Vol 18, Summer 1404

Information Technology in Engineering Design

https://sanad.iau.ir/journal/ited

Comparative Analysis of the Latest Improvements of Particle Swarm Optimization Algorithms on Automated Software Test Data Generation

Mojtaba Salehi[(1)] Saeed Parsa[(2)*] Saba Joudaki[(3)] Hoshang Kolivand[(4)]

(1) Department of Computer Engineering, Bo. C., Islamic Azad University, Borujerd, Iran
(2) Department of Computer Engineering, Tehran Branch, Iran University of Science and Technology, Tehran, Iran*
(3) Department of Computer Engineering, Khor.C., Islamic Azad University, Khorramabad, Iran
(4) School of Computer Science and Maths, Liverpool John Moores University, L3 3AF, Liverpool, UK

(Date received: 1403/09/08 Date accepted: 1404/02/27)

Abstract

This study examines enhancements to Particle Swarm Optimization (PSO) algorithms for generating automated software test data, with a focus on path coverage in software systems. PSO, valued for its scalability and simplicity, has been enhanced through various algorithmic improvements to address challenges like local optima entrapment and convergence inefficiencies. The research evaluates ten recent PSO variants on benchmark programs, comparing their performance based on coverage, runtime, and success rates. Among these, the A5 (EBPSO) algorithm demonstrated superior performance, excelling in exploration-exploitation balance, population diversity, and convergence efficiency. Experimental results affirm the efficacy of meta-heuristic approaches in generating test data across expansive search spaces, positioning A5 as a leading solution for test data generation in optimizing software testing processes.

Keywords: Test data generation, meta-heuristic algorithms, particle swarm optimization, path coverage

* Corresponding author:

Saeed Parsa
Address: Department of Computer Engineering, Tehran Branch, Iran University of Science and Technology, Tehran, Iran

Email: parsa@iust.ac.ir

1- Introduction

Although random testing is a low-cost approach to software testing, it struggles to address complex constraints and achieve comprehensive structural coverage. Search-based software testing (SBST) utilizes optimization techniques to generate test data, thereby efficiently overcoming these limitations. SBST leverages meta-heuristic algorithms, which excel at navigating large search spaces without assumptions about problem characteristics. Foundational studies by McMinn [1] and Harman et al. [2] highlighted the effectiveness of meta-heuristic approaches, including Particle Swarm Optimization (PSO) [3], Hill Climbing (HC) [4], Tabu Search (TS) [5], Ant Colony Optimization (ACO) [6], Differential Evolution (DE) [7], Artificial Bee Colony (ABC) [8], and Firefly Algorithm (FA) [9].

Significant advancements in these algorithms have enhanced their adaptability and performance. Liang et al. [10] introduced a DE algorithm with a one-test-at-a-time strategy, achieving notable improvements in solution quality through parameter optimization. Malhotra et al. [11] demonstrated that ABC outperformed ACO and GA in test data generation due to its efficient parallelism and neighborhood production mechanism. Similarly, Srivatsava et al. [12] employed the FA to optimize test paths, demonstrating its ability to minimize test efforts through features like guidance matrices and cyclomatic complexity-based traversal.

Inspired by swarm intelligence, PSO has emerged as a leading method for generating test data. PSO optimizes candidate solutions for improved coverage and efficiency by iteratively updating the positions and velocities of particles. Tiwari et al. [3] applied a PSO variant to regression testing, achieving superior path coverage. Jatana et al. [13] compared PSO with GA in mutation testing, finding that PSO offered faster convergence and smaller test suites. Sahoo et al. [14] introduced an Improved Combined Fitness function with Adaptive PSO (APSO), showcasing its ability to enhance critical path coverage.

This paper investigates the performance of ten advanced PSO variants in automated test data generation. Among these, Algorithm A5 emerged as the most effective, utilizing innovative solution-updating strategies that adapt to search space conditions. By balancing exploration and exploitation, Algorithm A5 mitigates the entrapment in local optima, enhances population diversity, and leverages collective intelligence to navigate the search space efficiently.

The rest of this paper is structured as follows: Section 2 reviews related work on SBST and PSO applications. Section 3 outlines the theoretical background supporting the proposed approach. Section 4 introduces the improved PSO algorithms, details the experimental setup, and presents a comparative analysis of their performance. Finally, Section 5 summarizes the findings, emphasizing the superiority of Algorithm A5, and discusses potential directions for future research.

2- Related Work

Test data generation is a critical challenge in software testing, with random testing often failing under complex constraints. Search-based software testing (SBST) employs meta-heuristic algorithms, such as Particle Swarm Optimization (PSO) [3], Hill Climbing [4], and Differential Evolution (DE) [7], to address these issues by efficiently navigating large search spaces without assumptions about problem characteristics [1, 2]. Liang et al. [10] improved DE with a one-test-at-a-time strategy, while Malhotra et al. [11] showed ABC's superiority in path coverage due to its robust parallelism. Srivatsava et al. [12] optimized test paths using the Firefly Algorithm (FA), and Lv et al. [15] proposed a metamorphic relations-based approach as an alternative to PSO despite its reliance on domain expertise.

PSO has demonstrated notable success in testing, with Tiwari et al. [3] applying it to regression testing and Sahoo [14] enhancing it with a fitness function to achieve 100% path coverage. Saadatjoo [16] and Ghiduk [17] demonstrated the potential of evolutionary and PSO-based methods for optimizing test data generation. Recent efforts, including Damia's adaptive PSO [21, 22] and Esnaashari's hybrid memetic algorithm [23], have addressed the limitations of PSO, such as entrapment in local optima and convergence issues. Researchers such as Semujju [19] and Rajagopal [20] have developed adaptive and hybrid approaches, enhancing efficiency in challenging testing scenarios.

These studies affirm the versatility of PSO and its variants in tackling diverse software testing challenges, with continuous innovations improving adaptability, efficiency, and coverage.

3- Background

This section describes the theoretical background being used in our proposed approach. This section introduces the basic concepts of software testing, path testing, and fitness functions, which will help us understand the test data generation based on the PSO algorithm.

3-1- Control Flow Graph

The control flow graph was constructed using source code. Source code is taken as input, and the instrumented code is generated [1]. After that, instrumented code is parsed and stored, line by line, in the adjacency list. CFG for a program P is a directed graph $G = (N, E)$ consisting of N as a set of nodes and E as a set of edges. A node represents a statement or a basic block of statements [2]. An edge represents the flow of control between nodes. CFG is generated using software such as Visustin or the pycfg or staticfg library in the Python programming language, and then extracts the paths of this graph from the desired graph. An example of the flow graph with the resulting program from Visustin output is shown in Fig.1.

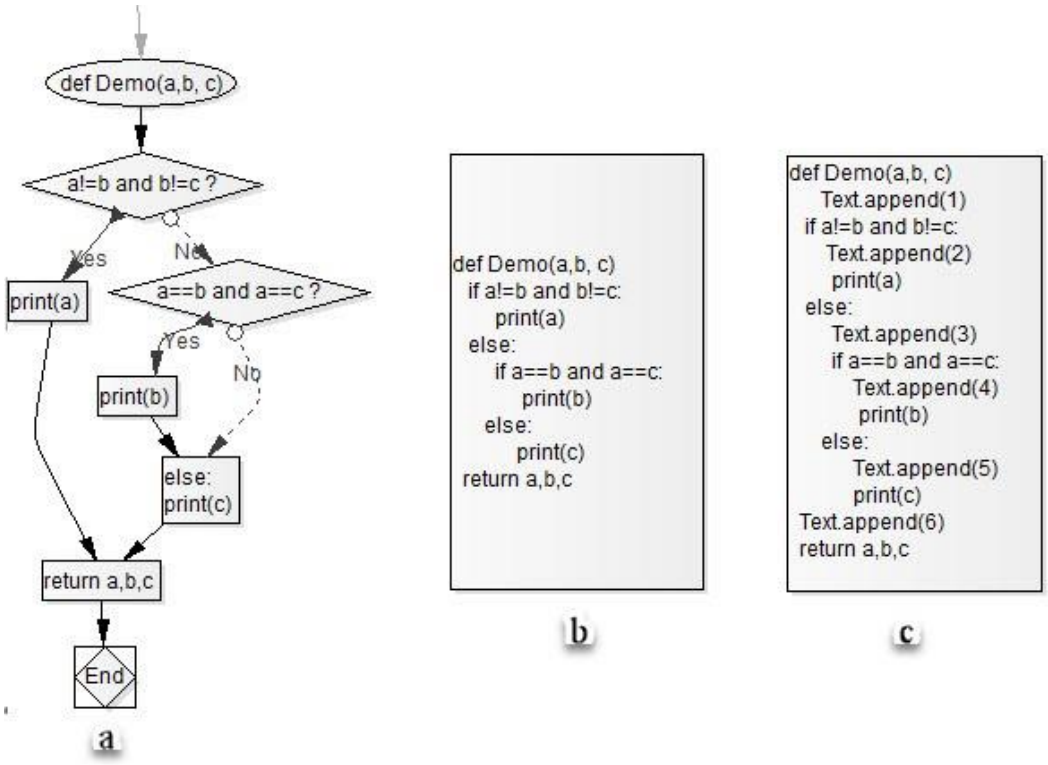

Figure 1: Corresponding CFG sample method, b: a sample source code c: instrumented method

3-2- Initial Population

The initial population of particles in the PSO algorithm can be considered the initial test data set. Each particle (or test data set) is initialized with a random set of values within the defined bounds of the search space [3]. The data structure considered here for

PSO-improved algorithms is a matrix structure, where the number of matrix columns equals the product of the number of program variables and their paths [4]. Suppose there are m execution paths, P_1 to P_m, to be covered by n test data, X_1 to X_n. Each test data, X_i, is a vector (x_{i_1}, x_{i_2}, ..., x_{i_m}), of m elements where each element is an input variable.

3-3- Fitness Function

The fitness function is a crucial component of search algorithms used to assess the quality of various solutions. The solution (test data) needs to be provided as input to the program under test, which is then executed, and its path coverage is measured. If the number of paths of the program under test is denoted as NP and the number of paths covered by the PSO algorithm is denoted as TP, in all implementation algorithms, the fitness of each solution (i) is calculated using Eq. (1):

$$F_i = \frac{TP}{NP} \tag{1}$$

3-4- Brief explanations of the Improved PSO algorithms

In this section, the simple PSO algorithm is first described according to the algorithm presented by Mao et al. [5]. Next, the improvements to the PSO algorithm are briefly outlined.

3-4-1- Basic Particle Swarm Optimization

In the PSO algorithm, which models the collective behavior of bird flocking, a swarm of particles changes their positions in the search space depending on their previous experience and the swarm's experience to find the global optimum. In general, the personal best position of particle *i* is denoted by *pbesti*, while the global best position of the entire population is called *gbest*. Suppose the population size is *s* in the *D*-dimensional search space; a particle represents a potential solution. The velocity and position of the d^{th} dimension of the *i*th particle can be updated by formulas (2) and (3) [5], respectively. In Figure 2, you can see the pseudocode of the basic PSO algorithm.

Algorithm. Simple PSO
Input: (1) the program under test *P*, and
The variable (a1, a2, a3, …., an) is the variable list of *P* ;
(2) structural coverage criterion target path ;

(3) algorithm parameter of PSO. *n*, *w*, c1, c2, and V_{max};
(4) the maximum evolution generation *max_gen.*
Output: test data set *TS* satisfying the target path.

```
01 encode input list variable into a m-dimension position vector;
02 instrument program P for gathering structural coverage information;
03 initialize the velocity vector V_i^d and position vector X_i^d
Test Suite Generation
04 while(Max_iteration)
06    for each particle i in the population with size n
07       for each dimension (1 ≤ d ≥m ) of particle i
08          Calculate the current velocity V_i^d
            of particle i in dimension d;
09          if V_i^d exceeds the boundary V_max then
10             adjust it within the boundary;
11       endif
12    calculate the current position X_i^d
13   endfor
14    decode vector X_i into a test case tc_i ∈ TS ;
15    execute the program with the test case tc_i and collect the
      coverage information to calculate the fitness f (X_i) ;
16    if f(X_i) > f (pbest_i) then
17       pbest_i = X_i
18   endif
19   if f(X_i) > f (gbest) then
          gbest=X_i
20   endif
21  end for
22 end while
23 return TS = (tc_i) ;
```

Figure 2: Algorithm simple Pso

We must encode the input list (a_1, a_2, a_3,…, a_n) into an m-dimensional position vector at the initialization stage. Path coverage is a criterion commonly applied in structural methods of test data generation. To calculate the fitness value of each particle (test case), we should instrument the program under test P to gather the coverage information about construct elements. On the other hand, some random values are utilized to initialize the velocity vector V_i^dand position vector X_i^d. In the algorithm body, the procedure between lines 07 and 14 is used to determine the current position X_i^d of particle i at different dimensions d. In line 14, each particle vector Xi in the population is decoded into a test case. Then, the fitness of each test case f (Xi) is evaluated. Based on the fitness value of each particle (test case), the personal best position, pbesti, and the global best position, best, can be updated (lines 16-20). The termination condition in line 05 controls the whole particle evolution process. For the testing problem, the termination condition

can be the following two cases: (1) all construct elements have been covered, or (2) the maximum evolution generation max_gen is reached

3-4-2- Improved PSO Algorithms

This section compares the ten most recent improved versions introduced by the particle swarm optimization algorithm for automatic test data generation. Particle swarm optimization (PSO) has been widely applied in various optimization fields due to its ease of implementation and high efficiency. However, it suffers from limitations, such as slow and premature convergence, when solving high-dimensional optimization problems. This paper attempts to address these critical issues. A_1 to A_{10} are, respectively, different versions of PSO-improved algorithms. In research A_1 [6], writers propose a novel PSO algorithm called Chaos Adaptive Particle Swarm Optimization (CAPSO), which adaptively adjusts the inertia weight parameter w and acceleration coefficients c_1, c_2, and introduces a controlling factor γ based on chaos theory to adaptively adjust the range of chaotic search. A_2 is the standard particle swarm algorithm.

In A_3 [7], a new method of parameter adjustment, known as piecewise nonlinear acceleration coefficients, is introduced to the simplified particle swarm optimization algorithm (SPSO). An improved algorithm, referred to as piecewise-nonlinear-acceleration-coefficients-based SPSO (P-SPSO), is then proposed. Then, a mean differential mutation strategy is developed for the update mechanism of P-SPSO, and another improved algorithm named mean-differential-mutation-strategy embedded P-SPSO (MP-SPSO) is proposed. A4 [8] introduced appropriate improvements to PSO and proposed a novel chaotic PSO variant with arc tangent acceleration coefficient (CPSO-AT). A5 [31] presents a new particle swarm optimization (EBPSO) algorithm. Firstly, based on an adaptive adjustment mechanism, the algorithm can select a more effective strategy from two search equations to balance exploration and exploitation abilities. Secondly, to utilize the information from individual historical optimal solutions and the optimal solution of the current population, a weight is introduced to adjust their influence in the search equation. Thirdly, by introducing population diversity, a dynamic equation for adjusting the algorithm's searchability is proposed. Finally, to avoid falling into a local optimum and to explore potential locations, a dynamic random search mechanism is proposed, which utilizes information from the current optimal solution.

In A6 [9], a constraint factor is introduced to control the velocity weight and reduce blindness in the search process. A dual-update (DU) strategy is based on new speed and position update strategies that are designed. Research A_7 [10] proposed a PSO algorithm with an adaptive two-population strategy (PSO-ATPS), which adaptively divides a

population into two groups representing excellent and ordinary populations. Inspired by animal hunting behavior, a new velocity–position update method is proposed for the general population. A velocity update formulation with decreasing inertia weights based on logistic chaotic mapping is applied to the excellent population. In A_8 [11], an improved particle swarm optimization (PSO) with adaptive weighted delay velocity (PSO-AWDV) is proposed.

A new scheme blending weighted delay velocity is first presented for a new PSO with a weighted delay velocity (PSO-WDV) algorithm. Then, to adaptively update the velocity inertia weight, an adaptive PSO-AWDV algorithm is developed based on the evolutionary state of the particle swarm evaluated via a new estimation method.

In A_9 [12], a hybrid HPSO-SSM algorithm is developed in which three significant improvements are made to the original PSO: First, a logistic map sequence is used to adjust the inertial weight ω, which provides sufficient variety and facilitates the avoidance of optimal solutions throughout the selection process. Second, a significantly improved update equation for creating the next-generation position is proposed, which can more effectively integrate exploration and exploitation. Third, a spiral-shaped mechanism (SSM) is coupled to the original PSO as a local search strategy for the known optimum solution. In A_{10} [13], a novel randomized particle swarm optimizer (RPSO) is proposed. The Gaussian white noise with adjustable intensity is utilized to randomly perturb the acceleration coefficients to explore the problem space more thoroughly.

4- Experiments and results

This study conducted two distinct experiments employing ten advanced Particle Swarm Optimization (PSO) algorithms (A_1 through A_{10}) to address five benchmark software problems. The first experiment focused on a comparative analysis of the meta-heuristics discussed in the paper, utilizing predetermined control parameter values. This comparison assessed the algorithms' performance based on the coverage metric and included a runtime analysis, emphasizing the importance of time efficiency in algorithm selection. Given that the fitness function directs the algorithms within the search space, the effectiveness of these functions was evaluated against the path coverage criterion. The second experiment conducted a comparative analysis of the success rates of algorithms derived from the first experiment, providing deeper insights into their performance and effectiveness.

4-1- Experimental setup

An architecture based on actual value execution was used for the experiments. It

consisted of a program analyzer, a path selector, and a test data generator. A program analyzer is provided with source code and generates a suitable representation (e.g., a control flow graph, a data dependence graph, or a program dependence graph) for subsequent analysis. Following the generation of test paths by the path selector, the test data generator generates path information for the specified test paths. The path selector utilizes this path information to regenerate the paths until a coverage criterion is met. The study utilized five benchmark problems, which comprised the following: triangle, quadratic equation, Max-Min number, leap year, and Fibonacci of marks. Triangle: The program checks whether a triangle can be formed using the given sides. If a triangle is formed, the program classifies the type of triangle as isosceles, equilateral, or scalene. Quadratic Equation: The program checks whether three input scans form a quadratic equation. If a quadratic equation is formed, then the roots of the equation are found. Max-Min: The program finds the largest and smallest numbers among input numbers. Leap: The program checks whether a given year is a leap year. Fibonacci: A Fibonacci program is a program that generates the Fibonacci sequence. The Fibonacci sequence is a series of numbers where each is the sum of the two preceding ones, starting from 0 to 1. In all of the problems, solutions were encoded using integer representation. Experiments were conducted on a platform equipped with an Intel(R) Core(TM) i7-6820HQ CPU at 2.70 GHz and 16 GB of RAM. All code fragments of the problems in Table 1 were implemented using the Python programming language.

Table 1: Properties of the benchmark problems used in the experiments

	Program title	Line count	Number of paths	CFG node size	No. of variables	References
1	Triangle	23	3	23	3	[14-16]
2	QuadEq	15	6	12	3	[15, 17]
3	Max-Min	31	25	21	5	[18]
4	Leap	7	6	8	3	[15]
5	Fibonacci	14	5	10	3	[17]
6	Bessj	49	30	32	2	[14]

4-2- Experiment 1: Comparison of Improved PSO Algorithms

The decision for the termination criteria is based on whether at least one test datum has traversed the target path or if the number of evolution iterations has reached the preset value. In either case, the evolution process will stop. Table 2 compares the results based on the coverage, evaluation, and time(s) criteria. Regarding the mean evaluations for the Quadratic Equation program, A_5 generates path-oriented data more efficiently than other

algorithms, and A4 has the worst mean evaluations among all algorithms. Regarding the search time for each approach, the mean A_5 has the least search time, while A_7 has the worst time.

Regarding the mean coverage for the Quadratic Equation program, A_3, A_5, and A_9 have achieved 100% coverage, while A10 has the lowest coverage. It shows that the paper performs best in this test code sample. Regarding the mean evaluations for the triangle classifier program, the paper for generating path-oriented data has the lowest evaluations compared to other Algorithms, and A2 has the worst mean evaluations among all algorithms. Regarding the mean coverage for the triangle classifier program, A_3, A_5, and A_9 have reached 100% coverage. And A_2 has the worst coverage. It shows that the paper performs best in this test code sample.

Regarding the mean evaluations for the Leap program, A_4 for generating path-oriented data has the lowest mean evaluations compared to other Algorithms, and A7 has the worst mean evaluations among all algorithms. Regarding the search time for each approach, the mean A2 has the least search time, while A_8 has the worst time. Regarding the mean coverage for the Leap program, A_1, with a coverage of 0.94, is the best, and A_7 has the worst coverage. It shows that the paper performs best in this test code sample. Regarding the mean evaluations for the Max-Min program, A_3 generates path-oriented data more efficiently than other algorithms, and A8 has the worst mean evaluations among all algorithms. For the search time of each approach, the mean A3 has the least search time, while A_9 has the worst time. In terms of the mean coverage for the Max-Min program, A_2, A_3, A_5, A_6, A_8, and A_9 have reached 100% coverage. And A_4 and A_7 have the worst coverage. It shows that the paper performs best in this test code sample.

Regarding the mean evaluations for the Fibonacci program, A_5 yields the lowest results among other algorithms for generating path-oriented data, and A3 has the worst mean evaluations among all algorithms. Regarding the search time for each approach, the mean A_3 and A_5 have the shortest search times, while A8 has the longest time. Regarding the mean coverage for the Fibonacci program, A_5 with 100% coverage is the most effective, while A_{10} has the worst coverage. It shows that the paper performs best in this test code sample. The experiments were repeated for the search space bounded by the range [-100, 100] to observe the algorithms' behavior in a constrained search space. We reported the mean and standard deviation in the first line, the median, and the rank of each algorithm in the second line of each cell. The number of evaluations and CPU times (in seconds) required to achieve maximum coverage were also reported in Table 2. The results produced by the algorithms were different versions of the PSO algorithm, ranging from A1 to A10, which were used, respectively. For a better understanding of Table 2, its structure is further explained in Figure 3.

	Program title	Coverage		Evaluation	Time (s)
Algorithms	Test program	Mean	standard deviation	Mean ± standard deviation	Mean ± standard deviation
		Median value	ranks	Median value	Median value

Figure 3: Structure of Table 2

Table 2: The results of the algorithms designed using a fitness function. Mean ∓ standard deviation are reported. Median values are given in parentheses, and ranks are presented next to the median.

	program title	Rang [-100, +100] Coverage		Evaluation	Time (s)		Rang [-100, +100] Coverage		Evaluation	Time (s)
A_1	QuadEq	0.635	0.150	20000 ± 0.0	0.610 ± 0.153	**A_6**	0.926	0.09705	15950.6 ± 4017.664	0.617 ± 0.134
		(0.5)	**3**	20000.0	0.54276		(1.0)	**2**	15630.0	0.623
	Triangle	0.62	0.34897	14080 ± 9252.425	0.437 ± 0.292		0.76	0.12649	17782 ± 5963.089	0.572 ± 0.192
		(0.5)	**4**	20000.0	0.59599		(0.7)	**4**	20000.0	0.614
	Leap	0.947	0.12669	10899.6 ± 6699.727	0.593 ± 0.382		0.91	0.14491	13224 ± 6307.299	0.793 ± 0.337
		(1.0)	**2**	9950.0	0.543		(1.0)	**3**	11920.0	0.793
	Max-Min	0.998	0.0199	2392 ± 3626.663	0.613 ± 0.982		1.0	0.0	3960 ± 3601.481	1.379 ± 1.290
		(1.0)	**1**	330.0	0.076		(1.0)	**1**	3190.0	1.121
	Fibonachi	0.561	0.083	20000 ± 0.0	0.624 ± 0.049		0.778	0.08478	19793.8 ± 1024.060	0.835 ± 0.06
		(0.6)	**5**	20000.0	0.611		(0.8)	**3**	20000.0	0.825
A_2	QuadEq	0.78	0.20986	15304 ± 6908.013	0.512 ± 0.234	**A_7**	0.92	0.10328	18628 ± 1621.883	0.405 ± 0.078
		(0.8)	**3**	20000.0	0.638		(1.0)	**1**	19150.0	0.403
	Triangle	0.5	0.21082	20000 ± 0.0	0.605 ± 0.096		0.76	0.12649	18864 ± 3536.486	0.393 ± 0.172
		(0.5)	**5**	20000	0.572		(0.7)	**4**	20000.0	0.335
	Leap	0.92	0.17512	8348 ± 7001.702	0.469 ± 0.441		0.8	0.18257	13372 ± 9011.692	0.498 ± 0.352
		(1.0)	**2**	4870.0	0.239		(0.7)	**2**	20000.0	0.679
	Max-Min	1.0	0.0	1982 ± 1527.130	0.537 ± 0.417		0.92	0.06324	17216 ± 5179.513	2.659 ± 0.967
		(1.0)	**1**	1850.0	0.529		(0.9)	**1**	20000.0	2.874
	Fibonacci	0.584	0.09289	19924.2 ± 758.0	1.366 ± 0.522		0.78	0.11353	19692 ± 973.981	0.439 ± 0.023
		(0.6)	**4**	20000.0	1.221		(0.8)	**3**	20000.0	0.447
A_3	QuadEq	1.0	0.0	680 ± 751.236	0.026 ± 0.032	A_8	0.984	0.05453	6179.4 ± 5711.668	2.208 ± 2.055
		(1.0)	**1**	470.0	0.016		(1.0)	**2**	3800.0	1.260
	Triangle	1.0	0.0	896 ± 803.923	0.063 ± 0.099		0.73	0.09487	18778 ± 3864.303	5.201 ± 1.071
		(1.0)	**1**	510.0	0.01402		(0.7)	**5**	20000.0	5.449
	Leap	0.911	0.17049	11260.4 ± 6880.619	0.583 ± 0.358		0.91	0.15472	11564 ± 6943.920	6.221 ± 3.796
		(1.0)	**3**	11240.0	0.557		(1.0)	**3**	11260.0	6.444

	Max-Min	1.0	0.0	585 ± 407.112	0.151 ± 0.105		1.0	0.0	1325.8 ± 1122.246	3.931 ± 3.475
		(1.0)	**1**	500.0	0.125		(1.0)	**1**	1010.0	2.883
	Fibonacci	0.994	0.03428	4222.4 ± 5122.860	0.152 ± 0.187		0.778	0.12679	18313.6 ± 4367.161	18.445 ± 99.342
		(1.0)	**2**	2270.0	0.076		(0.8)	**4**	20000.0	8.525
A_4	QuadEq	0.759	0.13714	2559.4 ± 2232.137	0.128 ± 0.117	A_9	1.0	0.0	974.6 ± 1121.560	0.071 ± 0.082
		(0.8)	**4**	1960.0	0.089		(1.0)	**1**	600.0	0.042
	Triangle	0.73	0.09487	8350 ± 2374.405	0.402 ± 0.159		1.0	0.0	722 ± 1493.346	0.034 ± 0.066
		(0.7)	**5**	8850.0	0.382		(1.0)	**1**	230.0	0.012
	Leap	0.834	0.18489	6815.6 ± 5913.631	0.521 ± 0.457		0.923	0.14829	10829 ± 7183.708	1.397 ± 0.932
		(1.0)	**3**	5200.0	0.394		(1.0)	**2**	10120.0	1.349
	Max-Min	0.92	0.04714	765.8 ± 701.780	0.289 ± 0.272		1.0	0.0	3116 ± 3056.596	8.658 ± 8.604
		(0.9)	**1**	550.0	0.200		(1.0)	**1**	2060.0	5.513
	Fibonacci	0.838	0.12615	13156.8 ± 4674.743	0.726 ± 0.259		0.812	0.17481	15287.8 ± 7030.108	1.696 ± 0.953
		(0.8)	**2**	12150.0	0.689		(0.8)	**3**	20000.0	1.922
A_5	QuadEq	1.0	0.0	303.8 ± 219.396	0.016 ± 0.011	**A_{10}**	0.664	0.18064	19157 ± 3061.330	0.998 ± 0.194
		(1.0)	**1**	260.0	0.013		(0.5)	**3**	20000.0	0.995
	Triangle	1.0	0.0	364 ± 263.447	0.016 ± 0.012		0.61	0.23309	18982 ± 3219.198	0.767 ± 0.206
		(1.0)	**1**	310.0	0.013		(0.7)	**4**	20000.0	0.760
	Leap	0.824	0.19904	14311 ± 6757.116	1.093 ± 0.522		0.881	0.17678	12282 ± 7529.841	0.962 ± 0.591
		(1.0)	**2**	18620.0	1.381		(1.0)	**2**	13050.0	1.03169
	Max-Min	1.0	0.0	2309 ± 2184.217	0.825 ± 0.790		0.998	0.019	3892.2±3990.593	1.672 ± 1.723
		(1.0)	**1**	1650.0	0.569		(1.0)	**1**	2220.0	0.953
	Fibonacci	1.0	0.0	2640.2 ± 2220.973	0.158 ± 0.135		0.522	0.09804	20000 ± 0.0	1.189 ± 0.078
		(1.0)	**1**	2100.0	0.119		(0.6)	**5**	20000.0	1.175

4-3- Experiment 2: comparison of success rate criteria

The next criterion for evaluating the efficiency of algorithms is the success rate. The success rate is calculated based on Eq. 4.

$$SR = \frac{\sum_i^{ps} bbci}{ps} \quad (2)$$

In Eq.4, ps is the number of executions of the algorithm, and bbc is a Boolean flag. If the algorithm reaches the maximum coverage from a fixed limit of calling the fitness function of the answer, this flag will be set to one. Otherwise, it will be similar to zero.

Figure 2 illustrates the success rates of ten Particle Swarm Optimization (PSO) algorithms (A1 to A10) across six test programs, highlighting their effectiveness in generating test data to achieve the desired path coverage. Among the algorithms, A5 stands out with a 100% success rate in three programs (Quadratic Equation, Triangle, and Fibonacci) and consistently high performance across all test scenarios, demonstrating its robustness and adaptability. Algorithms like A3 and A9 also show strong performance in specific programs, such as Max-Min and Fibonacci, but lack the consistent reliability of A5. Other algorithms, such as A1, A2, A4, A6, A7, A8, and A10, exhibit variable success rates, with some struggling in more complex programs like Fibonacci, indicating sensitivity to program complexity. A5's ability to maintain high success rates across both simple and complex programs underscores its superior balance of exploration and exploitation, as well as its innovative solution-updating strategies. This analysis emphasizes the significance of algorithmic enhancements in PSO for optimizing software testing processes, with A5 being the most effective and reliable option for automated test data generation. Future research could further enhance these algorithms to address even more complex testing challenges.

Figure 4: Success rate of comparison to other algorithms on the test programs

4-4- Experiment 3: Convergence Evaluation

Figure 3 presents the convergence behavior of the ten improved Particle Swarm Optimization (PSO) algorithms (A1 to A10) across six test programs. In automated software test data generation, the fitness function is a critical measure of an algorithm's ability to generate test cases that achieve the desired path coverage. The convergence behavior of the fitness function over iterations provides valuable insights into the efficiency, stability, and overall performance of different Particle Swarm Optimization (PSO) algorithms. Figure 3 illustrates the average fitness function values for ten improved PSO algorithms (A1 to A10) across six test programs. Each test program presents unique challenges, ranging from simple control flow structures to more complex decision-making processes, allowing for a comprehensive evaluation of the algorithms' capabilities.

The following analysis focuses on five key observations derived from Figure 3, which collectively highlight the strengths and weaknesses of the algorithms in terms of convergence speed, solution quality, stability, and adaptability to varying levels of program complexity. These observations are crucial for understanding how different algorithmic improvements impact the performance of PSO in the context of automated test data generation. By examining these aspects, we can identify which algorithms are most effective in achieving high path coverage while maintaining computational efficiency, ultimately guiding future research and practical applications in software testing.

- Convergence Speed

The convergence speed of the algorithms varies significantly across the test programs. Algorithms converging faster to a lower fitness value are generally more efficient, requiring fewer iterations to generate optimal or near-optimal test data. In most test programs, A5 demonstrates rapid convergence, often reaching the lowest fitness value earlier than the other algorithms. This indicates that A5 is highly effective in balancing exploration and exploitation, allowing it to navigate the search space and find optimal solutions quickly.

- Fitness Value at Convergence

The final fitness value at convergence is another important metric. Lower fitness values indicate better performance, corresponding to higher path coverage and more effective test data generation. A5 consistently achieves the lowest fitness values across all test programs, further confirming its superiority in terms of solution quality. This is particularly evident in programs like the Quadratic Equation and Triangle classifier, where A5 converges quickly and reaches the lowest possible fitness value.

- Stability and Robustness

Some algorithms exhibit fluctuations in fitness values during the iterations, indicating instability or difficulty in maintaining a consistent search direction. For example, A7 and A8 show more variability in their fitness values across different test programs, suggesting they may struggle with local optima or premature convergence. In contrast, A5 exhibits a more stable convergence pattern, characterized by smooth and consistent decreases in fitness values. This stability is a sign of robustness, as the algorithm is less likely to get trapped in local optima and can reliably find high-quality solutions.

- Comparison with Other Algorithms

While A5 stands out as the best-performing algorithm, other algorithms, such as A3 and A9, also demonstrate competitive performance in certain test programs. For instance, A3

performs well in the Max-Min and Fibonacci programs, achieving low fitness values with relatively fast convergence.

However, these algorithms do not consistently outperform A5 across all test programs. A10, for example, shows poor performance in several test programs, with slower convergence and higher final fitness values, indicating that it may not be as effective in navigating complex search spaces.

- Test Program Complexity

The complexity of the test programs also plays a role in the performance of the algorithms. Programs with more complex control flow graphs (e.g., Max-Min and Fibonacci) tend to challenge the algorithms more, resulting in slower convergence and higher fitness values for most algorithms.

Despite the increased complexity, A5 maintains its superior performance, suggesting that its adaptive mechanisms for balancing exploration and exploitation are particularly effective in handling complex search spaces.

The analysis highlights the superior performance of A5, which consistently achieves faster convergence, lower fitness values, and greater stability compared to the other algorithms. This makes A5 a highly effective choice for automated test data generation, particularly in scenarios where path coverage and efficiency are critical. The results also underscore the importance of algorithmic improvements in PSO, as they can significantly enhance the performance of meta-heuristic approaches in software testing applications.

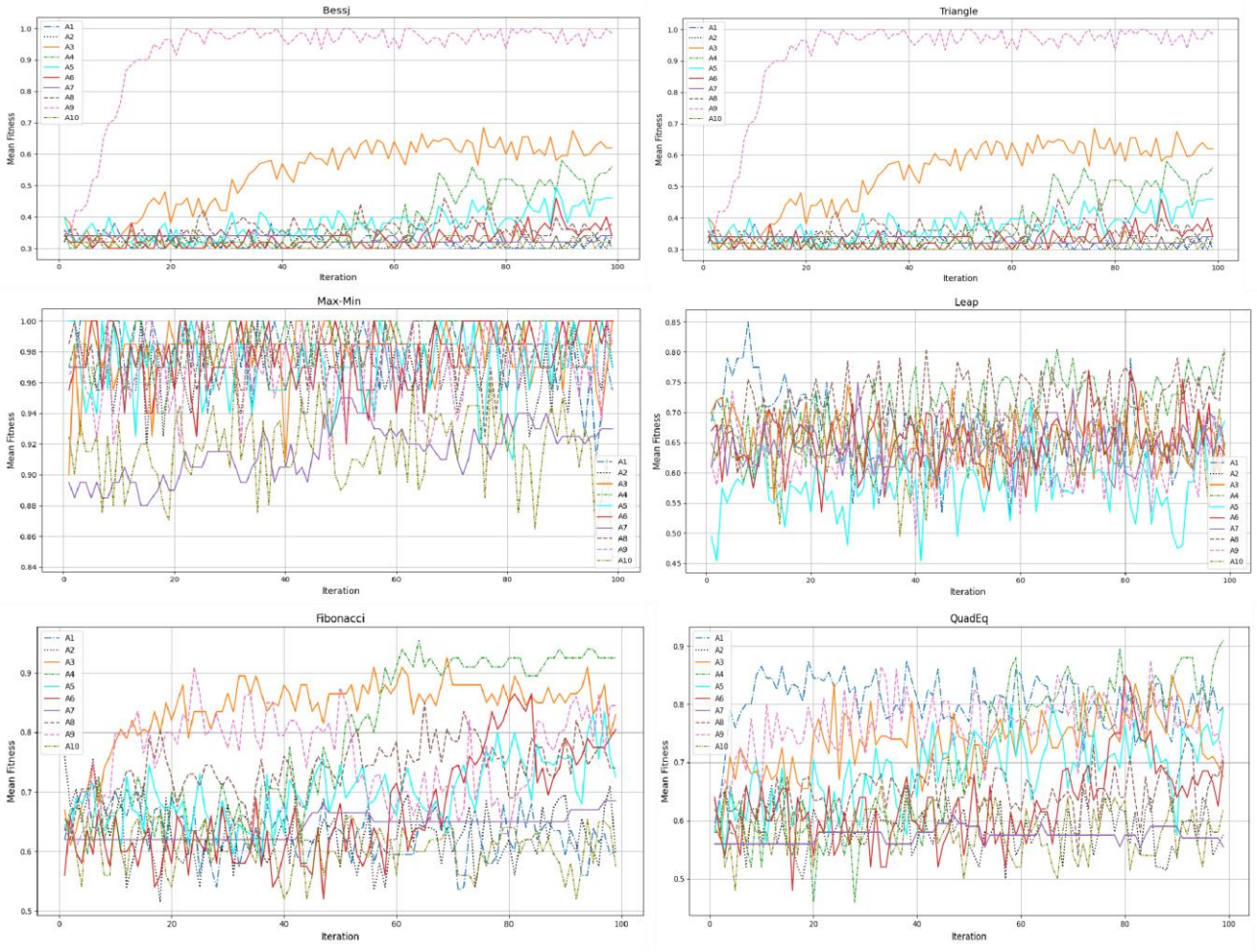

Figure 5: Mean Fitness function

5- Conclusion

This study has explored the latest advancements in Particle Swarm Optimization (PSO) algorithms for automated software test data generation, with a focus on achieving comprehensive path coverage in software systems. Through a comparative analysis of ten improved PSO variants (A1 to A10), the research evaluated their performance across multiple benchmark programs, considering key metrics such as coverage, runtime, and success rates. Among the algorithms tested, A5 emerged as the most effective, demonstrating superior performance in balancing exploration and exploitation, maintaining population diversity, and achieving efficient convergence. Its innovative solution-updating strategies, which adapt to the search space conditions, enabled it to consistently outperform other algorithms in terms of fitness evaluations, success rates, and coverage.

The experimental results underscore the efficacy of meta-heuristic approaches, particularly PSO, in navigating large and complex search spaces to generate high-quality test data. The success of A5 underscores the importance of algorithmic improvements in addressing common challenges, such as local optima entrapment and premature convergence, which are crucial for optimizing software testing processes. Furthermore, the study reaffirms the versatility of PSO and its variants in tackling diverse software testing challenges, making them valuable tools for enhancing the efficiency and effectiveness of automated test data generation.

Future research could extend this work by incorporating object-oriented programming constructs and multi-objective fitness functions to enhance the applicability of these algorithms further. Additionally, exploring parallelism in test data generation could improve scalability and reduce computational overhead, making these approaches more practical for large-scale software systems. Overall, this study contributes to the growing knowledge in search-based software testing, offering valuable insights into the potential of advanced PSO algorithms for optimizing software testing processes.

References

[1] X. Bao, Z. Xiong, N. Zhang, J. Qian, B. Wu, and W. Zhang, "Path-oriented test cases generation based adaptive genetic algorithm," *PloS one,* vol. 12, no. 11, p. e0187471, 2017.

[2] M. Esnaashari and A. H. Damia, "Automation of software test data generation using genetic algorithm and reinforcement learning," *Expert Systems with Applications*, vol. 183, p. 115446, 2021.

[3] 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.

[4] R. R. Sahoo and M. Ray, "PSO based test case generation for critical path using improved combined fitness function," *Journal of King Saud University-Computer and Information Sciences,* vol. 32, no. 4, pp. 479-490, 2020.

[5] C. Mao, X. Yu, and J. Chen, "Swarm intelligence-based test data generation for structural testing," in *2012 IEEE/ACIS 11th International Conference on Computer and Information Science*, 2012: IEEE, pp. 623-628.

[6] Y. Duan, N. Chen, L. Chang, Y. Ni, S. S. Kumar, and P. Zhang, "CAPSO: Chaos adaptive particle swarm optimization algorithm," *Ieee Access,* vol. 10, pp. 29393-29405, 2022.

[7] M. Lin, Z. Wang, F. Wang, and D. Chen, "Improved simplified particle swarm optimization based on piecewise nonlinear acceleration coefficients and mean differential mutation strategy," *IEEE Access,* vol. 8, pp. 92842-92860, 2020.

[8] Z. Ma, X. Yuan, S. Han, D. Sun, and Y. Ma, "Improved chaotic particle swarm optimization algorithm with more symmetric distribution for numerical function optimization," *Symmetry,* vol. 11, no. 7, p. 876, 2019.

[9] Y. Song, Y. Liu, H. Chen, and W. Deng, "A Multi-Strategy Adaptive Particle Swarm Optimization Algorithm for Solving Optimization Problem," *Electronics,* vol. 12, no. 3, p. 491, 2023.

[10] M. Zhao, H. Zhao, and M. Zhao, "Particle swarm optimization algorithm with adaptive two-population strategy," *IEEE Access,* 2023.

[11] L. Xu, B. Song, and M. Cao, "An improved particle swarm optimization algorithm with adaptive weighted delay velocity," *Systems Science & Control Engineering,* vol. 9, no. 1, pp. 188-197, 2021.

[12] Z. Ahmad, J. Li, and T. Mahmood, "Adaptive Hyperparameter Fine-Tuning for Boosting the Robustness and Quality of the Particle Swarm Optimization Algorithm for Nonlinear RBF Neural Network Modelling and Its Applications," *Mathematics,* vol. 11, no. 1, p. 242, 2023.

[13] W. Liu, Z. Wang, N. Zeng, Y. Yuan, F. E. Alsaadi, and X. Liu, "A novel randomised particle swarm optimizer," *International Journal of Machine Learning and Cybernetics,* vol. 12, pp. 529-540, 2021.

[14] Z. K. Aghdam and B. Arasteh, "An efficient method to generate test data for software structural testing using artificial bee colony optimization algorithm," *International Journal of Software Engineering and Knowledge Engineering,* vol. 27, no. 06, pp. 951-966, 2017.

[15] R. Malhotra, C. Anand, N. Jain, and A. Mittal, "Comparison of search based techniques for automated test data generation," *International Journal of Computer Applications,* vol. 95, no. 23, 2014.

[16] H. Sharifipour, M. Shakeri, and H. Haghighi, "Structural test data generation using a memetic ant colony optimization based on evolution strategies. Swarm Evol Comput 40: 76–91," ed, 2018.

[17] A. Damia, M. Esnaashari, and M. Parvizimosaed, "Software testing using an adaptive genetic algorithm," *Journal of AI and Data Mining,* vol. 9, no. 4, pp. 465-474, 2021.

[18] M. Khari, A. Sinha, E. Verdu, and R. G. Crespo, "Performance analysis of six meta-heuristic algorithms over automated test suite generation for path coverage-based optimization," *Soft Computing,* vol. 24, no. 12, pp. 9143-9160, 2020.