Front. Comput. Sci.
DOI 10.1007/s11704-014-3496-9

RESEARCH ARTICLE

A novel strategy for automatic test data generation using soft computing technique

Priyanka CHAWLA (✉)[1], Inderveer CHANA[1], Ajay RANA[2]

1 Computer Science and Engineering Department, Thapar University, Patiala 147004, India
2 Amity School of Engineering, Amity University, Noida 201301, India

Abstract Software testing is one of the most crucial and analytical aspect to assure that developed software meets prescribed quality standards. Software development process invests at least 50% of the total cost in software testing process. Optimum and efficacious test data design of software is an important and challenging activity due to the nonlinear structure of software. Moreover, test case type and scope determines the quality of test data. To address this issue, software testing tools should employ intelligence based soft computing techniques like particle swarm optimization (PSO) and genetic algorithm (GA) to generate smart and efficient test data automatically. This paper presents a hybrid PSO and GA based heuristic for automatic generation of test suites. In this paper, we described the design and implementation of the proposed strategy and evaluated our model by performing experiments with ten container classes from the Java standard library. We analyzed our algorithm statistically with test adequacy criterion as branch coverage. The performance adequacy criterion is taken as percentage coverage per unit time and percentage of faults detected by the generated test data. We have compared our work with the heuristic based upon GA, PSO, existing hybrid strategies based on GA and PSO and memetic algorithm. The results showed that the test case generation is efficient in our work.

Keywords software testing, particle swarm optimization, genetic algorithm, soft computing, test data generation

Received December 14, 2013; accepted September 21, 2014

E-mail: priyankamatrix@gmail.com

1 Introduction

Software testing forms an integral and indispensable part of software development life cycle. Software development process invests at least 50% of the total cost in software testing process [1]. Now, if the safety is an integral feature of the software the cost can go even higher [2]. To improve the quality of software, extensive manual testing is infeasible due to the huge requirement of time and cost. At present, there are lots of testing tools available in the market in which we can observe automation of test scripts execution implemented with the capture and playback mode feature of the tools. This again depends on the human intelligence and his involvement and hence does not perform software testing automatic completely. The selection of test cases is decided by software tester and success of testing process depends upon the expertise and intelligence of a tester. This can be fully automated if test data is generated automatically by the software testing tool. Thus, automatic test data generation is one of the potential areas of research in software testing. Software testing can be effectively automated to generate test data with soft computing strategies like genetic algorithm (GA) and particle swarm optimization (PSO). It works by transfiguring test data generation problem to optimization problem and test goal to objective functions. The objective functions forms the basis by which are solutions are compared and contrasted while carrying out overall search goal. The overall search is directed towards the search space of more fitter solutions possessing better objective function values. Keeping this point

in their view, many authors in the past have used intelligent search algorithms like PSO and GA to achieve the goal of automated generation of test data [3–9]. But by using these algorithms in isolation, authors faced the problems of locally stuck solutions.

Therefore authors started exploring the hybrid scheme of GA and PSO to gain the advantages of both the strategies like fast convergence rate, increased population diversity, ability to find the global optimum and solution stability [10–15].

But still there is limited application of soft computing techniques for automatic software test data generation in software industries. This may be due to the fact of complex and iterative nature of the existing strategies which requires huge cost in terms of computational resources and time. So there is need to develop the strategy that not only outperforms in terms of coverage and fault detection but also focus to minimize the cost in terms of computational resources requirement and time taken to solve the problem.

Based on these considerations, in this paper, we present a hybrid PSO and GAs based heuristic for automatic generation of test suites. The proposed algorithm is implemented in Java and it integrates the evolution ideas of both the soft-computing techniques (i.e., PSO and GA), in which crossover operator would be applied between particle personal best values and global best values. The mutation operator is applied to the global best particle and if it leads to better fitness function value, the global best particle is modified otherwise older gbest is retained. In this way, individual particles are enhanced. These enhanced parent individuals are capable of producing better offspring that can perform in better way than other offsprings. Additionally, the offspring which are poor in performance can easily be identified and can be removed from the population. As a result, we are able to achieve the following benefits:

i) Better run-time and space complexity.
ii) Able to generate test data for object oriented classes efficiently.
iii) Easier implementation.
iv) Only the best candidate solution are retained and evolved for other better solutions.
v) Better convergence speed, solution quality, ability to find the global optimum, and solution stability.

The primary contributions of our work are highlighted as below:

i) Test suite representation of the object oriented software consisting of series of object invocations, method calls and parameters combination using context free grammar.
ii) Novel test data generation strategy. The aim of the strategy is to generate test data which promises 100% branch coverage in minimum possible time.
iii) Experiments to investigate the effectiveness of our proposed approach. We conducted experiments and compared the results with our proposed strategy. The case study subjects taken are Java container classes.
iv) Fault seeding experiments to assess the quality of generated test data by our proposed strategy.
v) Empirical analysis of the existing hybrid and traditional strategies.

The paper is structured as follows: Section 2 introduces the PSO and GAs, Section 3 throws light on related work, Section 4 describes the proposed strategy in detail, Section 5 includes experimental evaluation, Section 6 throws light on threats on validity of the approach and Section 7 summarizes the paper and gives suggestions for future work.

2 Introduction to GA and PSO algorithm

This strategy embeds PSO with GA forming hybrid PSO strategy yielding superior results than either the GA or PSO can give when used in isolation. In this section, basic concepts of GAs and PSO are introduced, followed by a detailed introduction of our strategy in the next section.

GA is an evolved metaheuristic optimization technique that works with a set of promising solutions tracking optimization problem through encoding them on some data structures called as chromosomes. This technique copies principle of Darwian theory of biological evolution as it applies recombination and mutation operators on these chromosomes yielding one or many candidate solutions. A basic algorithm for GA is described in Algorithm 1 [16].

Algorithm 1 GA

```
input: Random population and appropriate fitness function
output: Optimized solution
Initialize(population);
Evaluate(population);
while not satisfied do
   Selection (population);
   Crossover (population);
   Mutate (population);
end
```

PSO utilizes swarm-based meta-heuristic search technique given by Eberhart and Kennedy in 1995 [17,18]. It mimics social norms of bird flocking, animals herding or fishes schooling. Like swarms searching for food in a unified way, PSO is also a set of random potential problem solutions, called particles. PSO strategy is based on the movement and exploration of virtual input search space by a particle. Particle can be observed as an object which consists of position and velocity as two components as shown in Eqs. (1) and (2) [19]. A basic algorithm for PSO is described in Algorithm 2 [19].

$$V_{j,d}(t) = wV_{j,d}(t-1) + cr_{j,d}(p\text{Best}_{j,d}(t-1) - X_{j,d}(t-1)) + cr_{j,d}(l\text{Best}_{j,d}(t-1) - X_{j,d}(t-1)). \quad (1)$$

$$X_{j,d}(t) = X_{j,d}(t-1) + V_{j,d}(t). \quad (2)$$

Algorithm 2 PSO

```
input: Random Population and appropriate fitness function
output: Optimized solution
for each Particle i do
    initPosition(i);
    initBestLocal(i);
    if i=1 then
        initBestGlobal( );
    end
    if improvedGlobal(i)==1 then
        updateBestGlobal(i);
        initVelocity(i);
    end
end
while not endingCondition do
    for each Particle i do
        createRnd(rp,rg);
        updateVelocity(i,rp,rg);
        updatePosition(i);
        if improvedLocal(i)==1 then
            updateBestLocal(i);
        end
        if improvedGlobal(i)==1 then
            updateBestGlobal(i);
        end
    end
end
```

3 Related work

Evolutionary structural testing is a search-based technique to automate corresponding unit test case generation. Although, various researchers have worked to prove its applicability in the area of software test data generation, but still its application is limited in the software industry [3, 5, 6, 20–23]. It is because of the complexity and huge computational cost associated with these techniques. Therefore, the strategy has been designed to reduce the complexity and cost of computation to a greater extent and also gets maximum benefits (like its intelligence) of the soft computing techniques. The search-based approaches employed for automated testing are very extensive. There are several metaheuristic search techniques like hill climbing, simulated annealing and evolutionary algorithms that have proved their elementary significance while automatic test data generations. For concision, we only emphasize here some of the closely related work. The most closely related works to ours are techniques based on genetic algorithm and particle swarm optimization that generate automatic test data for object oriented programs and structured programs. We summarized them in the comparison Table 1 and also illustrated them below:

Zhang et al. [13] combined the ideas of GA and PSO for automatic generation of test data. He took two population sets P_1 and P_2 and applied genetic operators and PSO calculation operations separately. Computation of fitness function values is carried out on population P_1 and particles are sorted based upon their fitness function values in population P_1 and best fitness particles are added to population P_2 at ratio phi. PSO operations are applied to population P_2 and updated fitness function of particles in population P_2 is computed. The particle transferred to P_2 and removed from P_1 and new random particles are added to P_1. The author took triangle classification problem as a case study and demonstrated four paths as test goal of his experiment and compared GA, PSO and GA-PSO independently on each trail 100 times and observed that GA-PSO took lesser number of iterations than GA and PSO. The average execution time of GA-PSO was found more than PSO but lesser than GA. Unlike our work, this technique generates tests that may be redundant and is more complicated as it applies GA after applying PSO and requires several transformations and exchanges in the different population sets. Moreover, the test adequacy criterion chosen is path coverage which consumes lot of computational cost and time. Furthermore, they did not assess fault detection capability of test data which is one of the essential requirements of quality test data. On the other hand, test adequacy criterion chosen by us is branch coverage per unit time and we have observed in our experiments that our approach is much simpler and efficient in terms of achieving branch coverage of the code of structured as well as object oriented programs.

Li et al. [10] also suggested the test data generation strategy based upon GA and PSO where he used PSO velocity and distance updates instead of mutation operator of genetic

Table 1 The comparison table of existing test data generation strategy

Sr. No.	Strategy	Technology used	Test adequacy criteria	Performance criterion	Case study	Input format
1	Windisch et al. [5]	PSO	Branch coverage	Fitness function evaluations Number of iterations	25 small artificial test objects and 13 complex industrial test objects	Instrumented source program in MATLAB GEATbx
2	Wegner et al. [6]	Evolutionary testing	Mean coverage	Mean number of test data	Six C programs	Source program in C
3	Li et al. [10]	Hybrid GA and PSO	Path coverage	Execution time	Triangle benchmark problems	C source code
4	Singla et al. [11]	Hybrid GA and PSO	dcu and dpu	Number of test cases and cover ratio percentage	Simple programs	MATLAB
5	Zhang et al. [13]	GA and PSO	Path coverage	Number of iterations and average execution time	Triangle classification problem	Instrumented source program in C
6	Kaur and Bhatt [15]	Hybrid GA and PSO	Fault coverage	Execution time and APFD	Simple programs	Java
7	Wappler and Schiefer-decker [28]	GA	Distance metrics	Branch coverage %	Seven Java classes	ECJ
8	eToc [30]	GA	Branch coverage	Test cases, number of fitness evaluation and time	Container classes	Instrumented source code
9	Arcuri and Yao [32]	GA and hill climbing	Branch coverage	Time and coverage %	Container classes	Java; Instrumented code
10	Nayak and Mohapatra [33]	GA and PSO	dcu%	dcu%	Small 14 FORTRAN programs	Instrumented code; MATLAB
11	Ahmed et al. [34]	Combinatorial testing using PSO	Interaction elements	Test size	IPOG, WHITCH, Tenny, TConfig and TVG	
12	Li and Zhang [35]	PSO	Path coverage	Iteration time; time consumption	Benchmark triangle problem & binary search program	Instrumented code in C
13	Fraser and Arcuri [36]	GA	Branch coverage	Branch coverage	Industrial project and five open source libraries	Bytecode; Java
14	Gong and Zhang [37]	GA	Path coverage	Fault detection and coverage	Bubble sort program Siemens suite, print_tokens, schedule, replace, tcas, space, flex	Instrumented program in C
15	Traditional GA	GA	Branch coverage	Branch coverage	Container classes	Instrumented source code in Java
16	Our proposed strategy	GA and PSO	Branch coverage per unit time	Time, No. of test cases, fault detection and coverage per unit time	Container classes	Bytecode or source code

algorithm. The chromosome representation used by him is a binary vector and divided the population into "*n*" population set based upon the length of the binary vector. Unlike our strategy, he experimented only with triangle benchmark problem with path coverage as test adequacy as criterion and his experimental results showed that his hybrid strategy is better than traditional GA. However, this cannot be generalized for every case study subjects.

Singla et al. [11] proposed test data generation strategy based upon genetic and particle swarm optimization in which authors used velocity and distance updates of particle swarm optimization to enhance the particles and genetic operations are applied after that. MATLAB was used for the implementation. This is somewhat similar to the approach proposed by Zhang et al. [13]. The fitness function is based upon "definition computational" use and "definition predicate use". He tested his strategy only on some simple programs.

Kaur and Bhatt [15] also proposed hybrid strategy based on GA and PSO in which author used only the mutation operator of GA to update the values of particles to solve prioritiza-

tion problem in regression testing. The author's strategy depends on randomly selected mutant which caused larger execution time to find optimal solutions. Also, the algorithm has been tested on less number of simple programs. Whereas, we tested our strategy with much complex programs and found efficient results in term of fault detection and branch coverage.

Wu et al. [12] suggested GAPSO hybrid strategy to overcome the optimization problem of both continuous and discrete parameters. The author has applied GA for discrete magnitude and PSO for continuous magnitude. They tested their approach by taking a quadratic equation and proved the effectiveness of their approach. In contrast to their approach, we utilized the hybrid strategy for solving test data generation problem and tested very complex programs including object oriented programs as well as procedural programs efficiently in very reasonable time.

Chen [14] amalgamated PSO with genetic operator and tested the new approach for multimodal functions taken from the black-box optimization benchmarking (BBOB) functions [24]. The author used crossover operations to the pbest particles to make it more explorative. He experimentally proved that this scheme is much more efficient than dividing the swarm into multi-swarm systems which periodically regroup in terms of being more exploitative and explorative. This is somewhat similar to our approach as we also applied crossover operation with the pbest particles and mutation operation by selecting gbest as mutant. Whereas Chen modified the velocity and distance update equations by adding crossover component to it. This crossover step sets the position of every particle to the midpoint of its pbest and a random pbest drawn without replacement from the population of pbest particles. In contrast to the author approach, we applied the hybrid approach to solve the optimal software test data generation problem and proved its effectiveness with the help of various experiments.

To the best of our knowledge no other authors have applied hybrid strategy of GA and PSO for test data generation. However, extensive work have been found that showed fruitful results by blending genetic and particle swarm optimization algorithm in the field like optimization for recurrent network design, optimization for multi-modal functions, class responsibility assignment problem in object oriented analysis and design, economic dispatch problem, optimal tuning of the controller parameter, document clustering, force method-based simultaneous analysis and design problems for frame structures etc., [25–27]. Furthermore, many authors have also advocated the hybrid approach and showed the effectiveness by testing it on some simpler benchmark functions, but limited work has been found that solves the problem of test data generation, so we have also investigated the test data generation strategies which utilizes only genetic and particle swarm algorithm for comparison with our proposed novel strategy.

Wapler and Schieferdecker [28] proposed the approach of evolutionary class testing. His approach is based on genetic programming system and utilized ECJ [29] to implement his approach. He also suggested fitness function and termed it as distance metrics in which he gave equal weightage to approach level and branch distance. Whereas in our research work we defined fitness function in which we normalized the branch distance to avoid its domination over approach level. This helped us to guide the search algorithm movement better and hence we achieved better coverage of more complicated programs in much less time. Furthermore, the authors have not mentioned about the computational cost of the generated test data.

Windisch et al. [5] conducted experiments with 25 small artificial test objects and 13 complex industrial test objects and proved that PSO is more efficient and effective than GA. Wegner et al. [6] compared evolutionary testing and random testing for six C programs and found mean coverage achieved by evolutionary testing much better than random testing (RT). He also conducted that although test data generated by RT was much higher than evolutionary testing but coverage achieved was not satisfactory.

Tonella [30] also tested container classes using conventional GA but made some changes to the original version of genetic algorithm to take care of the object oriented software testing. Author applied one-point crossover and in addition defined three types of mutation operators which were selected randomly. The author tested for container classes. Wang and Jeng [31] also compared hill climbing, GA and MA and shown MA the best algorithm but they performed experiments for procedural functions. Arcuri and Yao [32] proposed a metaheuristic based upon genetic algorithm and hill climbing known as memetic algorithm. Authors applied hill climbing technique on each generation to find local optimum. He has tested his strategy on container classes and has empirically proved their strategy better than random search, hill climbing, simulated annealing and strategy based upon conventional genetic algorithm. Nayak and Mohapatra [33] adopted MATLAB to simulate their strategy for test data generation using data flow testing and found PSO outperformed GA in def-use coverage %. She tested on small 14 FORTRAN programs and concluded that PSO required less number of generations to achieve the same def-use cover-

age %. Ahmed et al. [34] adopted T-way interaction testing using PSO and compared his tool with other strategy like IPOG, WHITCH, Tenny, TConfig and TVG and observed remarkable test suite minimization. He carried out 20 independent test suite runs and proved that PSTG produced test data of optimal size. Li and Zhang [35] employed PSO for all-path automatic generation of test data. He performed his experimental results for benchmark triangle problem & binary search program and did analysis on number of iterations and time consumption and shown that their strategy much efficient. Fraser and Prcuri [36] employed GA and took branch coverage as test criterion initially and later it was generalized for any rest criterion. He developed his tool in Java & performed byte code instrumentation. To evaluate the strategy, one industrial project and five open source libraries were chosen and observed smaller test suite generated. He compared whole test suite generation with single branch test case generation and observed small test suites generated. He repeated his experiments 100 times with different seeds. Gong and Zhang [37] proposed automatic generation of test data for both path coverage and fault detection using genetic algorithm. Authors tested their strategy for real world programs and also compared with random and evolutionary optimization techniques.

4 Proposed strategy

In this section, we describe the architecture adopted by us to adapt PSO and GA for the automatic generation of test suites for the unit testing of classes of given class under test (CUT). The framework is shown in Fig. 1. We divided our framework into four main modules as follows:

i) Instrumentor
ii) Optimizer
iii) Test executor
iv) JUnit file generator

Our proposed algorithm generates optimal test data for both the procedural programs and object oriented programs. The structure of test case for both the programs consists of input data, parameter values of methods and the assertions which verify the pass and failure status of a test. Figure 2 shows the format of test case of procedural program containing only set of parameter values of the procedure call. The test case format of object oriented program also consists of sequence of constructor and method calls in addition to the

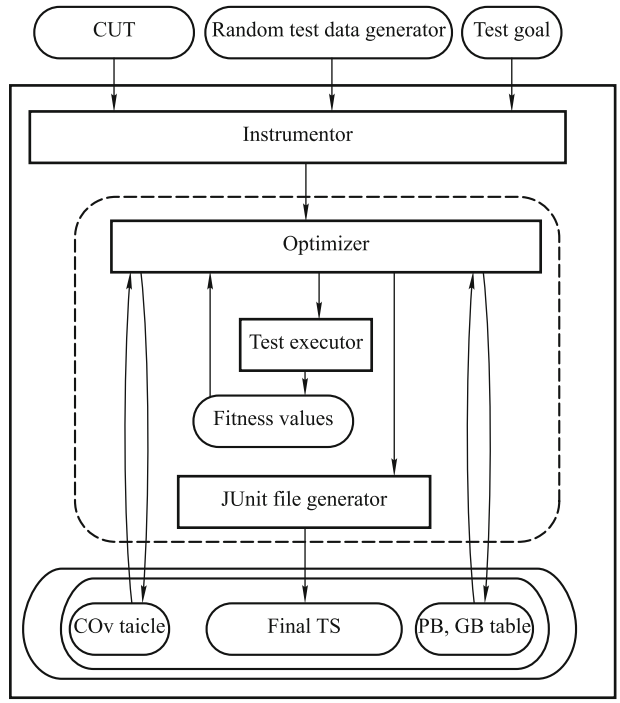

Fig. 1 Framework of novel strategy

above mentioned components. In object oriented paradigm, there may be multiple constructor invocations for object creation to test the class under test. In addition to this, the objects are also required to be put into special states to carry out the test scenario in the desired order. This is done with the aid of method calls on the required test objects. Figure 3 represents the structure of test case of object oriented program containing two constructor invocations and parameter value of the

```
float procedure(int x, float y)
{
   …                                  Test data=(5, 4.02)
}
```

Fig. 2 Test case format of procedural program

```
class F {…}
class U
{
    …
    // method under test
     public int add(F o1, int x)
           {                          Test data=(o2,o1,2)
              if( o1. ml( )==0)       where o2 is object of
                 {                    class U

                   …

                 }
           }
}
```

Fig. 3 Test case format of object oriented program

method. The first constructor invocation corresponds to the instance of the class under test and another constructor is required as a parameter for the creation of object under test. There is also an integer value required for method call.

4.1 Instrumentor

The instrumentation of CUT is essential for the evaluation of fitness function. The fitness function used in this work is based on branch coverage per unit time and branch distance function. The role of instrumentation is to modify the original executable image by inserting little instruction before every block to indicate that the block was executed. To evidence the branch distances, extra instruction is added before every predicate. These instructions are responsible for the computation of branch distance and notify the testing environment of it. The instrumentation can be carried out during runtime for java open source libraries. We used Jacaco open source tool to analyze the execution flow of the source code of CUT. Jacaco performs code coverage analysis using standard technology in Java VM based environment[1)]. It provides lightweight, flexible and well documented library for integration with various build and development tools. Jacaco can perform efficient on-the-fly instrumentation and analysis of applications even in the absence of source code.

4.2 Optimizer

It is the main component of our framework which is responsible for all the operations related to test data generation. Implementation has been carried out using the intelligence of GA and PSO algorithm. Retention of best particle positions of particle swarm optimization and genetic operators like mutation and crossover to evolve better solutions have been utilized. We have used genetic operators for the modification of the individuals in between the generations rather than using position and velocity update rules of PSO because these operators are more explorative. This was done to accommodate the object oriented structure of the container classes as it contains not only parameter values but also contains constructor and method invocations. This helped us to achieve the increased population diversity of GA and fast convergence rate of PSO by selecting only the best individuals as the new candidate solution in the upcoming generation. The loop is iterated until maximum number of iterations has been carried out or the optimal solution found with the best fitness values and coverage table has been formed as per the test goal received from the user.

Optimizer also instructs JUnit test data generator module to create JUnit file consisting of best individuals formed by the proposed algorithm. Algorithm 3 gives the macro steps of our proposed strategy. Particle representation and its corresponding test case format is a set of JUnit test cases for a given CUT as shown in Fig. 4.

Algorithm 3 Proposed strategy

```
input: Source code of the class
output: A JUnit class
Generate random population of "n" particles.;
Initialize CovTable with results of random test data generation.;
Instrument the SUT;
Create TestGoals;
for TG ← 1 to n do
    target=TG;
    while counter ⩽ maxIterations and target not satisfied do
        Evaluate fitness of each particle in the population based on
branch coverage analysis;
        Emit(Particle, Fitness);
        for particle ← 1 to n do
            if cpvector > pbest then
                /* Compare current value of particle with the previ-
ous best particle */
                pbest=cpvector;
                /* Set particle best with the current value of the vector
*/
            end
            if Cgbest > gbest the
                /* Compare current value of particle global best with
the previous global best */
                gbest=Cgbest;
                /* Set particle global best with current value of global
best */
            end
        end
        If all the individuals have been processed then write best indi-
viduals in PBest and GBest Tables;
        Apply crossover operation on each particle stored in PBest with
GBest;
        Apply mutation over Gbest;
        if its fitness value greater than GBest set it as new GBest;
        Execute the instrumented code for each particle to check if there
any other untested branches being tested;
        Update CovTable, TesGoal, TestSuite;
    end
end
```

At the first step, type of program is taken as input from the tester and the appropriate random test data is generated. After initialization, CUT is instrumented. In the process of instrumentation each condition is transformed into an expression which performs the same operations as by the original

[1)] Jacaco Web Page, http://www.eclemma.org/jacoco/

```
Particle representation
ob=B( ) a=C(5)
ob2=T(a) ob.insert(ob2):
d=D(5) ob.search(d)

Corresponding test case format
public void test( )
{
Bob=new B ( );
Ca=new C (5);
Tob2=new T (a);
ob.insert (ob2)
Dd=new D (5);
assert True (ob.search(d));
}
```

Fig. 4 The template of particle and test case format

program but also conveys the value taken by the reached condition. We have used Jacaco as the instrumentor and also explained it in our previous section. The set of test goals are also determined and is saved in a variable TG. The algorithm is iterated for each and every identified TG and it has been carried out maxIterations times. maxIterations denotes maximum allowable test data generation time and has been set by the tester. The randomly generated test cases are executed on the software under test with the intention to cover all the test goals. The fitness function is computed for each and every particle (i.e., test case). The pbest variables store the particles which have best fitness value when compared with its previous best fitness function values. The cpvector contains the current particle (i.e., solution) which is compared with previous best value. The better value (having better fitness function value) is stored in the variable pbest. In the similar fashion it occurs for Cgbest and gbest. The gbest variable stores the global best test case value (i.e., particle) having best fitness function value in whole population at a given iteration. All the individuals possessing better and best fitness function values are stored in the PBest and Gbest files. The test cases (or a particle) that succeeds to cover the targets are also saved in file named as FinalTestSuite. Crossover operation is applied between the gbest and pbest particles to uncover the new population individuals. Mutation is also applied to the newly formed particle to introduce the diversity into the population. This process is repeated until all the test goals are not covered or it exceeds the maximum limit of execution time of test case generation. The set of test cases saved in FinalTestSuite can be redundant which is minimized by applying the simple greedy algorithm by including the test cases which cover most of the test goals. CovTable will contain all the information about covered targets. The type of crossover and mutation operators used by us are explained below and shown in Fig. 5:

i) One-point crossover: Crossover operation acts on two particles (or test cases) by slicing particles at a randomly chosen point after the construction of target object and before the last method invocation. The two particles at the left can be transformed into the two particles at the right by using crossover operator. The cut point in the first particle is immediately after the call of f, while the cut point in the second particle is immediately before the call of g. The tails of the two cut particles are swapped, producing the results at the right. In our proposed algorithm we have implemented in such

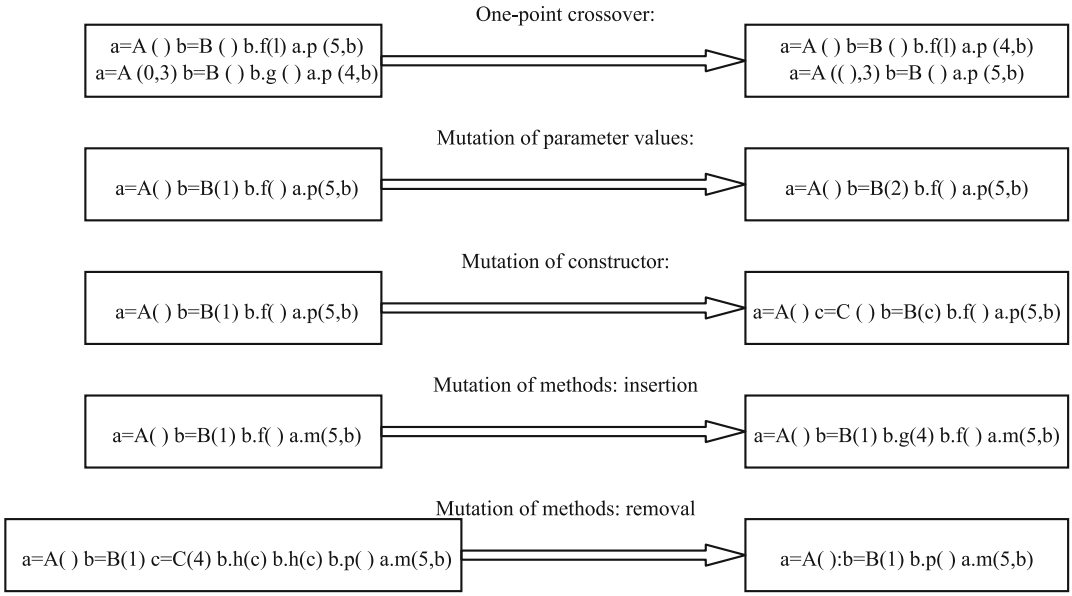

Fig. 5 Example demonstrating crossover and mutation

a way that only one particle having higher fitness function value is selected and the other one is dropped. It is shown with the example in Fig. 5.

ii) Mutation operator: Mutation operator acts on one particle (or test case). We have applied following type of mutation operators:

- Mutation of parameter values: A parameter value is replaced by a randomly generated value of the same type. In the example shown in Fig. 5, the parameter value passed to Class B is changed from 1 to 2.
- Mutation of constructor: Constructor types are changed in direction to improve the better fitness particle. The constructor type in the second particle is changed by another constructor from the same Class B having object as a parameter and the parameter value of the constructor is modified from integer value to an object of Class C as shown in Fig. 5.
- Mutation of methods: New method invocations are inserted or removed. Values or objects necessary as invocation parameters are also inserted or removed shown in the example in Fig. 5.

We have implemented our proposed hybrid approach using PSO and GA in Java programming language. The operative principle of the proposed approach as described above is also depicted through flowchart shown in Fig. 6. The objective function of test data generation optimization problem is computed by branch coverage achieved by test case and it is analogous to the quality or fitness of the associated solution.

4.2.1 Representation

The architecture used by us for particle is not just a combination of parameter values rather it is a sequence of constructor and method calls with the associated parameter values. Combination of parameter values can be used for procedural programs but not for object oriented software. The representation is shown below:

$$P \rightarrow \text{BV}$$
$$B \rightarrow \text{AB}$$
$$A \rightarrow c(V) \mid c \in \mid c.f(V)$$
$$V \rightarrow L, V$$
$$L \rightarrow N \mid \text{user} - \text{defined}$$
$$N \rightarrow \text{int} \mid \text{real} \mid \text{Boolean} \mid \text{String}$$

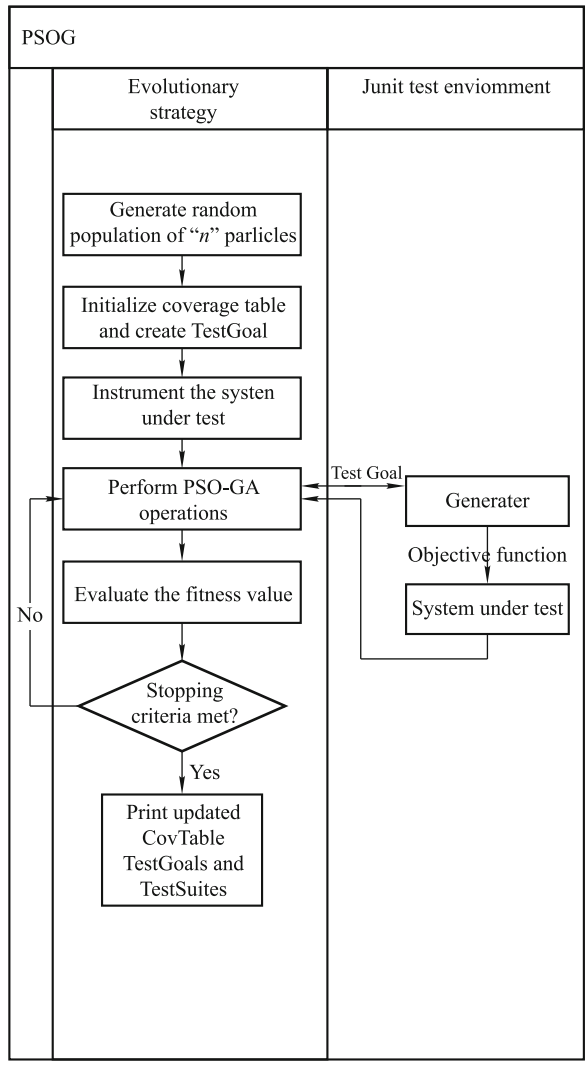

Fig. 6 Flowchart of the proposed strategy

Here P represents particle, B represents sequence of constructor and function instantiation represents actual parameters passed to the function, c represents class name and f represents function name.

4.3 Test executor

Test case executor component is responsible for fitness function evaluation by executing the CUT with the initial random test data generator and after that with the possible candidate produced by optimizer. Its main goal is to report with the best solution having optimum fitness function values which is then used by the optimizer to initiate JUnit file generator for the generation of JUnit file. Fitness function values are computed with the aid of control flow graph. The control flow graph of object oriented program is shown below in Fig. 7. The class control flow diagram shown in the above figure shows that upon the call of a method the control flow graph of a method

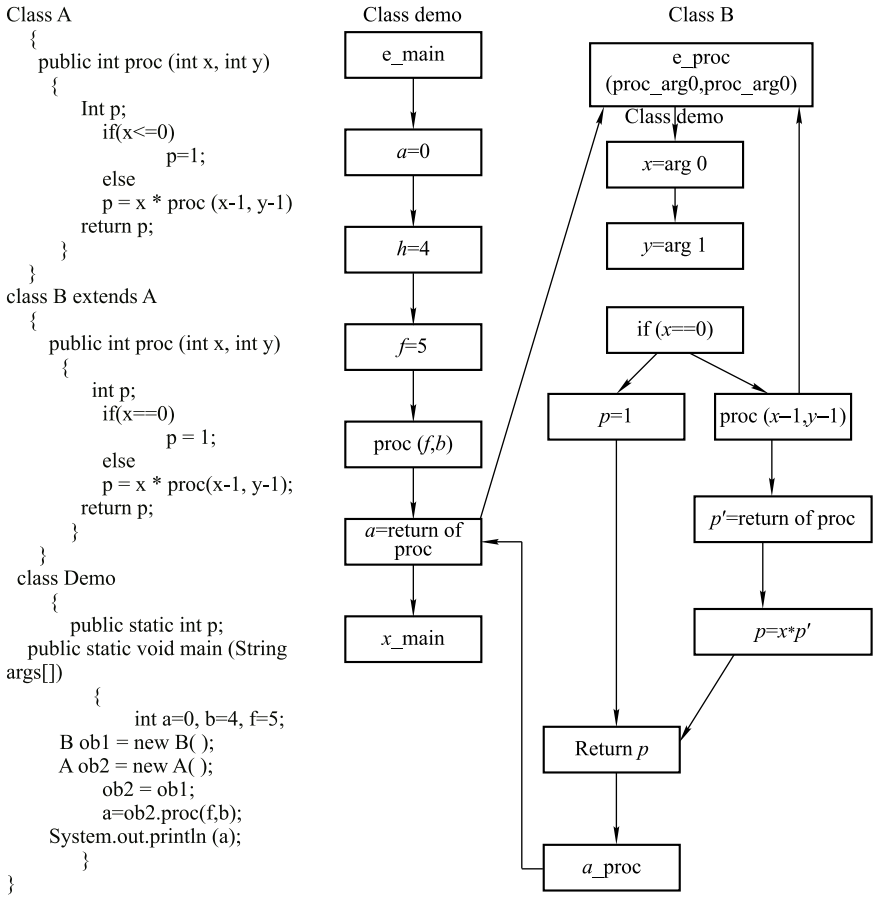

Fig. 7 The class control flow diagram

is created which is same as that of control flow graph of a method of procedural program.

The fitness function chosen by us in this research work is branch coverage. The fitness function is the ratio of edges covered during the execution of a test case to the total number of edges that leads to the given test goal or target in a control flow graph. Thus the fitness value would be close to 1 for the test cases which traverse most of the control edges, while it will be close to zero when the execution path does not coincide with the the edges leading to the test goal or target. We have also appended approximation level and branch distance to the fitness function in order to guide the test cases (or particles) towards the better optimal solution set.

4.3.1 Fitness function

Search based algorithms like GAs, PSO depend highly on the fitness function guidance to find out the optimal solution set. We have formulated multi-objective problem in which we have chosen branch coverage and computational cost as its constituent objectives. There is need to maximize branch coverage whereas cost should be minimized. The quality of branch coverage of a particle (or test case) is measured in terms of approximation level and branch distance.

$$\mathrm{BC} = A + \mathrm{normalize}(B'), \tag{3}$$

where BC is fitness function for branch coverage, A and B' represents approximation level and branch distance respectively. For a given path that does not cover the target branch, the approximation level can be defined as number of branching nodes that are in the way between nodes covered by the individual and the target node. Branch distance defines the extent of closeness of the input that was supposed to satisfy the condition of the last predicate but went wrong. The approximation level is required to point the search towards the target branch. Branch distance directs the exploration of the search algorithm towards optimal solution of the problem. It acts on the branch predicates and can be calculated with the application of the recursive rules after performing instrumentation of the software under test as defined in Table 2 [38]. For an instance, consider the control flow diagram shown in Fig. 8. The predicate values, branch distance and approximation level is shown in Table 3. The branch distance is normalized

in [0, 1] to shun the domination of branch distance over the approximation level. The normalization has been achieved with the following formula [39]:

$$\text{normalize}(B) = 1 - 1.001^{-B}, \tag{4}$$

$$F(\text{TS}_j) = \text{Coverage}(\text{TS}_j) + \text{BC}(\text{TS}_j),$$

$$\text{TS}_j = \{\text{TC}_1, \text{TC}_2, \ldots, \text{TC}_n\}, \tag{5}$$

where TC_i are test cases in a TS_j in a particular ordering; The aim is to find a suitable ordering of the sequence of test cases that achieves maximum coverage in minimal time. To decide if TS_k is better than TS_j, we can write it as

$$\left\{ \begin{array}{l} F(\text{TS}_k) > F(\text{TS}_j) \vee \\ F(\text{TS}_k) = F(\text{TS}_j) \wedge \text{Time}(\text{TS}_k) < \text{Time}(\text{TS}_j) \end{array} \right\}$$

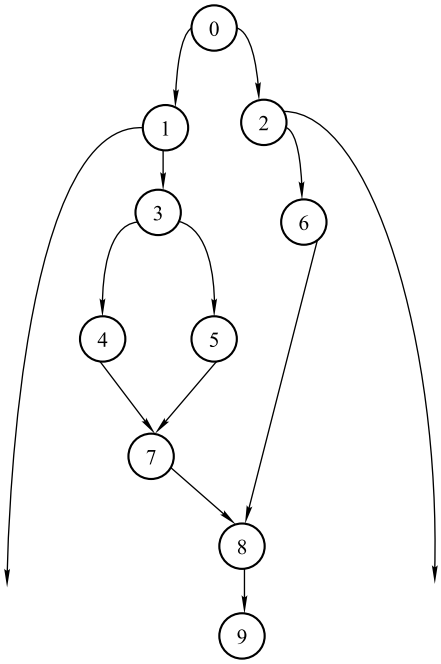

Fig. 8 Sample control flow graph

Table 2 Branch distance evaluation rules [38]

Element	Value
Boolean	If TRUE then 0 else K
$a = b$	If $\text{abs}(a - b) = 0$ then 0 else $\text{abs}(a - b) + K$
$a \neq b$	If $\text{abs}(a - b) \neq 0$ then 0 else K
$a < b$	If $a - b < 0$ then 0 else $(a - b) + K$
$a \leqslant b$	If $a - b \leqslant 0$ then 0 else $(a - b) + K$
$a > b$	If $b - a < 0$ then 0 else $(b - a) + K$
$a \geqslant b$	If $b - a \leqslant 0$ then 0 else $(b - a) + K$
$a \vee b$	min (cost (a), cost (b))
$a \wedge b$	cost (a) + cost (b)
$\neg a$	Negation is moved inwards and propagated over a

$F(S)$ consists of two components: coverage of a particular sequence which is to be maximized in minimal time and the second component is the sum of approximation level and branch distance which is to minimized and is only used to guide the search. Here we have also utilized the concept of test prioritization as defined by Rothermel et al. [40] to prioritize the minimal test suite which covers all the branches and detect maximum possible faults in the software under test. The test case prioritization problem is described as below:

Given: T, a test suite; PT, the set of permutations of T; f is a function from PT to the real numbers.

Problem: Find $R \in \text{PT}$ such that

$$\{\forall(R \in \text{PT})(R \neq T)[f(T) \geqslant f(R)]\},$$

where PT represents the set of all possible prioritization of T and f is a function that, applied to any such ordering, yields an award value for that ordering. Time taken can be computed using the method System.currentTimeMillis( ). Before executing a particular test suite on the given CUT we stored the current time in a variable startTime and after the execution of the CUT we have stored the current time into another variable endTime. Execution time can be found by subtracting startTime with endTime.

4.4 JUnit file generator

Its main goal is to produce JUnit file consisting of test cases formed by the optimizer and named as final test suite (FinalTS). JUnit file thus created can be executed using the tool JUnit. The tool JUnit can thus generate the detailed information regarding the pass and fail test cases.

5 Experimental evaluation

To evaluate the proposed strategy carefully and to provide the empirical evidence research questions should be designed. We have designed the research question stated below:

Does the proposed strategy improve the efficiency of test suite generation in terms of achieved code coverage per unit time?

5.1 Case study

We took ten Java classes from standard Java API 1.6, package Java.util as subject to carry out the experiments. Table 4 summarizes their features. The classes chose for testing covers varied range of lines of code i.e., from 100 to 1 600 lines of code. Moreover, these CUTs have varied internal complexity that can be seen from lines of code. Fitness function

Table 3 Branch distance and approximation level of CFG

Node	Predicate	Branch distance	Approximation level
0	$(a \leqslant 0 \vee b \leqslant 0 \vee c \leqslant 0)$	$\text{Min}((a-0),(b-0),(c-0))$	0
1	$(a > 1\,000 \vee b > 1\,000 \vee c > 1\,000)$	$\text{Min}((1\,000-a),(1\,000-b),(1\,000-c))$	1
2	$(c = a + b)$	$\text{Abs}(c-(a+b))$	2
3	$(a \leqslant b \wedge b \leqslant c \vee c \leqslant 2)$	$\text{Min}(((a-b)+(b-c)),(c-2))$	2
7	$(b \neg a + c)$	$-\text{Abs}(b-(a+c))$	2

evaluations, coverage and time taken for the generations of test suites have been obtained on an Intel Core i5-3210M CPU notebook having 2.5 GHz processor and 4 GB of RAM.

Table 4 Features of test object

Test object	LOC	No. of branches
TreeList	901	23
BitSet	606	156
Stack	136	20
StringTokenizer	195	55
Vector	1 019	100
LinkedList	708	84
BinaryHeap	334	61
TreeMap	1 636	191
BinomialHeap	335	79
BinaryTree	154	37

5.2 Comparing the search algorithm

We have compared our novel strategy with the test data generation technique based upon memetic algorithms [32], eToc [30], existing hybrid PSO and GA based strategy as proposed by Zhang et al. [13] and with the test data generation strategy based upon conventional genetic algorithms and PSO. eToc is open source and we downloaded it to perform the experiments on the container classes whereas we implemented the other four strategies as explained by the authors in their work in Java programming language. The reason for selecting the hybrid work of Zhang et al. [13] is that we were able to get all the details for implementing the strategy in their paper and other existing strategies are also somewhat similar. The empirical comparison of our proposed work is also necessary with conventional PSO based algorithm to prove its importance over simple PSO based algorithms.

The updating rules in the single PSO are only based upon velocity and distance updates which we found of no help while modifying the test suite representations involving object and method invocations. This approach is beneficial in case of parameter combinations only. Moreover PSO algorithms also face disadvantages like prematurity and decreased population diversity etc. We are able to achieve the following benefits of GA with the help of proposed strategy over single PSO:

i) We are able to achieve fast convergence rate, reduced complexity and avoid prematurity as compared to other local optimization techniques like hill climbing etc. in lesser time.

ii) We found that the application of genetic operators for the modification of the individuals representing string of test calls consisting of object invocations, method and parameter combinations in between the generations is much easier and fruitful rather than utilizing position and velocity update rules of PSO. It is much easier to apply crossover and mutation operation on the string of test calls consisting of object, method and parameter combination rather than computing position and velocity update of PSO method.

iii) We are able to achieve the increased population diversity because of using genetic operators of genetic algorithms.

iv) GA aided us in avoiding local optimum solutions and increased the stability of our approach.

The above claims can be easily seen in the Table 6. We have chosen inertia $w = 0.1$ and $c_1 r_1 = c_2 r_2 = 0.2$ for simplicity. For implementing PSO velocity and distance update rules we have taken the sequence of path covered by a test suite in the control flow graph as the distance vector and correspondingly velocity vector is updated. The test suite representation has been taken same as that of our proposed strategy. In the table is clearly shown that PSO based algorithm are better than conventional genetic algorithm in terms of time taken and coverage of the code but were found at a lower end when compared with hybrid strategies.

When we compare our proposed approach with the other existing hybrid approaches, we found and proved experimentally that our approach much easier and simpler as shown in Table 6. The existing hybrid PSO and GA based approaches are entirely different from our approach. We have used genetic operators for the modification of the individuals in between the generations rather than using position and velocity update rules of PSO. This was done to accommodate the object oriented structure of the container classes as it consist

not only parameter values but also contains constructor and method invocations. Pbest and gbest are evaluated from the initial set of randomly generated population using the PSO algorithm as explained in our proposed algorithm. Crossover operation is applied on each particles which is further enhanced using mutation operation. The gbest particle is mutated with the entire pbest particle set to generate another set of candidate solution. This process is repeated until best optimal solution is found.

Arcuri and Yao [32] proposed a metaheuristic based upon GA and hill climbing known as memetic algorithm. Authors applied hill climbing technique on each generation to find local optimum. Since application of hill climbing is comparatively costly so they kept population size and number of generation at lower side. He has tested his strategy on container classes and has empirically proved their strategy better than random search, hill climbing, simulated annealing and strategy based upon conventional genetic algorithm. This is the reason we selected memetic algorithm as one of the most important strategy for comparisons. Tonella [30] also tested container classes using conventional GA but made some changes to the original version of genetic algorithm to take care of the object oriented software testing.

Table 5 Parameters settings

Parameter setting	MEM	PSOG	GA
Population size	10	50	70
Crossover probability	0.9	0.2	0.2
Mutation probability	–	0.9	0.9
Rank selection bias	1.5	–	1.5

Table 5 shows the parameter settings adopted by us in case of MEM, PSOG and traditional genetic based algorithm. Statistical and practical differences have been computed using two-tailed Mann-Whitney U-test as per the guidelines in [41, 42]. We have also used Vargha and Delaney's $\hat{A}_{12}$ statistics, as standardized effect size [43, 44] in conjunction with null hypothesis significance testing (NHST). An effect is an objective and standardized measure of the magnitude of observed effect. Null hypothesis for our statistical test states that there is no difference between novel strategy and the existing strategy. Alternative hypothesis states novel strategy is better than the existing strategy. We performed pair wise comparisons between the novel strategy and all the other existing strategies. For the comparisons we decided to choose percentage coverage achieved per unit execution time (C) as a criterion. As coverage should always be maximized whereas execution time should be minimized, so percentage coverage achieved per unit execution time should be maximized for a better strategy. We executed all the strategies 30 times to gather sufficient information on the probability distribution of C, performance of all the compared strategies is shown in Table 6. The level of significance is set to 0.05. Vargha and Delaney's $\hat{A}_{12}$ statistics measures the probability that executing strategy X yields superior C values than executing another strategy Y. It can be computed easily as per the following formula [41]:

$$\hat{A}_{12} = (R_1/p - (p+1)/2)/q, \tag{6}$$

where R_1 is the rank sum of the first data group we are comparing. In the above formula, p is the number of observations in the first data sample, whereas q is the number of observations in the second data sample. Results in Table 7 exhibits positive results with high statistical confidence which certainly answers RQ. Our proposed novel strategy is better than all other strategies except MEM in some few cases. Our novel strategy PSOG is 100% of the time better than MEM in case of BinaryHeap, TreeList, BitSet and Vector. The performance of PSOG in case of TreeMap and that of StringTokenizer and LinkedList is 80% and 76% respectively of the time better than MEM. This is due to the following reason:

i) Our strategy intelligently retained only those particles that possess better fitness function values resulting from crossover and mutation operators. The other particles were rejected and not included in the population. Our proposed strategy is able to generate particles with best fitness function values in much less iterations of GA and PSO algorithm. This resulted to fastest convergence rate. On the other hand, the memetic algorithm is quite complex as it utilized hill climbing metaheuristic for local optimization is quite complex as it involves much compute intensive operations. Furthermore, by using the combination of genetic operators to the traditional PSO algorithm it resulted to be more exploitative and explorative. Whereas the elitism rate and population set of memetic algorithm are set to one and ten respectively.

ii) The test cases were prioritized and minimized effectively. The redundant test cases were deleted. Thus generated test suites were compact and took less time while execution.

The performance of PSOG has not shown remarkable results in case of Hashtable, Stack and BinaryTree. MEM is equivalent or slightly better than our strategy in case of Stack, BinaryTree and Hashtable. This may be due to the fact that in these classes there is comparatively lesser number of branches and hence the effectiveness of our strategy could not produce remarkable difference.

Table 6 Comparison of the different optimization algorithms on the container cluster. Each algorithm has been stopped after evaluating up to 10 000 solutions. The reported values are calculated on 30 runs of the test.

Case study	Strategy	Minimum	Median	Maximum	Mean	Std. Deviation	Std. Error
BinaryHeap	GA	0.830 4	0.851 9	0.862 4	0.846 9	0.011 28	0.002 06
	ETOC	1.179	1.192	1.22	1.197	0.017 2	0.003 139
	MEM	4.963	5.222	5.588	5.258	0.260 8	0.047 62
	PSOG	5.765	6.094	6.6	6.194	0.340 1	0.062 1
	PSO	0.886 8	0.896 2	0.904 8	0.897 5	0.006 453	0.001 178
	GAPSO	1.220	1.227	1.240	1.227	0.007 303	0.001 333
TreeList	GA	0.502 9	0.514 5	0.524 4	0.513 9	0.008 948	0.001 634
	ETOC	0.857 1	0.942 7	0.997 7	0.935 4	0.040 88	0.007 464
	MEM	5.5	5.765	6.188	5.844	0.260 2	0.047 51
	PSOG	6.667	7.071	7.654	7.131	0.412 1	0.075 24
	PSO	0.599 3	0.605 0	0.613 3	0.605 8	0.004 685	0.000 855 4
	GAPSO	0.956 7	0.963 6	0.985 3	0.968 5	0.012 39	0.002 262
Bitset	GA	0.450 1	0.455 9	0.467 3	0.457 7	0.007 263	0.001 326
	ETOC	0.562 1	0.564 7	0.565 5	0.564 1	0.001 455	0.000 266
	MEM	1.153	1.193	1.207	1.184	0.023 38	0.004 268
	PSOG	1.269	1.282	1.286	1.279	0.007 188	0.001 312
	PSO	0.520 0	0.529 5	0.533 8	0.527 8	0.005 862	0.001 070
	GAPSO	0.690 1	0.698 6	0.707 1	0.698 4	0.006 191	0.001 130
Vector	GA	3.043	3.197	3.41	3.217	0.153	0.027 93
	ETOC	5.389	6.188	6.533	6.037	0.487 4	0.088 99
	MEM	11	12.38	12.5	11.86	0.705 3	0.128 8
	PSOG	14.29	19.8	20	18.3	1.815	0.331 3
	PSO	3.770	3.881	4.133	3.913	0.146 2	0.026 70
	GAPSO	4.600	4.947	5.278	4.943	0.237 8	0.043 42
StringTokenizer	GA	4.261	4.543	4.796	4.56	0.184 5	0.033 69
	ETOC	12.13	12.38	16.5	13.2	1.433	0.261 6
	MEM	16.33	19.8	33	22.31	6.242	1.14
	PSOG	19.8	24.88	33.33	25.76	4.99	0.911 1
	PSO	4.335	4.605	4.861	4.639	0.186 2	0.034 00
	GAPSO	9.700	10.34	11.00	10.34	0.567 6	0.103 6
TreeMap	GA	5.326	5.683	5.755	5.694	0.077 73	0.014 19
	ETOC	12.13	12.38	16.5	13.2	1.433	0.261 6
	MEM	24.5	32.67	33	29.05	4.181	0.763 3
	PSOG	19.8	33.08	33.33	31.07	3.971	0.725
	PSO	6.555	6.640	7.193	6.804	0.276 5	0.050 49
	GAPSO	10.78	11.00	12.25	11.30	0.632 3	0.115 4
Hashtable	GA	4.261	4.543	4.796	4.56	0.184 5	0.033 69
	ETOC	12.13	12.38	16.5	13.2	1.433	0.261 6
	MEM	33.17	49.75	50	46.55	6.765	1.235
	PSOG	19.8	49.5	50	39.74	12.63	2.305
	PSO	4.300	4.619	4.901	4.644	0.187 7	0.034 27
	GAPSO	13.86	16.33	19.80	16.48	2.013	0.367 5
Linked List	GA	4.261	4.543	4.796	4.56	0.184 5	0.033 69
	ETOC	12.13	12.38	16.5	13.2	1.433	0.261 6
	MEM	33	50	49.75	46.51	6.762	1.235
	PSOG	33	99	100	76.97	28.77	5.252
	PSO	4.400	4.684	4.944	4.722	0.190 5	0.034 78
	GAPSO	13.86	16.17	19.80	15.83	1.849	0.337 5
Stack	GA	6.082	6.16	7.108	6.318	0.303 3	0.055 37
	ETOC	25	25	33.33	27.22	3.748	0.684 3
	MEM	33.33	33.33	50	41.11	8.457	1.544
	PSOG	25	33.33	50	40	9.129	1.667
	PSO	7.750	8.165	8.591	8.166	0.368 0	0.067 18
	GAPSO	16.67	20.00	20.00	19.11	1.499	0.273 7
BinaryTree	GA	6.373	6.52	7.523	6.667	0.326 1	0.059 54
	ETOC	25	25	33.33	27.22	3.748	0.684 3
	MEM	33.33	33.33	50	41.11	8.457	1.544
	PSOG	25	33.33	50	40	9.129	1.667
	PSO	16.67	16.67	20.00	18.00	1.661	0.303 2
	GAPSO	25.00	25.00	33.33	27.22	3.748	0.684 3

Table 7 Comparison of percentage coverage per unit time obtained by the novel strategy PSOG and by memetic algorithm MEM. Significant values of Vargha and Delaney's effect size ($\hat{A}_{12}$) are shown in bold.

CUT	Strategy	Sum of ranks	$\hat{A}_{12}$
BinaryHeap	MEM	465	**1**
	PSOG	1 365	
TreeList	MEM	465	**1**
	PSOG	1 365	
Bitset	MEM	465	**1**
	PSOG	1 365	
Vector	MEM	465	**1**
	PSOG	1 365	
StringTokenizer	MEM	675	**0.766 67**
	PSOG	1 155	
TreeMap	MEM	639.5	**0.806 67**
	PSOG	1 191	
Hashtable	MEM	1 112	0.281 67
	PSOG	718.5	
LinkedList	MEM	675.5	**0.765 56**
	PSOG	1 154	
Stack	MEM	946	0.465 56
	PSOG	884	
BinaryTree	MEM	946	0.465 56
	PSOG	884	

Figure 9 shows the Box Plots showing the comparison of all the strategies w.r.t CUT which clearly presents the performance of novel strategy better than others.

We also performed fault seeding experiments with the aid of Jaca[2)], a fault injecting tool to assess the quality of test data generated by our proposed strategy. We injected ten faults to simulate classic programming errors like use of wrong relational operator, wrong parameter value and condition missing etc. Out of our ten case study subjects test data of seven case studies were able to detect 100% faults injected in the code, which is highly optimal. In the case of other three case study subjects the assertion failures was found out to be 90%. It has been shown in Table 8. This was mainly due to the lack of required data flows associated with some attributes of an object.

Table 8 Experimental results of fault seeding

Case Study subjects	Assertion failures
BinaryHeap	10/10
TreeList	10/10
BitSet	9/10
Vector	10/10
StringTokenizer	9/10
TreeMap	9/10
Hashtable	10/10
LinkedList	10/10
Stack	10/10
BinaryTree	10/10

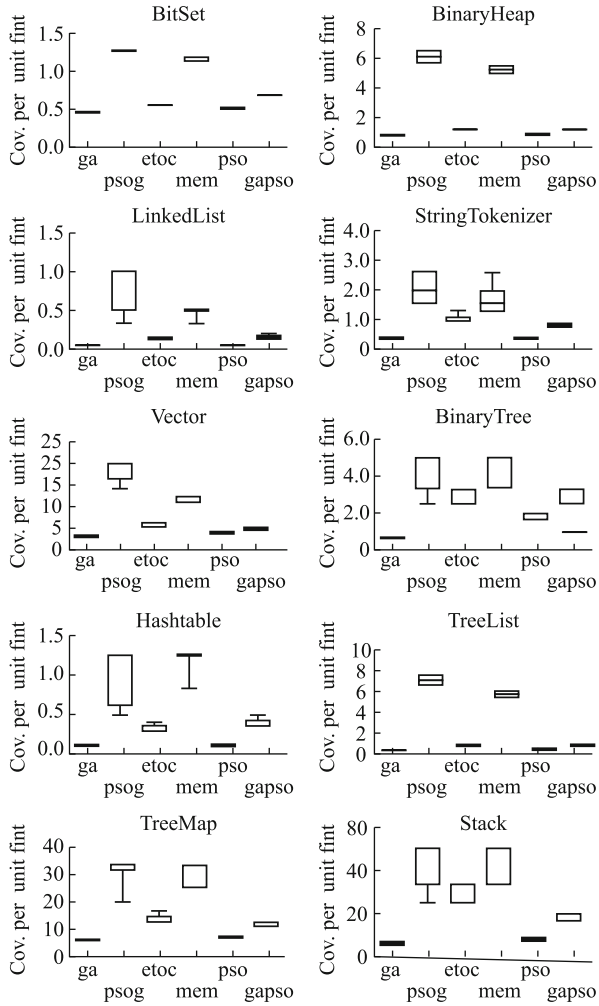

Fig. 9 Boxplots percentage coverage per unit time for all the container classes

6 Threats to validity

This paper presents the hybrid strategy for automatic test data generation based upon PSO and GA and also contrasted it with other test data generation strategies. Threats to construct validity are on how the efficacy of the testing strategy is measured. We measured the efficiency in terms of branch coverage per unit time and percentage of fault detected. However, we have tried to implement the strategy which works for other test adequacy criterion also but it has not been tested. Threats to internal validity may arise from how the experimental study was carried out. To minimize the probability of having faults in our proposed testing framework, it has been carefully tested. Furthermore, because our strategy is based

[2)] Jaca Web Page, http://www.ic.unicamp.br/~eliane/JACA.html

upon randomized algorithms, we repeated each experiment 30 times on each class and for the comparison of algorithms we adopted meticulous statistical process to assess the results. There is also the threat to external validity regarding the generalization to other types of software, which is common for any empirical analysis. Because of the large number of experiments required, we only used ten classes for our evaluation. To make claims on generalization we will need to conduct further studies on representative sets of classes.

7 Conclusions and future work

This paper proposed automated software test data generation for procedural and object oriented programs. The objective of minimizing the time and maximizing the branch coverage and fault detection has been addressed. Soft computing techniques like GA, PSO, MA and hybrid strategies based on genetic and particle swarm optimization algorithm has been implemented and compared with the proposed strategy. The empirical results showed that proposed strategy usually performs better than the other algorithms. Although our proposed strategy performs well on all the tested classes for all other strategies under evaluation but the results on Hashtable, Stack and BinaryTree were almost equivalent to that of memetic algorithm. We have tested proposed strategy on ten Java classes and the results have been very encouraging, but extensive research and testing should be done before any conclusions can be drawn about its effectiveness for generic object oriented software. This would inspire us to investigate new algorithms (like bacterial foraging and cuckoo search based testing) and new fitness functions like definition computational use and definition predicate use to improve performance in the future. Furthermore, additional fault seeding experiments are required to evaluate the quality of test suites generated by the proposed strategy. Future work would be concentrated to find out the possibility of parallelization of the proposed strategy. The feasibility of using the search heuristic with cloud computing for the test data generation of complex programs consisting of multiple classes would be studied and carried out. Main emphasis would be given to decrease the computation time and the cost to produce optimal test suites.

References

1. Ramler R, Wolfmaier K. Economic perspectives in test automation: balancing automated and manual testing with opportunity cost. In: Proceedings of the International Workshop on Automation of Software Test. 2006, 85–91
2. Grottke M, Trivedi K S. 2007. Fighting bugs: remove, retry, replicate, and rejuvenate. IEEE Computer, 2007, 40(2): 107–109
3. Pargas R P, Harrold M J, Peck R R. Test-data generation using genetic algorithms. Software Testing Verification Reliability, 1999, 9(4): 263–282
4. Chen X, Gu Q, Qi J X, Chen D X. Applying particle swarm optimization to pairwise testing. In: Proceedings of 34th Annual IEEE Computer Software and Applications Conference, COMPSAC'10. 2010, 107–116
5. Windisch A, Wappler S, Wegener J. Applying particle swarm optimization to software testing. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, GECCO '07. 2007, 1121–1128
6. Wegener J, Baresel A, Sthamer H. Evolutionary test environment for automatic structural testing. Information and Software Technology, 2001, 43(14): 841–854
7. Fraser G, Arcuri A. Evolutionary generation of whole test suites. In: Proceedings of the Quality Software International Conference, QSIC'11. 2011, 31–40
8. Wappler S, Wegener J. Evolutionary unit testing of object-oriented software using a hybrid evolutionary algorithm. In: Proceedings of IEEE Congress on Evolutionary Computation, CEC'06. 2006, 851–858
9. Alba E, Chicano F. Observations in using parallel and sequential evolutionary algorithms for automatic software testing. Computers and Operation Research, 2008, 35(10): 3161–3183
10. Li K, Zhang Z, Kou J. Breeding software test data with genetic-particle swarm mixed algorithms. Journal of Computers, 2010, 5(2): 258–265
11. Singla S, Kumar D, Rai H M, Singla P. A hybrid PSO approach to automate test data generation for data flow coverage with dominance concepts. International Journal of Advanced Science and Technology, 2011, 37: 15–26
12. Wu X, Wang Y, Zhang T. An improved GAPSO hybrid programming algorithm. In: Proceedings of International Conference on Information Engineering and Computer Science, ICIECS'09. 2009, 1–4
13. Zhang S, Ying Z, Hong Z, Qingquan H. Automatic path test data generation based on GA-PSO. In: Proceedings of IEEE International Conference onIntelligent Computing and Intelligent Systems, ICIS'10. 2010, 142–146
14. Chen S. Particle swarm optimization with pbest crossover. In: Proceedings of IEEE Congress on Evolutionary Computation, CEC'12. 2012, 1–6
15. Kaur A, Bhatt D. Hybrid particle swarm optimization for regression testing. International Journal on Computer Science and Engineering, 2011, 3 (5): 1815–1824
16. Goldberg D E, Holland J H. Genetic algorithms and machine learning. Machine Learning, 1988 3(2): 95–99
17. Eberhart R C, Kennedy J. A new optimizer using particle swarm theory. In: Proceedings of the 6th International Symposium on Micromachine Human Science, 1995, 39–43
18. Kennedy J, Eberhart R C. Particle swarm optimization. In: Proceedings of the IEEE International Conference on Neural Networks. 1995, 4, 1942–1948

19. Rabanal P, Rodriguez I, Rubio F. A functional approach to parallelize particle swarm optimization. In: Proceedings of Metaheuristicas, Algoritmos Evolutivos y Bioinspirados, MAEB'12. 2012
20. Jones B F, Sthamer H, Eyres D E. Automatic test data generation using genetic algorithms. Software Engineering Journal, 1996, 11(5): 299–306
21. Xanthakis S E, Skourlas C C, LeGall A K. Application of genetic algorithms to software testing. In: Proceedings of the 5th International Conference on Software Engineering and its Applications. 1992, 625–636
22. Harman M, Jones B. Search-based software engineering. Information and Software Technology, 2001 43(14): 833–839
23. Clark J, Dolado J J, Harman M, Hierons R, Jones B, Lumkin M, Mitchell B, Mancoridis S, Rees K, Roper M, Shepperd M. Reformulating software engineering as a search problem. IEE Proceedings-Software, 2003, 150(3): 161–175
24. Hansen N, Finck S, Ros R, Auger A. Real-parameter black-box optimization benchmarking 2009: noiseless functions definitions. INRIA Technical Report RR-6829, 2009
25. Younes M, Benhamida F. Genetic algorithm-particle swarm optimization (GA-PSO) for economic load dispatch. PRZEGLAD ELEKTROTECHNICZNY (Electrical Review), 2011, 87(10): 369–372
26. Juang C F. A hybrid of genetic algorithm and particle swarm optimization for recurrent network design. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 2004, 34(2): 997–1006
27. Saini D K, Sharma Y. Soft computing particle swarm optimization based approach for class responsibility assignment problem. International Journal of Computer Applications, 2012, 40(12): 19–24
28. Wappler S, Schieferdecker I. Improving evolutionary class testing in the presence of non-public methods. In: Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering, ASE'07. 2007, 381–384
29. Wilson G C, McIntyre A, Heywood M I. Resource review: three open source systems for evolving programs: Lilgp, ecj and grammatical evolution. Genetic Programming and Evolvable Machines, 2004, 5(1): 103–105
30. Tonella P. Evolutionary testing of classes. In: Proceedings of the International Symposium on Software Testing and Analysis, ISSTA'04. 2004, 119–128
31. Wang H C, Jeng B. Structural testing using memetic algorithm. In: Proceedings of the 2nd Taiwan Conference on Software Engineering. 2006
32. Arcuri A, Yao X. Search based software testing of object-oriented containers. Information Sciences, 2008, 178(15): 3075–3095
33. Nayak N, Mohapatra D P. Automatic test data generation for data flow testing using particle swarm optimization. Communications in Computer and Information Science, 2010, 95(1): 1–12
34. Ahmed B S, Zamli K Z, Lim C P. Constructing a T-way interaction test suite using particle swarm optimization approach. International Journal of Innovative Computing Information Control, 2011, 7(11): 1741–1758
35. Li A, Zhang Y. Automatic generating all-path test data of a program based on PSO. In: Proceedings of World Congress on Software Engineering, 2009, 4: 189–193
36. Fraser G, Arcuri A. Whole test suite generation. IEEE Transactions on Software Engineering, 2013, 39(2): 276–291
37. Gong D W, Zhang Y. Generating test data for both path coverage and fault detection using genetic algorithms. Frontiers of Computer Science, 2013, 7(6): 822–837
38. McMinn P. Search-based software test data generation: a survey. Software Testing, Verification and Reliability, 2004, 14(2): 105–156
39. McMinn P, Holcombe M. Evolutionary testing of statebased programs. In: Proceedings Conference on Genetic and Evolutionary Computation, GECCO'05. 2005, 1013–1020
40. Rothermel G, Untch R, Chengyun C, Harrold M J. Prioritizing test cases for regression testing. IEEE Transaction of Software Engineering, 2001, 27(10): 929–948
41. Arcuri A, Briand L. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: Proceedings of 33rd International Conference on Software Engineering ICSE'11. 2011, 1–10
42. Ali S, Briand L C, Hemmati H, Panesar-Walawege R K. A systematic review of the application and empirical investigation of search-based test case generation. IEEE Transactions of Software Engineering, 2010, 36(6): 742–762
43. Vargha A, Delaney H D. A critique and improvement of the CL common language effect size statistics of McGraw and Wong. Journal of Educational and Behavioral Statistics, 2000, 25(2): 101–132
44. Grissom R, Kim J. Effect sizes for research: a broad practical approach. Lawrence Erlbaum, 2005

Ms Priyanka Chawla is pursuing doctoral program (PhD) at Computer Science and Engineering (CSE) Department of Thapar University, India. Her qualifications includes B.Tech (CSE), M.Tech(CSE). She is a dedicated researcher in the field of soft computing, cloud computing and software engineering. She has authored various research papers and book chapters in journals of good repute. She is member of ACM and ISTE.

Dr. Inderveer Chana is PhD in Computer Science with specialization in Grid Computing and ME in Software Engineering from Thapar University, India and BE in Computer Science and Engineering. She is presently serving as associate professor in the Computer Science and Engineering Department of Thapar University, India. Her research interests include grid and cloud computing and other areas of interest are software engineering and software project manage-

ment. She has more than 70 research publications in reputed Journals and Conferences. Under her supervision, one PhD thesis has been awarded, two are submitted and five PhD thesis are on-going in the area of grid and cloud computing. She is also working on two major research projects in the area of energy aware utility and cloud computing.

Prof (Dr.) Ajay Rana has a rich experience of industry and academia of around 15 years. He has had an outstanding academic record and is a product of prestigious system throughout. He has published more than 177 research papers in reputed journals and proceedings of international and national conferences. He has co-authored 05 books and co-edited 36 conference proceedings. He has delivered invited lectures in more than 36 technical and management workshop/conferences programs in India and abroad. He is a member of board of governess (BOG), advisory council (AC), academic executive (AE) member, board of studies (BOS) and special member of many Indian and foreign universities as well as industry. He is editor in chief, technical committee member, advisory board member for 18 plus technical journals and conferences at national and international levels. He has received a number of awards and honors like eduCLUSION AWARD 2014 for displaying extraordinary initiative for higher technical education (Singapore 2014), IMTT award in Italy (2014), International WHO'S WHO of Professionals, USA. (2014), IT Next CIO award 2011 in pune, best advisor SIFE at Mumbai in 2011 etc and was recipient of national merit scholarship.