STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
Stat., Optim. Inf. Comput., Vol. 10, June 2022 , pp 968–982.
Published online in International Academic Press (www.IAPress.org)

A Hybrid Harmony Search and Particle Swarm Optimization Algorithm (HSPSO) for Testing Non-functional Properties in Software System

Nurudeen Muhammad Bala *, Suhailan Bin Safei

Faculty of Informatics and Computing, Universiti Sultan Zainal Abidin Gong Badak, 21300 Kuala Terrengganu, Malaysia

Abstract An important aspect of improving software system is testing. However, it is time demanding and sometimes labour intensive if done manually. In this paper, we developed an automatic search-based approach for testing the non-functional properties of a software system using hybrid harmony search and particle swarm optimization algorithms. The approach birthed a new algorithm named HSPSO, which is proposed based on the strength of HS over Genetic algorithm (GA) in terms of less adjustable parameters, quick convergence and smooth implementation. On the other hand, we propose the PSO to complement the drawback of HS in terms of time consumption problem. Besides, we used four programs for the comparative efficiency analysis of the proposed algorithm in relation to competing algorithms based on average branch coverage and execution time. The results from the analysis showed that the HSPSO algorithm was able to achieve 100% average coverage with a fewer number of generated test cases and under limited execution time.

Keywords Test data generation, Harmonic Search (HS), Particle Swarm Optimization (PSO), Branch coverage, WCET, BCET

DOI: 10.19139/soic-2310-5070-1039

1. Introduction

Software testing identifies the test cases which determine errors or detect the presence of a fault in the program, which eventually causes software failures [1, 2, 3]. The process of testing any software system is a tedious task which is expensive and time-consuming; almost 50% of software system development resources used while adding nothing to the raw functionality of the final product [4]. Moreover, the increase in the difficulty in the growth of software has directed the clients to demand higher quality software. Thus it was noticed that properties concerning non-functionalities in software need to be considered as a primary factor in the developmental process. However, it was also observed that several software systems failed due to higher inadequacies in these properties [5]. Previous studies shows that non- functional part are treated as properties or traits of completed programming to be considered among software engineering researchers [6]. Software testing is generally used to estimate the quality of a software system where the condition is defined as a functional attribute on the behaviour of the system and non-functional requirements such as reliability, efficiency, portability, maintainability, compatibility, and usability [8]. Consequently, software test is comprised of two significant components, such as an input to the executable program and a definition of the expected outcome [9]. Nevertheless, it is essentially importance to maintain the number of test cases as smallest as possible and ensure that the generated test inputs covers as many lines of codes as possible [3]. Moreover, since human testers perform these tasks, several errors were noticed to occur [10]. Therefore, it is necessary to consider search-based methods incorporated with criteria, such

*Correspondence to: NURUDEEN MUHAMMAD BALA (Email: SI2450@putra.unisza.edu.my). Faculty of Informatics and Computing, Universiti Sultan Zainal Abidin Gong Badak, 21300 Kuala Terrengganu, Malaysia.

ISSN 2310-5070 (online) ISSN 2311-004X (print)
Copyright © 2022 International Academic Press

as branch or method coverage to extent the effectiveness of a test case and optimize such that there are no faults exist [11]. Hence, it is necessary to develop an exhaustive testing technique for efficient testing of non-functional properties of the software system [12]. Current researches observed the significance of non-functional search-based software testing (NFSBST) [13]. There by allowing the identification of potential non-functional properties suitable for applying numerous techniques that provides a general view of existing non-functional properties using metaheuristic search techniques. In this study, the existing non-functional properties were identified and reviewed based on the properties to determine the constraints and limitations. The study seeks to extend the early optimistic results by applying NFSBST to larger real world systems towards a generalization of results. According to [12], it is required to identify various software systems to which search-based software testing might be useful to correlate with a set of non-functional properties . Besides poor productivity, slow processing, high cost, and low quality were found to be the issues in the software system [6]. This will provide list out a set of challenges and suggestions to solve them. However, the data needed for the research is considered as a prime concern. Therefore, it is necessary to extend the idea such that would lead to the formulation of a framework for Non-functional Search-based Software Testing (NFSBST). Moreover, a larger framework will provide the identification of diverse information.

An extensive review in [7] identified the various practical challenges and problems of search-based software (SBST) test data generation as execution environment handling issues, branch coverage, fitness function designs, maximization problem exploration, structured parallel approaches, single versus multi-objective functions, regression test optimization and most especially testing non-functional properties. Of the eight challenges identified, seven have been well documented and solved in many studies while there are still many rooms for improvement as regards non-functional software testing such as optimizing test cases to analyze the Best Case Execution Time (BCET) and Worst Case Execution Time (WCET). Thus, it is important to develop optimal test cases generation technique useful for testing non-functional properties such as BCET and WCET.

In this paper, we developed an enhanced search-based technique for testing of non-functional properties of a software system using hybrid harmony search and particle swarm optimization algorithm. The remainder of this paper proceeds as follows. Section 2 presents related work. Section 3 presents the proposed methodology. Section 4 presents the experimental setup, Section 5 discusses the results, and finally, Section 6 concludes the paper with some indication of future work.

2. Related Works

In the recent past, it was observed that with the increase in the size of the software systems, the manual generation of test inputs was found to be costly and a difficult task [14]. Thus, several studies were conducted using diverse techniques for the automation of test data generation. From those researches, it was noticed that the enumeration of test inputs is infeasible for large programs as the random methods involved in the process ignores features of the software which are not included in the tests [15]. The foremost complexities engaged with test data generation were found to be the size and complexity of the software as the problems were found to be unpredictable [16]. Also, there are several approaches employed to derive test cases from models by automated test case generation [17]. Other methods often used include the production of test cases based on input source code for dealing with the size of the software, such as Genetic Algorithms (GA) [18, 19] . Similarly, Simulated Annealing (SA) [20], Ant Colony Optimization (ACO) [21] have also been used for the same purpose. Ram¨ªrez, Romero, & Ventura [22] develop a search-based algorithm based on the Simulated Annealing (SA) optimization to solve complex problems. This search-based meta-heuristic technique was employed for the processing of annealing in metallurgy. The SA algorithm was further used in extracting source code design abstractions. This showed the applicability of the SA algorithm in solving multi-objective tasks. The author also compared the algorithm with other search-based algorithms. Qin et al. [21] defined the opportunities of simulated annealing in designing software architecture and further applied the algorithms to modularize the source code classes into packages. This approach was developed by automating the process of reducing package coupling and cycle dependencies for systems that are based on an object oriented system. In addition to the GA and SA algorithm, Hill Climbing (HC) algorithm has also been applied to solve the software modularization problem in the experiments conducted by [23]. According to [24] Genetic Algorithm (GA) uses a scientific model to generate test data for numerous paths coverage but fail to establish test platform and generate test data when the number of target paths is vast. In addition, Firefly algorithm(FA) is used to maximize the mathematical function, but the study fails to use fitness function that may give better code, statement coverage and maximize path coverage [25]. According to [26], Genetic Algorithm(GA) and Particle Swarm Optimization(PSO) were combined for Software Test Case Generation. The result fails to analyze the test case generation in each iteration. Also, [28] combined Particle Swarm Optimization (PSO) and Artificial Bee Colony (ABC) Algorithm for optimizing the Test Cases is used, but time for convergence is uncertain. This technique does not solve the test case generation problem for non-functional software testing.

2.1. Evaluation Metrics

The metrics employed in this paper is based on system utility which is determined by its structural (branch coverage: Average Coverage and Average Generation) and non-functional (Worst-Case Execution Time [WCET] and Best-Case Execution Time [BCET]) characteristics [27, 29, 28, 30, 31, 32]. This paper was implemented by focusing on these characteristics incorporated in the software system to overcome the poor productivity, slow processing, high cost, and low quality which were found to be the issues in the software system. As a recent population-based metaheuristic algorithm, HS algorithm was first used to solve the optimization problem and was considered an efficient combinatorial optimization algorithm. Compared to traditional evolutionary algorithms (genetic based), this algorithm has been shown to have several advantages such as less adjustable parameters, rapid convergence and smooth implementation. However, this conventional HS algorithm is not possible to be directly applicable to the time consumption problem. Thus, the algorithm is modified and then applied to solve the object-oriented software re-modularization problem. Therefore, in this paper, we developed an efficient methodology by integrating Harmony search-based algorithm with Particle Swarm optimization algorithm for the testing of non-functionalities in the software system.

2.2. Harmonic Search

The common harmony search optimization approach has three parameters which are, the harmony memory size HMS, the harmony memory consideration rate HMCR and the pitch adjustment rate PAR [33]. As required in most optimization technique, the maximum number of iterations is also provided. HMS procedure starts with generation of initial solutions $x_1, x_2, \ldots, x_{HMS}$ which follows a uniform distribution $U(0,1)$. The process is repeated iteratively with newly generated solutions until the maximum number of iterations is exceeded. The pattern of solutions is unique such that they are independent of p decisions variables used. By the use of the consideration probability HMCR, a memory consideration step is achieved. The next step involves the selection of a decision variable value from the stored uniform random variables initially stored in the HMS.

Furthermore, the next step involves the consideration of pitch adjustment rate using the probability PAR. If PAR is set at 0.5, pitch adjustments increase the decision by 1, and if otherwise, it decreases the decision variable by 1. If the probability exceeds the interval bounds, the decision variable remains at the initial point. In contrast, the step will consider no memory such that it occurs with a probability that equals $1 - HMCR$, which implies a typical random selection has occurred. At this step, the decision variable takes any value from the uniform distribution.

Consequently, the yielded solution is supplied to the objective function for convergence examination. If the current solution is better than the worst solution in the harmonic memory, the new solution suffices and if otherwise the worst solution is retained for further examination in the preceding step provided the

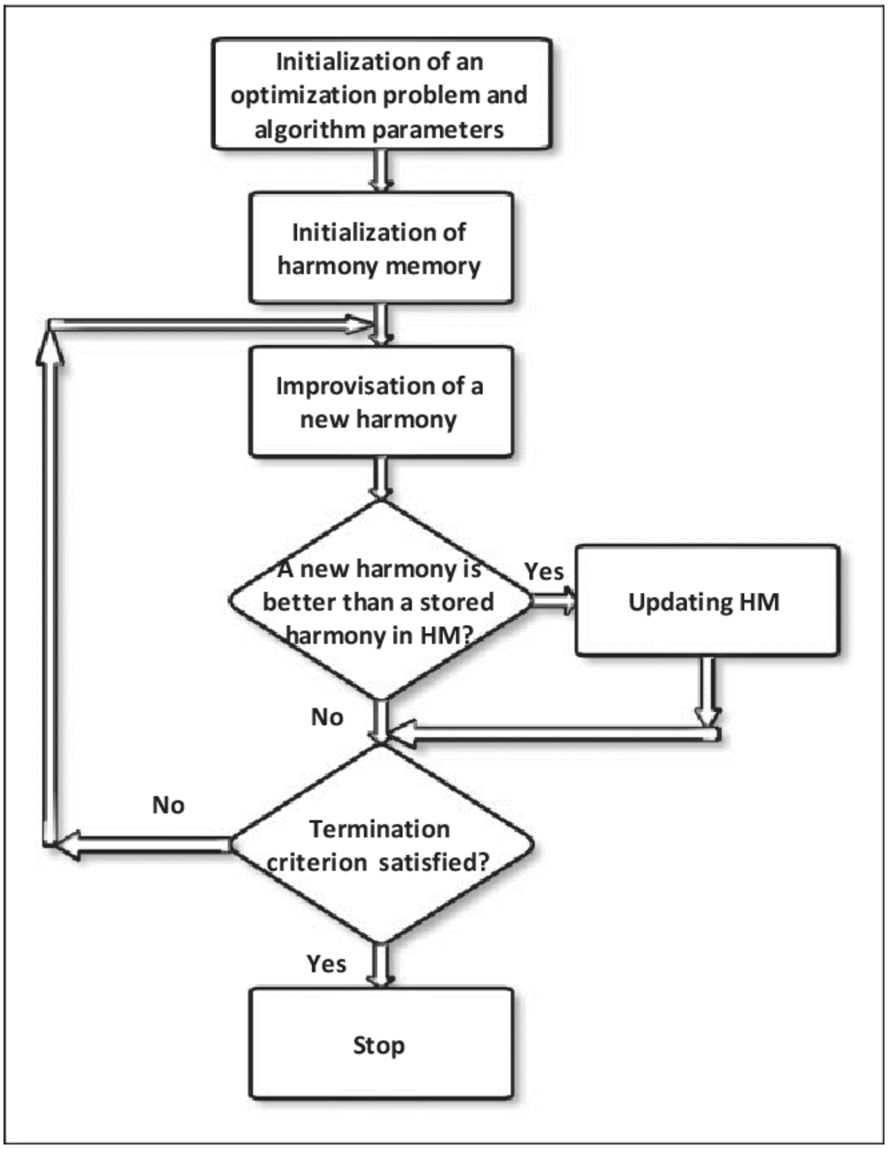

Initialization of an optimization problem and algorithm parameters
Initialization of harmony memory
Improvisation of a new harmony
A new harmony is better than a stored harmony in HM?
Yes
Updating HM
No
No
Termination criterion satisfied?
Yes
Stop

Figure 1. Flow chart of Harmony Search Optimization [34]

maximum number of iterations is not exceeded. If the number of iteration is reached, the best solution from the harmonic memory is returned. Figure 1 presents the flowchart of the algorithm.

2.3. Particle Swarm Optimization (PSO)

Towards the end of the last century, the particle swarm optimization algorithm was first proposed by [32]. This kind of artificial intelligence algorithm is an imitation of birds in the process of searching for food in nature. Each particle swarm is set with velocity and position parameters like the birds' move, and its diversity can be ensured by changing the particle velocity and position in the process of evolution. At the beginning of each iteration, particle information is compared with others to find the best mobile solutions which can meet the end conditions [35]. If an optimal solution is found, the movement terminates otherwise it will continue to evolve until the output particle. Besides, its speed and position parameters of each particle, there is a value of the fitness for each particle. This value can be used to judge the good or bad particle. In the evolution process, the choice of the best particle is the evolutionary process by which a particle is continuously updated to compare itself with the optimal particle. Assuming that c_1 and c_2 are respectively, the acceleration factor of the best particle selected from the overall situation and the individual, w is the inertia factor, $r_1()$ and $r_2()$ are uniform random values in the interval $[0, 1]$, and the update rate of the particle is:

$$v_{id}(t+1) = wv_{id}(t) + c_1 r_1()[pbest_{id}(t) - x_{id}(t)] + c_2 r_2()[gbest_{id}(t) - x_i d(t)] \tag{1}$$

With the use of the dimension vector D to represent the particle's individual information, the position of each particle is expressed as $X_i = (x_{i1}, x_{i2}, \ldots, x_{id})$, and the formula for the particle to update its position is:

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

In formula (1), the acceleration factors of c_1 and c_2 are relative to the key. When the values are relatively small, it will cause the small particle's moving distance, and it is difficult to obtain qualified particles. When the values are more substantial, it will be too far away from the particle flight distance and make it easy to deviate from the target area. According to previous studies, it is easier to obtain qualified particles when the selected value is 1.

The algorithm flow of particle swarm algorithm is as follows. (1) To initialize the parameters such as velocity and position of particle, and initial parameters are randomly generated in the space of the solution. Furthermore, the initial parameters are set up to set the global and the extreme value of the particle. (2) The velocity and position of the particle are updated, and the extreme value is compared with the set value. If it is in accordance with the requirements of evolution, the particle information and the global extremum are transformed. Otherwise, the conversion of particle information and individual extremum will be made. (3) To update the velocity and position of the particle in the solution space, continue to search for the updated particles as the new particles. The updated particles are used as the new particles, and the following steps are repeated to carry out a round of evolution. (4) Each evolution is compared to whether particles can meet the maximum number of iterations set. If the algorithm satisfies the algorithm, the optimal particle will be output. Otherwise, repeated cycle steps are needed until the optimal solution is obtained [36]. The basic algorithm flow of particle swarm is shown in Figure 2.

3. HYBRID HARMONIC SELECTION AND PARTICLE SWARM OPTIMIZATION BASED TEST DATA GENERATION (HSPSO)

The HSPSO algorithm is designed to integrate the potential of HS and PSO to simultaneously minimize the number of test generation and execution of search-based testing. The algorithm starts with the implement of the HS algorithm to find preliminary optimization vectors needed to be supplied as the initialization particles in the PSO algorithm. The optimal solutions of HS are used to replace random initial values in the PSO algorithm. This procedure will enhance the execution time of PSO by reducing the search time since near-optimal values are used as initial values. In cases where the HS optimal equals the PSO optimal, the algorithm computation time of HSPSO equals HS and if otherwise the execution time is bounded in the interval:

$$time(HS) \leq time(HSPSO) \leq time(HS + PSO)$$

.

Generally, the framework of test data generated using search-based technique involves the harmonization of Meta Heuristics Search (MHS) algorithm and programs dynamic execution. Information coverage is achieved when the search algorithm results in the test suite where programs can be executed through the Program Under Test (PUT). Thus, the corresponding criterion of the fitness value is achieved

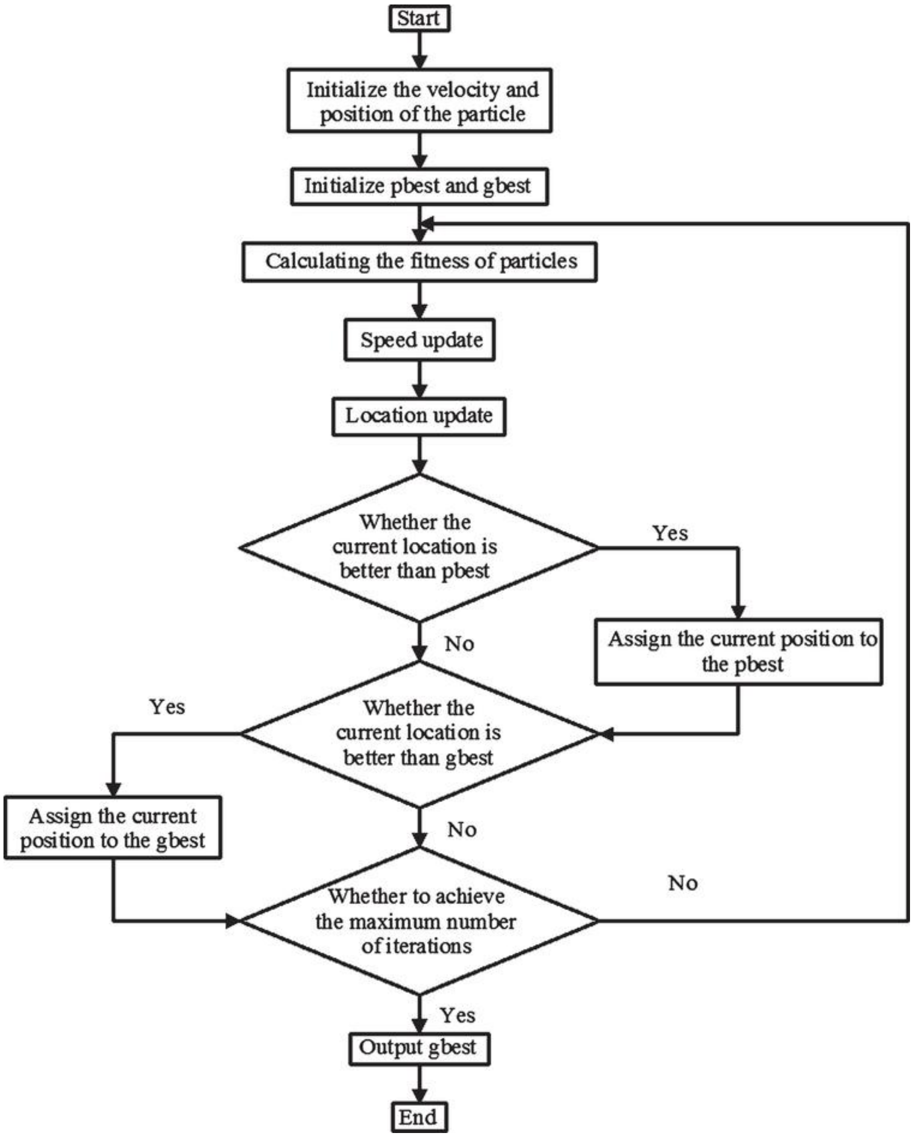

Start
Initialize the velocity and position of the particle
Initialize pbest and gbest
Calculating the fitness of particles
Speed update
Location update
Whether the current location is better than pbest
Yes
Assign the current position to the pbest
No
Whether the current location is better than gbest
Yes
Assign the current position to the gbest
No
Whether to achieve the maximum number of iterations
No
Yes
Output gbest
End

Figure 2. Particle Swarm Optimization [35]

and modified to fit into a new test suite so that maximum coverage can be attained. Nevertheless, attaining the perfect communication among the coverage information extraction and basic search algorithm remain a major challenge. In the case of the HSPSO search-based testing, each argument of the desired PUT $(arg_1, \ldots, \text{arg}_p)$ is first initialized as HMS of various solutions $(arg_i^{(1)}, arg_i^{(2)}, \ldots, arg_i^{(HMS)})$ and are generated from a uniform distribution. The process is then repeated iteratively with newly generated solutions until the maximum number of iterations is exceeded. The entire flow of HS is followed until the optimal solutions for each arg_i is achieved at maximum iteration. Next step is to supply the best solution from HS for all argument as a p-dimension position vector. For some given non-functional criteria such as (coverage, execution time, number of generation), a fitness function $f(.)$ is defined for PSO. Instead of initializing the PSO with random values or zeros, the f(pbest) and f(gbest) are computed by setting initial values to HS optimal solution. The optimal test cases are then obtained from the optimal PSO values. Figure 3 displays HSPSO search-based testing.

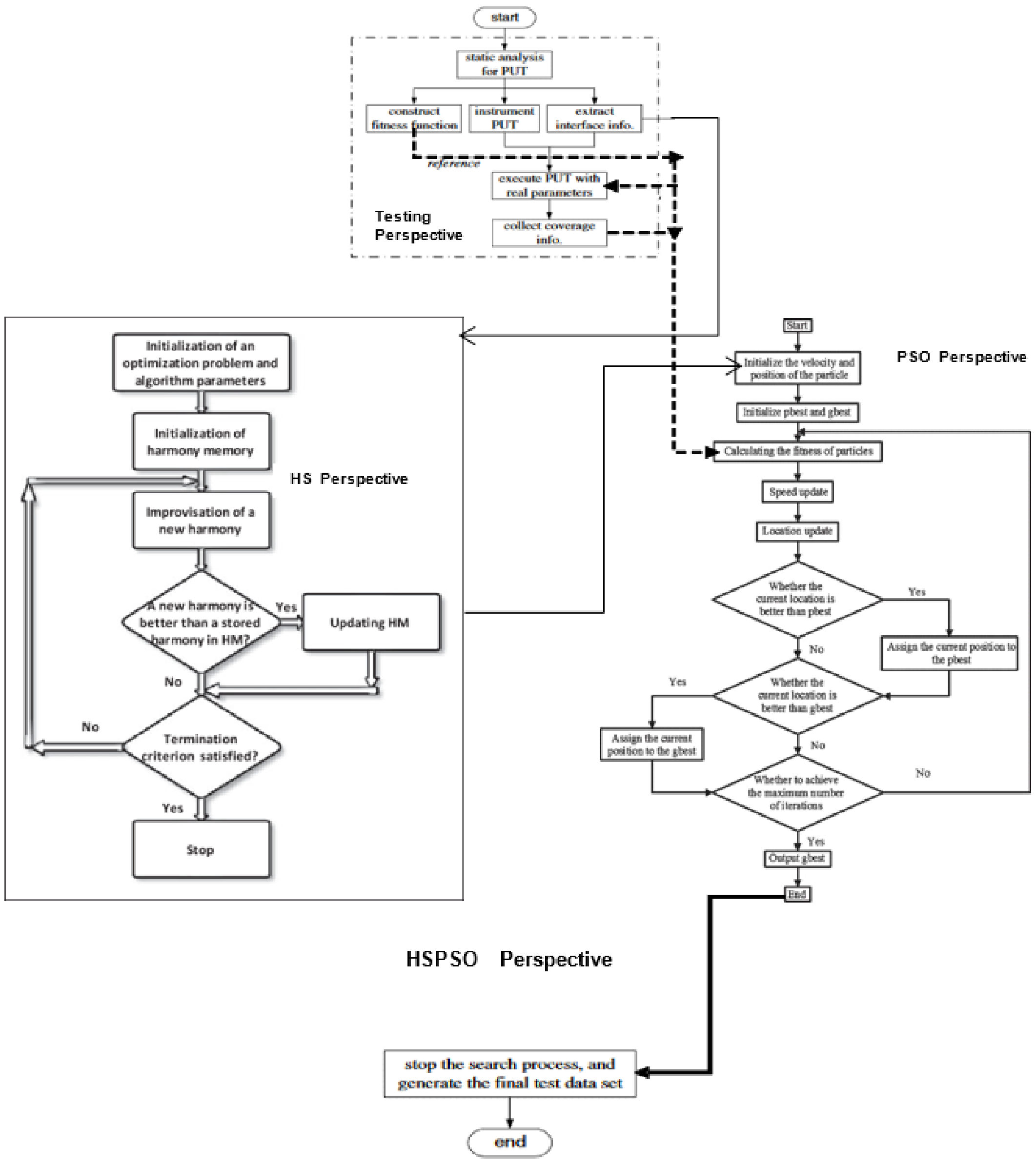
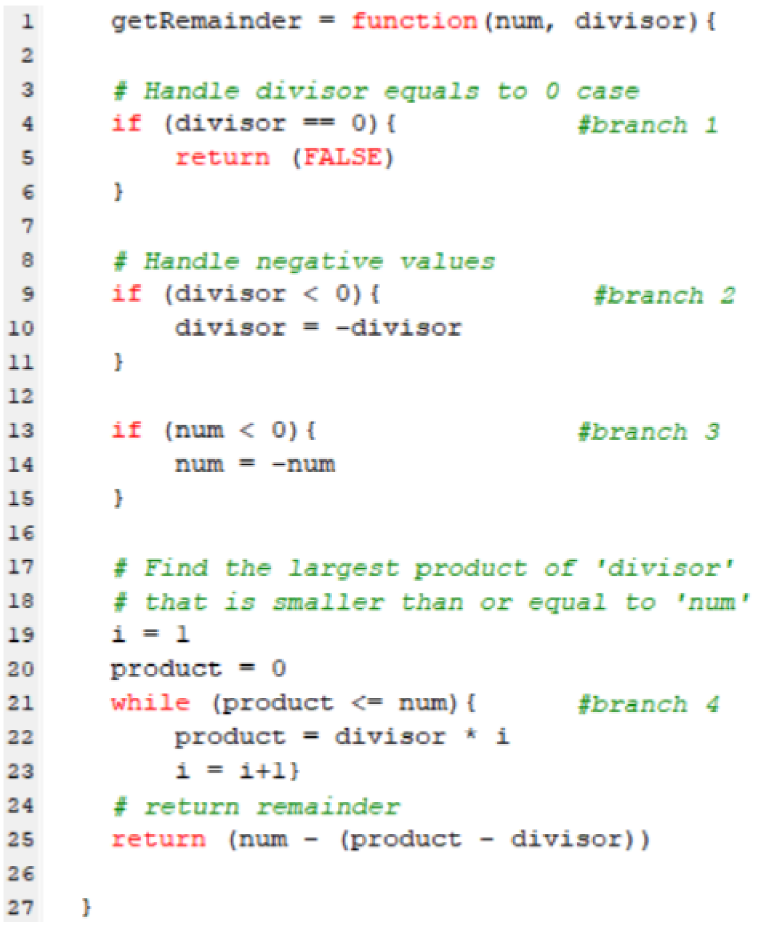

start
static analysis for PUT
construct fitness function
instrument PUT
extract interface info.
reference
execute PUT with real parameters
collect coverage info.
Testing Perspective
Initialization of an optimization problem and algorithm parameters
Initialization of harmony memory
HS Perspective
Improvisation of a new harmony
A new harmony is better than a stored harmony in HM?
Yes
Updating HM
No
Termination criterion satisfied?
No
Yes
Stop
Start
Initialize the velocity and position of the particle
PSO Perspective
Initialize pbest and gbest
Calculating the fitness of particles
Speed update
Location update
Whether the current location is better than pbest
Yes
Assign the current position to the pbest
No
Whether the current location is better than gbest
Yes
Assign the current position to the gbest
No
Whether to achieve the maximum number of iterations
No
Yes
Output gbest
End
HSPSO Perspective
stop the search process, and generate the final test data set
end

Figure 3. HSPSO-based test data generation algorithm.

3.1. Fitness Function

Coverage information and criterion are of paramount importance in ensuring the adjustment of the search direction that helps to find a lasting solution to the search based software testing algorithm.. In coverage testing, the construct basics, such as statements, branches, paths and definition-use pairs, are treated as coverage objects [34]. In this study, Search objective is commonly used as branch coverage and all branches in program code can be covered by functional test suite.

The probes previously embedded in the PUT code are often used to collect the branch coverage information. Each branch is monitored by a specific probe installed to determine if the path is covered



```
1     getRemainder = function(num, divisor){
2
3     # Handle divisor equals to 0 case
4     if (divisor == 0){              #branch 1
5         return (FALSE)
6     }
7
8     # Handle negative values
9     if (divisor < 0){                #branch 2
10        divisor = -divisor
11    }
12
13    if (num < 0){                   #branch 3
14        num = -num
15    }
16
17    # Find the largest product of 'divisor'
18    # that is smaller than or equal to 'num'
19    i = 1
20    product = 0
21    while (product <= num){         #branch 4
22        product = divisor * i
23        i = i+1}
24    # return remainder
25    return (num - (product - divisor))
26
27  }
```

Figure 4. getRemainder program showing branches

or otherwise. Finding optimal solution in search base test data generation, fitness function determines a prominent role because it is the only available information about the PUT at this stage. For the HSPSO, the fitness function will enable us to know the goodness of fit of the provided particle solution concerning the overall optimum solution. In general, comparing the fitness function of solutions in each population helps in attaining an optimal solution. To measure the fitness function, objective function is decoded by positioning vector of the program¡¯s argument [32]. For example, in the program getRemainder in Fig. 4, *line*4 corresponds to branch 1 of the code with the statement below:

$$If(divisor == 0)\{return(FALSE)\} \;\; else \;\; \{continue\}. \tag{3}$$

If x is represented as input, then the correlation between input variable and data in divisor can be indicated as: $line_4(x)$.

Then the fitness function for this condition (3) can be express as:

$$f(x) = -line_4(x) + k \tag{4}$$

Where k is a positive constant when $f(x) \leq 0$ the branch goal is achieved. Substantial value of $f(x)$ is set as fitness to represent the condition (3) that is not related to the input data and preferred condition to minimize the problem of generating test data [37].

For HSPSO search-based, a branch distance function is created as in the case of $f(x)$ for each predicate in PUT (see Table 1). The fitness of each branch determines the fitness function of entire program, for

Table 1. Branch fitness function for different predicates

S/N	Predicate	Fitness function
1.	Boolean	If true then 0 else k
2.	$\sim x$	Negation over x
3.	$x_1 = x_2$	If $\lvert x_1 - x_2\rvert = 0$ then 0 else $\lvert x_1 - x_2\rvert + k$
4.	$x_1 \neq x_2$	If $\lvert x_1 - x_2\rvert \neq 0$ then 0 else k
5.	$x_1 < x_2$	If $x_1 - x_2 < 0$ then 0 else $\lvert x_1 - x_2\rvert + k$
6.	$x_1 \leq x_2$	If $x_1 - x_2 \leq 0$ then 0 else $\lvert x_1 - x_2\rvert + k$
7.	$x_1 > x_2$	If $x_2 - x_1 > 0$ then 0 else $\lvert x_2 - x_1\rvert + k$
8.	$x_1 \geq x_2$	If $x_2 - x_1 \geq 0$ then 0 else $\lvert x_2 - x_1\rvert + k$
9.	x_1 and x_2	$f(x_1) + f(x_2)$
10.	x_1 or x_2	$\min[f(x_1), f(x_2)]$

example the fitness function of the entire program that has B branches can be determined by the formula (5) below:

$$fitness = \frac{1}{\sum_{b=1}^{B} f(x)_b} \tag{5}$$

4. EXPERIMENTAL SETUP

The HSPSO-based test data generation procedure was demonstrated using four real-world programs. The four programs are well-known benchmark programs and documented in [29]. Specific details of each of the four programs are documented in Table 2. The experiment is performed in the environment of MS Windows 10 with 64-bits and runs on Core i5 with CPU @ $1.60GHz - 1.80GHz$ and 8 GB memory. The algorithms are implemented in Splus and run on the platform of R Studio version 3.6.1. In the experiments, we compare the coverage, the number of generations and execution times for GA, ABC, HGPSTA, PSO, PSABC and HSPSO. The population size (Ps) for each algorithm was varied between $10 - 60$ with an increment of 10. Other peculiar parameters for each algorithm are illustrated in Table 3. The six algorithms were compared across the four programs using the following metrics:

Table 2. Description of programs used for comparative analysis

Program	$\#(args)$	Branch	Description
getRemainder	2	18	Calculate the remainder of an integer division
getDayofWeek	3	11	Calculate the day of the week
classifyTriangle	3	5	Type classification for a triangle
getDifference	6	18	Compute the days between two dates

1. Average coverage (AC): this is the average of the branch coverage attained by all test inputs in 1,000 runs.
2. Average generation (AG): this is the average evolutionary generation for realizing all branch coverage.
3. Best Case Execution Time (BCET): this is the minimum execution time (ms) a program can be executed using an optimal generated test case. This process is repeated 100000 times for stability.
4. Worst-Case Execution Time (WCET): this is the maximum execution time (ms) a program can be executed using an optimal generated test case. This process is repeated 100000 times for stability.

5. Average Case Execution Time (ACET): this is the average execution time (ms) a program can be executed using an optimal generated test case. This process is repeated 100000 times for stability.

Table 3. Specific parameter setup for each algorithm

Algorithm	Parameter	Value
GA	Selection strategy	Roulette Whell Selection
	Cross-over probability	$Pc = 0.8$
	Mutation probability	$Pm = 0.1$
ABC	Cycle limit	$cl = Ps \times \#args$
PSO	Inertia weight (w)	
	Acceleration factors c_1 and c_2	$c_1 = c_2 = 1.49445$
	Maximum velocity	$Vmax = 20$
HGPSTA (GA + PSO)	Same as GA and PSO	Same as GA and PSO
PSABC (PSO + ABC)	Same as PSO and ABC	Same as PSO and ABC
HSPSO (HS + PSO)	Harmony memory considerate rate	$HMCR = 0.95$
	Pinch Adjusting Ratio	$PAR = 0.3$
	Same as PSO	Same as PSO

5. RESULTS AND DISCUSSION

Table 4 shows the results for AC and AG for each program and two criteria, the best algorithm on average for all programs is the proposed HSPSO. For a specific program, the best algorithm in terms of the minimal number of generations and 100% coverage is HSPSO. However, the existing hybrid algorithms (HGPSTA and PSABC) also compete favourably with the proposed HSPSO. After HSPSO, the next is HGPSTA then PSABC follows. PSO has high coverage with a high number of generations, thus making it not entirely usable as this also results in high execution time. The GA algorithm is the lowest in terms of coverage but with a considerable number of generations.

Table 4. Comparison analysis for Average Coverage (AC) in % and Average Generation (AG).

Program	Metrics	GA	ABC	PSO	HGPSTA	PSABC	HSPSO
getRemainder	AC	97.07	99.54	99.86	99.89	99.93	100.00
	AG	9.41	10.81	10.52	17.70	17.41	12.82
getDayofWeek	AC	96.50	100.00	98.69	99.97	100.00	100.00
	AG	2.97	3.36	3.09	6.17	5.93	7.05
classifyTriangle	AC	84.78	100.00	100.00	100.00	99.99	99.99
	AG	46.73	47.26	46.69	94.36	95.31	93.64
getDifference	AC	99.63	100.00	100.00	100.00	100.00	100.00
	AG	7.99	7.71	7.54	14.93	14.77	14.39
Average	AC	94.50	99.89	99.64	99.97	99.98	100.00
	AG	16.78	17.29	16.96	33.29	33.36	31.98

Table 4 present the results for the AC and AG for each of the eight programs. For the two criteria, the best algorithm on average for all programs is the proposed HSPSO. However, in terms of the number of generation, the hybrid algorithms (HGPSTA, PSABC and HSPSO) returned high average generations

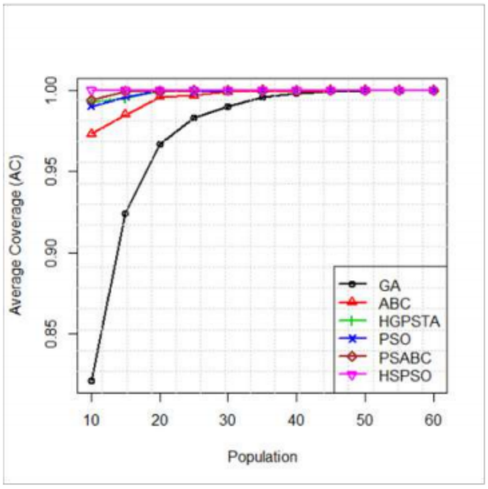
Table 5. Comparison analysis for Best Case Execution Time (BCET), Worst-Case Execution Time (WCET) and Average Case Execution Time (ACET) in nanoseconds (ns).

Program	Metrics	GA	ABC	PSO	HGPSTA	PSABC	HSPSO
getRemainder	BCET	710.04	994.88	920.85	1441.83	1464.12	1047.43
	ACET	940.55	1081.44	1051.51	1770.43	1741.14	1281.66
	WCET	1171.06	1168.00	1182.17	2099.03	2018.15	1815.90
getDayofWeek	BCET	2.51	3.00	2.63	6.03	4.85	5.85
	ACET	2.97	3.36	3.09	6.17	5.93	7.05
	WCET	3.42	3.71	3.55	6.32	7.01	8.26
classifyTriange	BCET	43.71	44.22	44.08	88.22	89.57	88.58
	ACET	46.73	47.26	46.69	94.36	95.31	93.64
	WCET	49.76	50.31	49.29	100.49	101.05	98.70
getDifference	BCET	6.42	6.38	6.00	12.80	11.72	11.60
	ACET	7.99	7.71	7.54	14.93	14.77	14.39
	WCET	9.56	9.04	9.09	17.06	17.81	17.19
Average	BCET	190.67	262.12	243.39	387.22	392.57	288.37
	ACET	249.56	284.94	277.21	471.47	464.29	349.19
	WCET	308.45	307.77	311.03	555.73	536.01	485.01

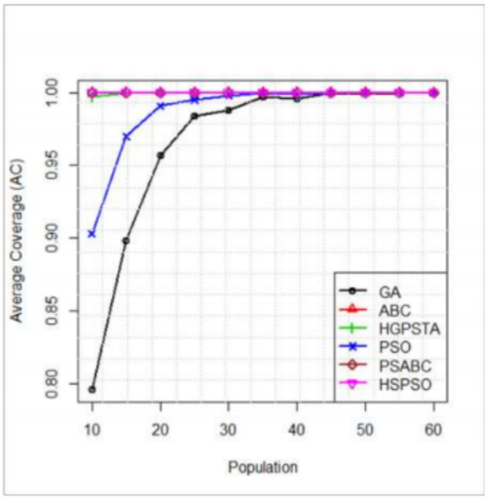
compared to single algorithms (GA, ABC and PSO). This is obvious as the number of generations is summed up for the two algorithms combined to form the hybrid algorithms. Thus, within the class of single algorithms (GA, ABC and PSO) the best in terms of a minimal number of generations is ABC while for the hybrid procedure, the best is the proposed HSPSO. For a specific program, the best algorithm for getRemainder, getDayofWeek, classifyTriangle, getDifference, getBesselj, printCalendar, computeTax, lines, is HSPSO in terms of coverage while in terms of average generations the best is ABC. The result of ABC is not reliable as it could not guarantee high coverage. Overall, the best algorithm that can guarantee high coverage with minimal sufficient average generations within the class of hybrid procedures is HSPSO.

Figure 5 presents the association between the population size and average coverage for the various algorithms. The plots show that the higher the population size, the higher the coverage for all programs. The worse algorithms, in terms of coverage, is GA. Figure 6 shows the impact of population size on Average Generation AG. Although, the results of AC in Figure 5 shows that for the AC, the higher the population size, the higher the AC, however, in contrast, the higher the population size, the lower the number of generations.

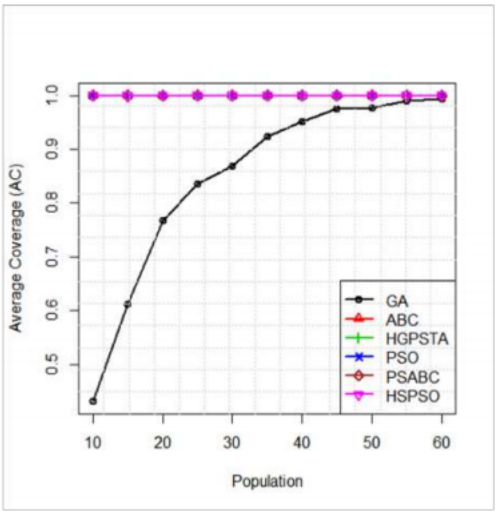
Table 4 further gives the average of the average generation (AG) across various population sizes. The AG results show that there exists a clear distinction between single and hybrid algorithms. The hybrid algorithms returned high values in average generations compared to the single algorithms. For the getRemainder program, GA returned the lowest AG among the single algorithm while within the class of hybrid algorithms, HSPSO returned the lowest AG. This shows that for the getRemainder program, HSPSO is the best among the main comparison of algorithms within the class of hybrid algorithms. For the getDayofWeek program, GA returned the lowest AG among the single algorithm while within the class of hybrid algorithms, PSABC returned the lowest AG. This shows that for the getDayofWeek program, PSABC is the best among the main comparison of algorithms within the class of hybrid algorithms. For the classifyTriangle program, PSO returned the lowest AG among the single algorithm while within the class of hybrid algorithms, HSPSO returned the lowest AG. This shows that for the classifyTriangle program, HSPSO is the best among the main comparison of algorithms within the class of hybrid algorithms. For the getDifference program, PSO returned the lowest AG among the single algorithm while within the class



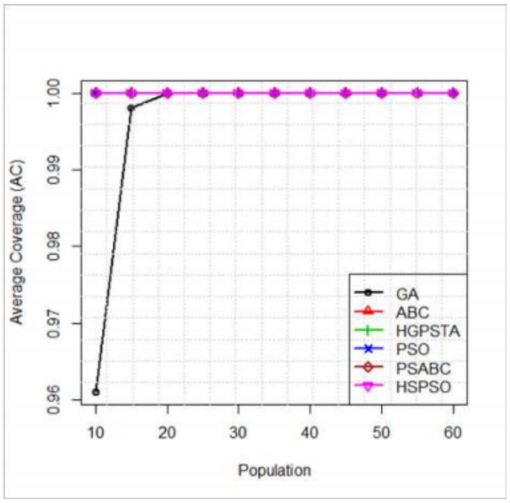
(a) getRemainder



(b) getDayofWeek



(c) classifyTriangle



(d) getDifference

Figure 5. Average Coverage (AC) versus population size

of hybrid algorithms, HSPSO returned the lowest AG. This shows that for the getDifference program, HSPSO is the best among the main comparison of algorithms within the class of hybrid algorithms.

Table 5 presents the results of the execution time. Three types of execution timings were considered; BCET, ACET and WCET. For getRemainder program, the algorithm that returned test cases that minimize the BCET within the class of single algorithms is GA, while PSO returned test cases that minimize the ACET and ABC returned test cases that maximize the WCET. Similarly, within the hybrid algorithms, HSPSO returned test cases that minimize the BCET, ACET and maximize the WCET. For getDayofWeek program, the algorithm that returned test cases that minimize the BCET within the class of single algorithms is GA, while GA also returned test cases that minimize the ACET and as well returned test cases that maximize the WCET. Similarly, within the hybrid algorithms, PSABC returned test cases that minimize the BCET, ACET and maximize the WCET. For classifyTriangle program, the

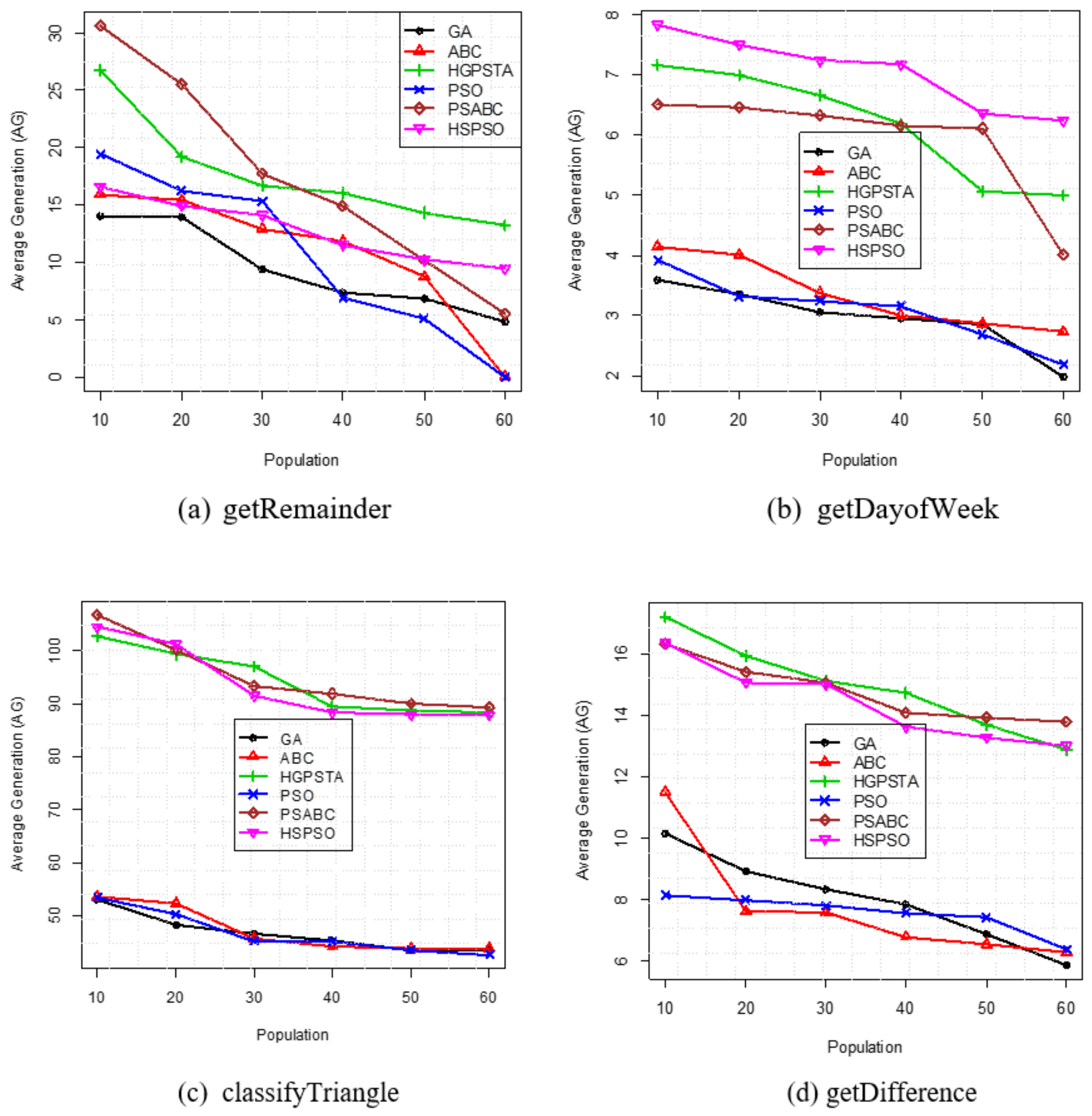


Figure 6. Average generations (AG) versus population size

algorithm that returned test cases that minimize the BCET within the class of single algorithms is GA, while PSO returned test cases that minimize the ACET and as well returned test cases that maximize the WCET. Similarly, within the hybrid algorithms, HGPSTA returned test cases that minimize the BCET, while HSPSO returned test cases that minimize the ACET and as well returned test cases that maximize the WCET. For getDifference program, the algorithm that returned test cases that minimize the BCET within the class of single algorithms is PSO, and it also returned test cases that minimize the ACET and ABC returned test cases that maximize the WCET. Similarly, within the hybrid algorithms, HSPSO returned test cases that minimize the BCET, ACET while HGPSTA returned test cases that maximize the WCET.

6. Conclusion

In this paper, a new metaheuristics approach for testing software using hybrid Harmony Search HS and Particle Swarm Optimization (HSPSO) was proposed. By way of introduction, different existing search-based testing algorithms were reviewed, and the current place within the sphere of non-functional properties is presented. Also, the main algorithm of HSPSO was presented, and its application in the generation of test data for four different programs was presented. The algorithm is implemented by defining fitness functions for branch coverage and execution time. While the coverage and BCET execution time are minimization optimization problems, the execution time in terms of WCET is a maximum optimization problem. The branch coverage results for all programs show that the HSPSO search-based testing is usable as it achieved 100% AC in all the entire programs used. Similar performances were observed in terms of execution times (BCET, ACET and WCET). The main limitation of the work is in terms of complexity of programs used, which ranges from simple to moderate. The generalizability performance of the HSPSO algorithm should be observed on more complex programs such as ATM or stopwatch to assess its usability and execution time.

Acknowledgement

This work was supported by Petroleum Technology Development Fund (PTDF), Nigeria by providing scholarship to N.M Bala.

REFERENCES

1. J. Kempka, P. McMinn, and D. Sudholt, Design and analysis of different alternating variable searches for search-based software testing, Theor. Comput. Sci., 2015.
2. C. Sharma, S. Sabharwal, and R. Sibal, A Survey on Software Testing Techniques using Genetic Algorithm, Nov. 2014.
3. D. R. Kuhn, D. R. Wallace, and A. M. Gallo, Software fault interactions and implications for software testing, IEEE Trans. Softw. Eng., 2004.
4. P. Zech, P. Kalb, M. Felderer, C. Atkinson, and R. Breu, Model-based regression testing by OCL, Int. J. Softw. Tools Technol. Transf., vol. 19, no. 1, pp. 115–131, 2017.
5. J. Meinicke, T. Thüm, R. Schröter, F. Benduhn, and G. Saake, An overview on analysis tools for software product lines, in ACM International Conference Proceeding Series, 2014.
6. L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, Non-Functional Requirements in Software Engineering, 2000.
7. M.Khari and P. Kumar, An extensive evaluation of search-based software testing: a review, Soft Computing, 23(6), 1933–1946.
8. B. Lindström, S. M. Ali, and M. Blom, Testability and Software Performance : A Systematic Mapping Study, ACM Symp. Appl. Comput., pp. 1566–1569, 2016.
9. C. Sharma, S. Sabharwal, R. Sibal, A. Hind, and F. Marg, A Survey on Software Testing Techniques using Genetic Algorithm, 2013.
10. B. Korel, Automated Software Test Data Generation, IEEE Trans. Softw. Eng., vol. 16, no. 8, pp. 870–879, 1990.
11. S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege, A systematic review of the application and empirical investigation of search-based test case generation, IEEE Transactions on Software Engineering. 2010.
12. W. Afzal, R. Torkar, and R. Feldt, A systematic review of search-based testing for non-functional system properties, Information and Software Technology, vol. 51, no. 6. pp. 957–976, 2009.
13. R. E. Lopez-Herrejon, L. Linsbauer, and A. Egyed, A systematic mapping study of search-based software engineering for software product lines, Information and Software Technology. 2015.
14. B. Xu, X. Xie, L. Shi, and C. Nie, Application of Genetic Algorithms in Software Testing, Adv. Mach. Learn. Appl. Softw. Eng., pp. 287–317, 2011.
15. M. A. Jamil, M. Arif, N. S. A. Abubakar, and A. Ahmad, Software testing techniques: A literature review, in Proceedings - 6th International Conference on Information and Communication Technology for the Muslim World, ICT4M 2016, 2017.
16. S. Varshney and M. Mehrotra, Search based software test data generation for structural testing, ACM SIGSOFT Softw. Eng. Notes, 2013.
17. M. Utting, A. Pretschner, and B. Legeard, A taxonomy of model-based testing approaches, Softw. Test. Verif. Reliab., vol. 22, no. 5, pp. 297–312, 2012.
18. A. Aleti and L. Grunske, Test data generation with a Kalman filter-based adaptive genetic algorithm, J. Syst. Softw., vol. 103, pp. 343–352, 2015.

19. G. Fraser and A. Arcuri, 1600 faults in 100 projects: automatically finding faults while achieving high coverage with EvoSuite, Empir. Softw. Eng., vol. 20, no. 3, pp. 611–639, 2015.
20. R. Matinnejad, S. Nejati, L. C. Briand, and T. Bruckmann, Effective test suites for mixed discrete-continuous stateflow controllers, in 2015 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE 2015 - Proceedings, 2015.
21. Z. Qin, G. Denker, C. Giannelli, P. Bellavista, and N. Venkatasubramanian, A software defined networking architecture for the internet-of-things, in IEEE/IFIP NOMS 2014 - IEEE/IFIP Network Operations and Management Symposium: Management in a Software Defined World, 2014.
22. A. Ram¨ªrez, J. R. Romero, and S. Ventura, An approach for the evolutionary discovery of software architectures, Inf. Sci. (Ny)., vol. 305, pp. 234–255, 2015.
23. Amarjeet and J. K. Chhabra, Harmony search based remodularization for object-oriented software systems, Comput. Lang. Syst. Struct., 2017.
24. X. Yao and D. Gong, Genetic algorithm-based test data generation for multiple paths via individual sharing, Comput. Intell. Neurosci., vol. 2014, 2014.
25. R. K. Sahoo, D. P. Mohapatra, and M. R. Patra, A Firefly Algorithm Based Approach for Automated Generation and Optimization of Test Cases, Int. J. Comput. Sci. Eng., vol. 4, no. 8, pp. 1–6, 2016.
26. A. Singh, N. Garg, and T. Saini, A hybrid Approach of Genetic Algorithm and Particle Swarm Technique to Software Test Case Generation, Int. J. Innov. Eng. Technol., vol. 3, no. 4, pp. 208–214, 2014.
27. A. Kiran and G. Radhamani, A Hybrid Model of Particle Swarm Optimization (PSO) and Artificial Bee Colony (ABC) Algorithm for Test Case Optimization, Int. J. Comput. Sci. Eng.(IJCSE), pp. 266–271, 2011.
28. O. R. Olaniran and M. A. A. Abdullah, Bayesian Variable Selection for Multiclass Classification using Bootstrap Prior Technique, Austrian Journal of Statistics, vol. 48, no. 2, pp. 63–72, 2019.
29. O. R. Olaniran and W. B. Yahya, Bayesian hypothesis testing of two normal samples using bootstrap prior technique, Journal of Modern Applied Statistical Methods, vol. 16, no. 2, pp. 618–638, 2017.
30. O. R. Olaniran and M. A. A. Abdullah, Subset Selection in High-Dimensional Genomic Data using Hybrid Variational Bayes and Bootstrap priors, Journal of Physics: Conference Series, IOP Publishing, vol. 1489, pp. 012030, 2020.
31. J. Popoola, W. B. Yahya, O. Popoola and O. R. Olaniran, Generalized Self-Similar First Order Autoregressive Generator (GSFO-ARG) for Internet Traffic, Stat., Optim. Inf. Comput., Vol. 8, pp. 0–11. September, 2020.
32. C. Mao, Generating Test Data for Software Structural Testing Based on Particle Swarm Optimization, Arab. J. Sci. Eng., 2014.
33. D. Weyland, A critical analysis of the harmony search algorithm-How not to solve sudoku, Oper. Res. Perspect., 2015.
34. K. Lakshmi and A. R. M. Rao, Multi-objective optimal design of composite box beam using hybrid adaptive harmony search with dynamically reconfigurable harmony memory, Proc. Inst. Mech. Eng. Part C J. Mech. Eng. Sci., 2019.
35. J. Shang, Y. Tian, Y. Liu and R. Liu, Production scheduling optimization method based on hybrid particle swarm optimization algorithm, Journal of Intelligent & Fuzzy Systems, Vol. 34 no.2, pp. 955–964, 2018.
36. T. Lengauer and R. E. Tarjan, A Fast Algorithm for Finding Dominators in a Flowgraph, ACM Trans. Program. Lang. Syst., 1979.
37. Y. Jia, W. Chen, J. Zhang and J. Li, Generating software test data by particle swarm optimization, Asia-Pacific Conference on Simulated Evolution and Learning. Springer, Cham, 2014.