 while l ≠ r do
13 |  if f(⌊(l + r)/2⌋) < f(⌊(l + r)/2⌋ + 1) then
14 |  |  r = ⌊(l + r)/2⌋
15 |  else
16 |  |  l = ⌊(l + r)/2⌋ + 1
17 |  |  return l
18 |  end
19 end
```

and $f(x_{j-1}) > f(x_j) \le f(x_{j+1})$, it indicates that if f is a unimodal function, the global optimization will be produced in set $\{x_{j-1}, x_j, \ldots, x_{j+1}\}$.

3.4 Construction and instrumentation of fitness function

The fitness function is constructed by the Branch distance method [10]. Branch distance denotes the distance between the input variable and a given Branch predicate, which offers corresponding methods to compute distance among predicates for different relations. Tracey et al [21] improved this algorithm, so we adopt their improved scheme to construct the fitness function of each branch in the programs under test. In addition, we introduce parameter weight w to assign different weights according to difficulty level of branch coverage, which is determined by two factors: boundary condition and branch hierarchy. This is illustrated by the program code "Is Valid Date" [22] described below. Branch 1-3 is more difficult to cover than branch 4-6, in view of the boundary condition; In view of the branch hierarchy, branch 5-6 lies in the third layer and they can be covered only after branch 3 and 4.

```
Program Is Valid Date
...
// branch 1
If (month==1||)|| month==3|| month==5|| month==7
|| month==8 || month==10 || month==12) {
...; }
// branch 2
else if (month==4||)|| month==6|| month==9|| month==7
|| month==11) {
...; }
// branch 3
else if (month==2||) {
...; }
// branch4
If ((year%4==0 && year%100!=0) || year%400==0)
// leap year judgement
{
// branch5
If (date>29 || date <1) {
...; }
// branch6
Else {
...;
}
...
}}
...
```

The fitness value computation is shown as follows:

$$f = 1/[0.01 + \sum_{i=1}^{x} w_i f(bra_i)], \tag{6}$$

where $f(bra_i)$ is the branch distance function of i_{th} branch; w_i is corresponding weight and $\sum_{i=1}^{x} w_i = 1$. The fitness value is decided by the number of branch coverage. The fitness function increases the difference between a good candidate solution and a bad one. Therefore, it fastens the convergence speed of search algorithm to a certain extent.

4 Performance tests

4.1 Testing case to generate common triangular paths

In this experiment we choose classic Triangular decision procedures without polymorphic information to test our algorithm with other evolutionary algorithms. The problem is: input three integers as the sides of a triangle. Compare the sides and output the type of triangle, including ordinary triangle, equilateral triangle, isosceles triangle, and non triangle, output by program modules. Since the decision procedure has many judgements and there is no loop structure or polymorphic, we use branch function overlaps to construct the instrument function.

First we focus on three indicators of the algorithms, average iteration, coverage rate and time consumption. Table 2 depicts the related test results about three algorithms in triangle generation test. SPSOLS has the minimum iteration and average time consumption of the other algorithms. The iteration number is reduced to 1/10 of standard PSO and 1/6 of GAPSO, while the time cost is about 1/11 of standard PSO and 1/8 of GAPSO. All the algorithms have higher coverage rate and can correctly output the testing paths of testing cases. SPSOLS keeps relative stable coverage rate with the increase of convergence speed. The increase of population size brings few effects on the algorithm performance. So our scheme shows better comprehensive performance improvement with little cost of price.

4.2 Performance comparisons under standard testing programs

In this experiment we choose 8 programs for testing. 5 of them are of independent patterns and the other 3 are not. The detailed information of these programs is described in table 2.

From the experimental results as shown in figure 3 which provides the performance comparison distribution in three indexes, SPSOLS-based testing data generation are superior than the other two algorithms in all programs. Towards the average coverage rage of 8 testing programs, its results are more than 98%, meaning it arrives at complete branch coverage in most tests. For standard PSO and GAPSO, they can achieve part of complete branch coverage. In the programs, the worst coverage of PSO is 88.79% and that of GAPSO is 95.47%.

The convergence speed of SPSOLS is 2 times of PSO and GAPSO. Illustrated by program “Equals”, PSO needs 16 generation to get the optimal solution, while GAPSO only needs 5 generation to get convergence. For other programs with simple independent pattern, the average speed of SPSOLS increased by 70 percent or so compared to the other two algorithms.

For the time consumption, GAPSO-based testing case generation has obvious advantage over the other two algorithms. Standard PSO has nearly ten times average time

Table 2: Testing results of three algorithms in iteration number, time consumption and coverage rate

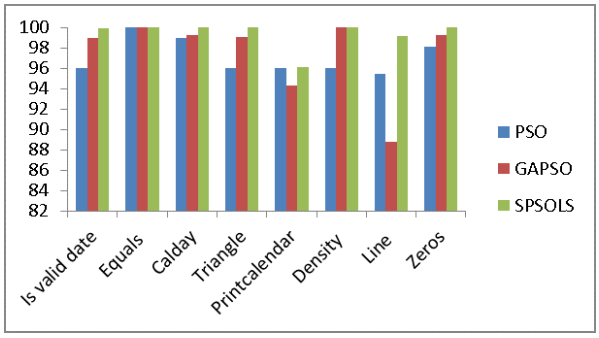
Population size	Iteration number of optimal solution			Iteration time consumption of optimal solution (ms)			Average coverage rate of optimal solution (%)		
	SPSOLS	PSO	GAPSO	SPSOLS	PSO	GAPSO	SPSOLS	PSO	GAPSO
10	22.4	232.1	123.4	21.1	320.1	188.3	99.9	95.2	95.6
20	19.6	239.4	111.1	19.3	286.5	160.5	99.8	941	97.7
30	16.8	156.8	84.7	18.6	262.3	154.3	99.1	95.1	96.5
40	9.8	160.1	87.3	19.2	260.1	150.7	99.9	96.3	98.2
50	11.3	113.5	71.6	17.5	227.3	134.4	99.9	94.2	94.4
100	9.9	102.5	79.2	22.3	199.8	139.1	99.9	93.4	93.9

Table 3: Program under test

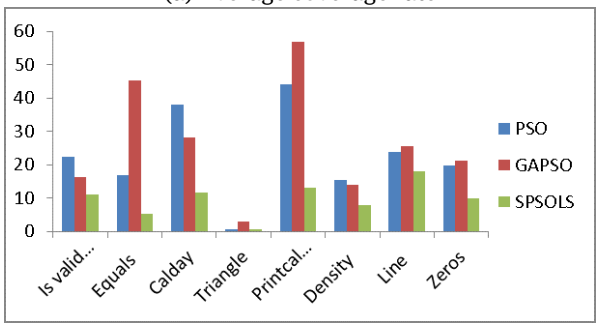
NO.	Name	Variables number	Number of independent pattern	Source
1	Is valid date	15	16	Literature [22]
2	Equals	20	10	Java collections
3	Calday	11	10	Literature [9]
4	Triangle	3	0	Literature [4]
5	PrintCalendar	33	2	Literature [5]
6	Density	4	0	Apache commons math
7	Line	36	0	Literature [3]
8	Zeros	20	20	Literature [3]

consumption than SPSOLS and GAPSO. Illustrated by program "Density", SPSOLS saves about 100 times of seconds of PSO.

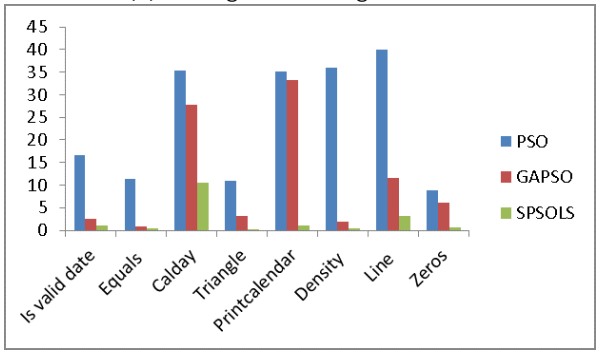
Specially, we choose 4 of representative programs to analyze the relation between evolutionary generation number and average coverage and the comparison curves are shown in figure 4. As summarized above, SPSOLS always obtain the highest branch coverage rate. For most of the programs, GAPSO has faster increasing speed of coverage rate than standard PSO. For program "line" and "Printcalendar", the convergence speed relation of PSO and GAPSO are contrary. PSO shows frequent fluctuation in local range and the coverage rate has a stable increase while GAPSO has local stability. For a few programs the coverage rate shows a little degeneration, that is, the cov-



(a) Average coverage rate

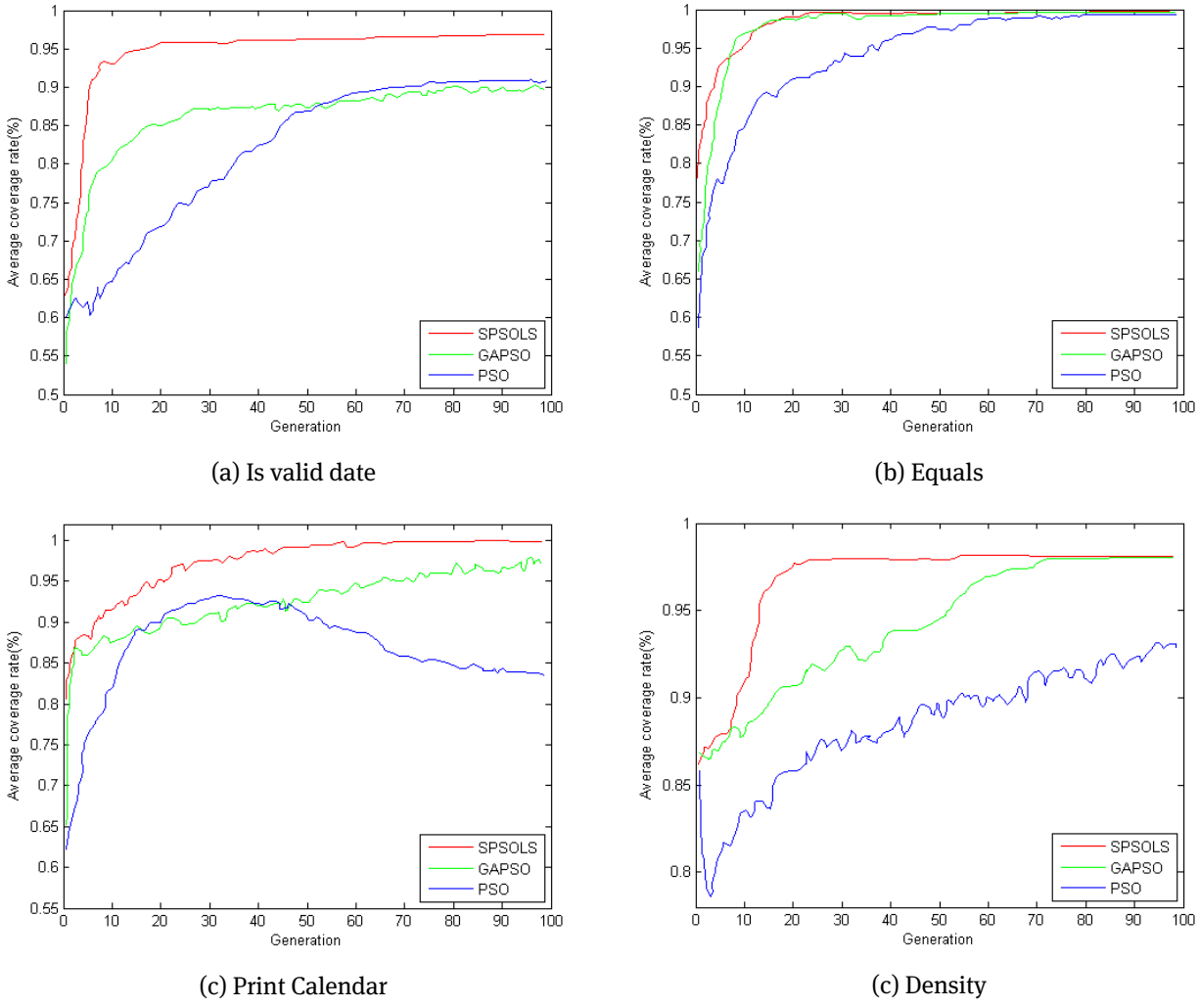


(b) Average iteration generation



(c) Time consumption

Figure 3: Testing results of three algorithms about their performance indexes



(a) Is valid date

(b) Equals

(c) Print Calendar

(c) Density

Figure 4: The relation between branch coverage rate and evolution generation

erage rate is inversely proportional to the evolutionary generation number. The algorithm proposed in this article has comprehensive performance improvement and it is more stable in the convergence process.

5 Conclusion

PSO algorithm has deficiency in premature convergence, and the local search accuracy is low. It also brings factors of program structure in testing case generation. Therefore the PSO-based testing case generation technology is discussed intensively in this article. To balance the ability of exploring and self-improvement of algorithms, and to achieve better convergence speed in global search, we provide adaptive particle inertia weight factor adjusting approach, integrated with fitness and particle aggregation degree. During evolution, a local searching strategy is performed on the optimal individual of each generation to further improve the efficiency of testing case generation. The simulations indicate that the SPSOLS algorithm proposed in this article shows better testing case generation performance and it achieves coverage rate optimization of tested cases. It also has certain advantages in testing case generation compared to homogeneous algorithm under approximate environment.

Acknowledgement: This work is partially supported by the Science and Technology Commission of Shanghai Municipality Projects (No. 15511104700, No. 16DZ1100600 and No. 15ZR1410400), National Natural Science Foundation of China Projects (No. 61602178), The Integration of Industry, Education and Research Project of SHEITC, Shanghai. (No. CXY-2015-015)

References

[1] Bertolino A., Software testing research: Achievements, challenges, dreams, Int. Conf. on Soft. Eng. (23-25, May, 2007, Minneapolis, MN, USA), IEEE Computer Society Washington, DC, USA, 2007, 85-103.

[2] Gascoyne S., Productivity improvements in software testing with test automation, Electr. Eng., 2000, 72(885), 65-66.

[3] Wang L., Issues on software testing for safety-critical real-time automation systems, AIAA/IEEE Digital Avionics Syst. Conf. - Proceedings (28-28, Oct, 2004, Salt Lake City, UT, USA, USA), IEEE, 2004, 2(10), 2-12.

[4] Baek C., Park S., Choi K.,TEST: An effective automation tool for testing embedded software, In: Mastorakis N.E. (Ed.), Proceedings of the 9th WSEAS Int. Conf. on Computers (14, July, 2005, Stevens Point, Wisconsin, USA), World Scientific and Engineering Academy and Society (WSEAS), 2005, 2(8), 1214-1219.

[5] María R., Núñez, M.L.G., Multiplier method and exact solutions for a density dependent reaction-diffusion equation, Appl. Math. Nonlin. Sci., 2016, 1(2), 311-320.

[6] Gallagher M., Narasimhan V. L., ADTEST: A Test Data Generation Suite for Ada Software Systems, IEEE Trans. Soft. Eng., 1997, 23(8), 473-484.

[7] Hametner R., Kormann B., Vogel-Heuser B., Test case generation approach for industrial automation systems, Proceedings of the 5th Int. Conf. on Automation, Robotics and Applications (6-8, Dec, 2011, Wellington, New Zealand), IEEE, 02 30 February 2012, 57-62.

[8] Liu J., Yang Z., Yang Z.X., Test case generation based on orthogonal table for software black-box testing, J. Harbin Inst. Techn. (New Series), 2008, 15(3), 365-368

[9] Clerc M., The Swarm and the Quceen: Towards a deterministic and Adaptive Particle Swans Optimizatior, proceedings of Congress Evolutionary Computation (6-9 July 1999, Washington, DC, USA, USA) IEEE, 1999, 951-957, DOI: 10.1109/CEC.1999.785513

[10] Korel B., Automated software test data generation, IEEE Trans. Soft. Eng., 1990, 16(8), 870-879, DOI:10.1109/32.57624

[11] Ribeiro J.C., Zenha-Rela M.A., Fernández V.F., Test case evaluation and input domain reduction strategies for the evolutionary testing of object-oriented software, Inf. Soft. Technol., 2009, 51(11), 1534-1548.

[12] Anand S., Burke E.K., Chen T.Y., Clark J., Cohen M.B., Grieskamp W. et al., An orchestrated survey of methodologies for automated software test case generation, J. Syst. Soft., 2013, 86(8), 1978-2001.

[13] Sadiq M. Firoze F., A fuzzy based approach for the selection of software testing automation framework, In: Jain L., Behera H., Mandal J., Mohapatra D. (Eds.), Computational Intelligence in Data Mining - Volume, Springer India, 2015,444-449.

[14] Jee E., Shin D., Cha S., Automated test case generation for FBD programs implementing reactor protection system software, Software Testing Verification and Reliability, 2014, 24(8), 608-618.

[15] Nie P., Geng J. Qin Z., Self-adaptive inertia weight PSO test case generation algorithm considering prematurity restraining, Int. J. Dig. Cont. Technol. Appl., 2011, 5(9), 125-133

[16] Ahirwal M.K, Kumar A., Singh G.K., Analysis and testing of PSO variants through application in EEG/ERP adaptive filtering approach, Biomed. Eng. Let., 2012, 2(3),186-197.

[17] Shi Y., Eberhart R., A modified particle swarm optimizer, IEEE International Conference on Evolutionary Computation (4-9 May 1998), Anchorage, AK, USA, 1998, 69-73.

[18] Hosamani S.M., Correlation of domination parameters with physicochemical properties of octane isomers. Applied Mathematics & Nonlinear Sciences, 2015, 1, 345-352.

[19] Mishra K.K., Tiwari S., Misra A.K., Improved environmental adaption method and its application in test case generation, Journal of Intelligent and Fuzzy Systems, 2014, 27(5), 2305-2317.

[20] Kempka J., McMinn P., Sudholt D., A theoretical runtime and empirical analysis of different alternating variable searches for search-based testing, In: Blum Ch. Ikerbasque, GECCOGenetic and Evolutionary Computation Conf. (06 - 10, July, 2013, Amsterdam, The Netherlands), ACM New York, NY, USA, 2013, 1445-1452.

[21] Tracey N., Clark J., Mander K., et al., An automated framework for structural test-data generation, Proceedings of International Conference on Automated Software Engineering(13-16, Oct, 1998, Honolulu, HI, USA, USA), IEEE Computer Society Washington, DC, USA, 1998, 285-288.

[22] Cheng G., Huang J.Y, Yue S., Particle swarm optimization based RVM classifier for non-linear circuit fault diagnosis. Journal of Central South University of Technology, 2012, 19(2), 459-464.