https://doi.org/10.31449/inf.v42i3.1497 Informatica **42** (2018) 417–438 **417**

A Hybrid Particle Swarm Optimization and Differential Evolution Based Test Data Generation Algorithm for Data-Flow Coverage Using Neighbourhood Search Strategy

Sapna Varshney and Monica Mehrotra
Department of Computer Science, Jamia Millia Islamia, India
E-mail: sapna_varsh@yahoo.com, drmehrotra2000@gmail.com

Keywords: search based software testing, particle swarm optimization, differential evolution, data flow testing, dominance tree

Received: January 15, 2017

Meta-heuristic search techniques, mainly Genetic Algorithm (GA), have been widely applied for automated test data generation according to a structural test adequacy criterion. However, it remains a challenging task for more robust adequacy criterion such as data-flow coverage of a program. Now, focus is on the use of other highly-adaptive meta-heuristic search techniques such as Particle Swarm Optimization (PSO) and Differential Evolution (DE). In this paper, a hybrid (adaptive PSO and DE) algorithm is proposed to generate test data for data-flow dependencies of a program with a neighbourhood search strategy to improve the search capability of the hybrid algorithm. The fitness function is based on the concepts of dominance relations and branch distance. The measures considered are mean number of generations and mean percentage coverage. The performance of the hybrid algorithm is compared with that of DE, PSO, GA, and random search. Over several experiments on a set of benchmark programs, it is shown that the hybrid algorithm performed significantly better than DE, PSO, GA and random search in data-flow test data generation with respect to the measures collected.

Povzetek: Razvit je nov algoritem kot kombinacija hibridnega roja delcev in diferenčne evolucije z uporabo sosednje iskalne strategije.

1 Introduction

Software testing aims at assessing the quality and reliability of software product by detecting as many defects as possible. The cost of software testing increases exponentially with the size of input search space, thereby making manual testing a difficult and tedious task. There are software testing tools available with capture and playback features to automate the execution of test scripts. However, the test cases are manually selected by the human tester and may not be optimal. It is therefore desirable to generate optimal test data that reveals as many errors as possible according to a test adequacy criterion [1]. *Structural (white-box) testing* tests software for its structure and has the inherent capability to expose faults. The structural test adequacy criteria can be *statement coverage, branch coverage*, or *path coverage* that aim at executing every statement, branch or path respectively at least once. *Data-flow coverage*, an effective and robust test adequacy criterion, focuses on the definition and usage of variables in a program. Data-flow testing, therefore, could lead to more efficient and targeted test suites.

The attempts to reduce the cost of software testing by automating the process of software test data generation have been constrained by the ever increasing size and complexity of software. In the early period of automated test data generation, gradient descent and meta-heuristic search (MHS) algorithms such as Tabu Search, Hill Climbing and Simulated Annealing [2, 3, 4]. In the past two decades, evolutionary search-based algorithms such as Genetic Algorithm (GA) have been widely employed for test data generation as an effective alternative [5, 6, 7, 8, 9]. A search-based approach captures the test adequacy criteria as a fitness function that is used to guide the search. Due to an extensive application of search-based algorithms to test data generation problem, the approach has come to be known as Search Based Software Testing (SBST, coined by Harman and Jones). Recently, the focus is on the use of other highly adaptive search-based techniques such as Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO) and Differential Evolution (DE). It has been observed that GA and ACO have slow convergence towards the optimal solution. PSO and DE are conceptually very simple and the knowledge of previous good solutions is retained by all the members of the current population by means of constructive cooperation among them. PSO and DE have been found to be robust in solving optimization problems; however, the performance depends on control parameters. PSO has been shown to be well suited for test data generation with better performance than GA [10, 11, 12, 13, 14]. Hybridization of search-based algorithms for test data generation has also been reported in literature. GA with a local search algorithm [15] and more recently, GA with

PSO has been applied for test data generation in some studies [16, 17, 18, 19, 20, 21].

In this study, we propose a hybrid global search algorithm by combining an adaptive PSO with DE mutation operator to automatically generate test data for data-flow dependencies of a program. In the proposed hybrid algorithm, a new term based on DE differential operator is included for velocity update in PSO for some additional exploration capability. The greedy selection scheme of DE is used wherein position of a particle is updated only if it yields a better fitness value. This results in movement of particles only to better locations in the input search space. A local neighborhood strategy is also included in the proposed hybrid algorithm to explore more promising candidate solutions and overcome the problem of boundary constraints. Design of the fitness function [22] is based on dominance concepts and branch distance that is used to guide the search for optimal test data for data-flow dependencies of a program. The performance of the proposed hybrid algorithm is compared with that of DE, PSO, GA and random search. It is demonstrated that the proposed hybrid algorithm outperformed DE, PSO, GA and random search in terms of mean percentage coverage achieved, and mean number of generations to produce the final test suite for data-flow coverage of a program.

The rest of the paper is organized as follows: Section 2 provides a brief description of automated software test data generation process and related work. Section 3 provides an overview of data-flow analysis. Sections 4 and 5 provide a brief description of PSO and DE algorithms. Section 6 describes the proposed hybrid algorithm. Section 7 gives the experimental results. Section 8 provides the discussion and the detailed statistical analysis of the experimental results. Section 9 deals with threats to validity and limitations of the proposed hybrid algorithm. Finally, section 10 gives the conclusion.

2 Related work

This section presents the methods to generate test data for software structural testing and the related literature. ***Symbolic execution,*** a static method, has been employed for test data generation [2]; however, the performance is constrained by programming constructs such as pointers, loop conditions with input variables, array subscripts and procedure calls [23]. Dynamic methods that have been employed for test data generation can be classified as *random*, *path-oriented* and *goal-oriented* techniques [9, 23]. A ***random test data generator*** arbitrarily selects test data from the input domain. Though easy to implement, it may fail to find optimal test data. ***Path-oriented test data generator*** [5] uses control flow information to identify a set of independent paths to generate test data. However, it does not work well with infeasible paths or paths that contain loops. A ***goal-oriented test data generator*** [9, 23, 24] generates test data for a selected goal such as a statement or a branch, irrespective of the path taken.

The meta-heuristic search techniques guided by a *fitness function* have been adopted to generate optimal test data mainly according to a structural test adequacy criterion. From the literature on structural test data generation, it can be inferred that branch coverage and path coverage are the most often used and well-understood measures [25]. For branch coverage, fitness values are calculated by finding approximation level and branch distance for a target branch from control flow graph [8, 26]. Data-flow coverage criterion has not been used much [27] due to difficulty in writing test cases that satisfy data-flow dependencies of a program. Wegener et al. [28] defined different types of fitness functions for structural testing; data-flow test criteria being classified as node-node-oriented methods. Recently only there has been more work on search based test data generation for data-flow coverage using GA as the algorithm of choice [6, 7, 22, 24, 29, 30]. Now, other highly adaptive search-based techniques such as PSO [14, 18] and ACO [31] are also being applied to generate test data for data-flow coverage due to simplicity and faster convergence. ACO [32] and Harmony Search [33] has also been applied to generate structural test data for branch coverage.

Vivanti et al. [30] have proposed a GA-based technique for data-flow coverage evaluated on open source Java applications. The results have indicated the scalability and applicability of data-flow criteria for test data generation.

In our previous work [22], an elitist GA-based approach is proposed to generate test data for data-flow dependencies of a program using dominance concepts and branch distance. The fitness function is derived from the work by Ghiduk et al. [6]; it is augmented with branch distance to produce a smoother landscape for guiding the search and also takes into account that a definition may be killed by another definition before the associated use is reached. The performance of the proposed approach is compared with random search and earlier studies on test data generation for data-flow dependencies of a program by Girgis [7], Ghiduk et al. [6] and Girgis et al. [21]. The proposed GA-based approach guided by the novel fitness function outperformed random search and the earlier studies [6, 7, 21] to generate test data for data-flow coverage of a program.

Windisch et al. [10] applied PSO to artificial and complex industrial test objects to generate test data for branch coverage. Their results showed efficiency and efficacy of PSO over GA for most code elements to be covered.

Agarwal et al. [11] applied binary PSO, Agarwal and Srivastava [12] applied discrete quantum PSO and Mao [13] applied standard PSO to generate test data for branch coverage test adequacy criterion.

Nayak and Mohapatra [14] proposed an algorithm to generate test cases using PSO for data flow coverage. This technique cannot rank test cases because the fitness function, as simply taken from Girgis [7], assigns the same fitness value to all the test cases that cover the same number of test requirements and a fitness value of 0 to all the test cases that do not cover any test requirement or

cover a partial aim. Here, the fitness function is unable to guide the search.

Application of hybrid algorithms have also been studied for test data generation problem. Zhang et al. [16] proposed a hybrid algorithm (GA and PSO) to generate test data for path coverage. GA and PSO operations are applied to two population sets. Triangle classification problem is taken as the case study and the hybrid algorithm is compared with GA and PSO. The hybrid algorithm is shown to be better than GA and PSO with respect to number of iterations. The average time taken is found to be more than PSO but less than GA. Their hybrid technique is complicated and may generate redundant test cases for automatic test data generation.

Li et al. [17] also proposed a hybrid algorithm (GA and PSO) to generate test data for path coverage. PSO equations to update particle's velocity and position distance are used instead of mutation operator of GA. The algorithm is applied only to the triangle benchmark problem.

Singla et al. [18] applied a hybrid algorithm (GA and PSO) to generate test data for data-flow coverage. The fitness function used is same as in [6]; it does not take into account the traversal of killing nodes as well as closeness of test data in case if only partial aim is covered. The strategy is tested only on some simple programs.

Kaur and Bhatt [19] proposed a hybrid algorithm (GA and PSO) to prioritize test data in regression testing. The algorithm has been tested on few simple programs.

Girgis et al. [21] proposed a hybrid Genetical Swarm Optimization (GSO) Technique to generate a set of test paths that cover the all-uses criterion for data-flow coverage. The authors have claimed that the set of paths generated by the proposed GSO can be passed to a test data generation tool to find program inputs that will execute them to complete the data flow paths testing of the program under test. The fitness function used is same as in [7]; it is not able to guide the search and results in loss of valuable information in case if only partial aim is covered.

Chawla et al. [20] proposed a hybrid PSO and GA algorithm for automatic generation of test suites with branch coverage as the test adequacy criterion. The experiments are performed with ten Java container classes. The algorithm is shown to perform better than GA, PSO and existing hybrid strategies based on GA and PSO.

Each optimization algorithm has its own advantages and disadvantages. Also, one optimization algorithm will not work well for all the optimization problems. DE, a meta-heuristic search-based algorithm, has been applied to several optimization problems [34, 35] to demonstrate its potential. Das et al. [36] has explored hybridization of PSO with DE applied to the design of digital filters. However, DE has not been applied for test data generation and optimization problem [25, 27, 37].

The proposed study will focus on the application of a hybrid adaptive PSO-DE algorithm to generate test data for data-flow dependencies of a program. The proposed hybrid global search algorithm combines the evolution scheme of both PSO and DE incorporating the best of both the algorithms in the context of test data generation. A new term based on DE differential operator is included for velocity update in PSO. The greedy selection scheme of DE is also used wherein position of a member is updated only if it yields a better fitness value. The hybridization scheme has resulted in movement of particles only to better locations in the input search space. The design of fitness function [22] is based on the dominance relations between the nodes of a program's control flow graph augmented with branch distance which produces a smoother landscape for guiding the search. This leads to faster and better convergence of test data to achieve the desired coverage. A neighborhood search strategy is also incorporated into the proposed hybrid algorithm that further helps in overcoming the problem of boundary constraints and local optima by exploring more promising candidate solutions. This is the main contribution of this paper. The proposed hybrid algorithm generates test data for one test requirement at a time; other test requirements are also checked for coverage thereby reducing the overall number of fitness evaluations.

3 Data flow analysis

In this study, data-flow coverage is used as the test adequacy criteria. Data-flow analysis [38] augments the control-flow testing criteria; the emphasis is on the definition and use of the variables in a program. The control flow of a program is represented by a directed graph G (V, E) also known as control flow graph (CFG), where V is the set of all the nodes and E is the set of all the edges in the graph. Each node corresponds to a program statement or group of sequential program statements and an edge represents flow of control from one node to another. There are two distinct nodes: an entry node n_0 and an exit node n_{end}. Node n dominates node m (dominance relationship) if every path from entry node n_0 to m contains n. By applying dominance relationship to all the nodes of CFG, a tree can be obtained that is rooted at n_0. This tree is called the dominator tree [39]. For each node m in the CFG, Dom (m) is the set of all the nodes that dominate node m. Figure 2 gives the CFG of the example program as given in Figure 1. The dominator tree is shown in Figure 3. For example, Dom (12) = {1, 2, 6, 7, 12}.

In a program, the definition and use occurrences of each variable are identified. A variable is said to be defined in a program statement (def-node) if a value is associated with the variable. A variable is said to be used in a program statement if its value is referenced for computational use (c-use node) or a predicate use (p-use node). Data-flow testing should cause the traversal of def-clear sub-paths from the variable definition to either some or all of the p-uses, c-uses, or their combination. Empirically, the all-uses criterion has been shown to be most effective compared to the other data-flow criteria [40]. A def-clear path does not include any intermediate nodes containing other definitions of that variable (killing nodes). A def-clear path can be further

```
#include<stdio.h>
#include<conio.h>

1     1  void main() {
2     1  int a, b, c, valid;
3     1  printf("\nEnter the value of three sides: ");
4     1  scanf("%d %d %d", &a, &b, &c);
5     1  valid=0;
6     2  if((a>=0)&&(a<=100)&&(b>=0)&&(b<=100)&&(c>=0)
           &&(c<=100)) {
7     3                  if(((a+b)>c)&&((c+a)>b)&&((b+c)>a)) {
8     4                        valid=1;
9     5                  }
10    5     }
11    6     if (valid==1) {
12    7                  if ((a==b)&&(b==c))
13    8                        printf("\nEquilateral triangle.");
14    9                  else if ((a==b)||(b==c)||(c==a))
15    10                       printf("\nIsosceles triangle.");
16    11                 else
17    11                       printf(("\nScalene triangle.");
18    12    } else {
19    13                 printf("\n Invalid input ").;
20    14    }
21    15 }
```

Figure 1: Triangle classification program.

Table 1: List of variables and def-use occurrences in the example program

Variable	def Node	c-use Node	p-use Edge
a b c	1	None	2-3 2-6 3-4 3-5 7-8 7-9 9-10 9-11
valid	1,4	None	6-7 6-13

Table 2: List of def-use paths for the example program.

Path No.	def-use Path (Terminates with -1 for c-use)	Killing Node(s)
1	1-2-3	None
2	1-2-6	None
3	1-3-4	None
4	1-3-5	None
5	1-7-8	None
6	1-7-9	None
7	1-9-10	None
8	1-9-11	None
9	1-6-7	4
10	1-6-13	4
11	4-6-7	None
12	4-6-13	None

categorized as a dcu-path (c-use of the variable) or a dpu-path (p-use of the variable). For the example program, Table 1 provides definition and use nodes for each variable, Table 2 provides the list of all-def-use paths and

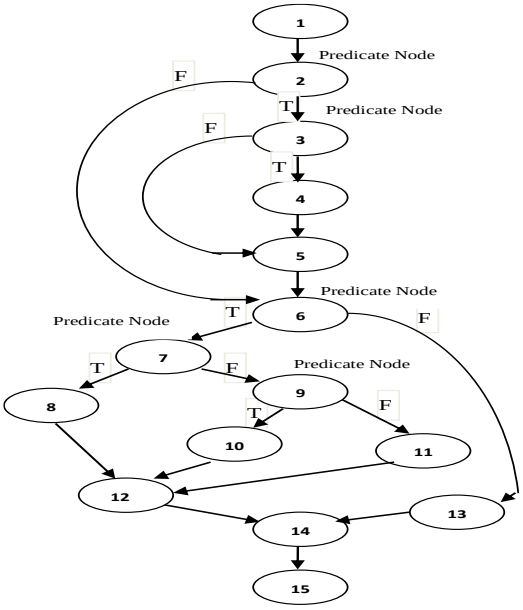


Figure 2: CFG of the example program.

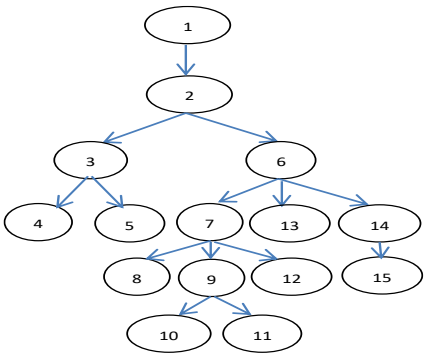


Figure 3: Dominator tree for the example

Table 3: Dominance paths for the nodes of the CFG.

Node No.	Dominance Path
1	1
2	1-2
3	1-2-3
4	1-2-3-4
5	1-2-3-5
6	1-2-6
7	1-2-6-7
8	1-2-6-7-8
9	1-2-6-7-9
10	1-2-6-7-9-10
11	1-2-6-7-9-11
12	1-2-6-7-12
13	1-2-6-13
14	1-2-6-14
15	1-2-6-14-15

Table 3 provides the dominance paths for the nodes of the program flow graph.

4 Particle swarm optimization

In 1995, Kennedy and Eberhart [41] introduced Particle Swarm Optimization algorithm, a population-based search algorithm based on the social and cognitive behavior of different swarms such as flock of birds, herd of animals or school of fishes. The application of PSO for solving many continuous space problems in the field of Computer Science and Engineering has demonstrated its potential. Unlike GA, PSO does not use evolution operators such as crossover and mutation. Instead, each member of the swarm (called particle) attains optimal solution by learning from its own experience and the experience of other members of the swarm. Each particle maintains its current position, current velocity and the best position it has achieved so far, called pbest. The global best position of the swarm is called gbest. Both pbest and gbest are used by the particle in determining its next best position in the swarm. Thus, the knowledge of previous good solutions is retained by all the particles resulting in a faster convergence towards the optimal solution.

Consider a swarm of *n* particles denoted as (p_1, p_2... p_n). Position of the i^{th} particle in the d-dimensional search space is denoted as $X_i = (Xi^1, X_i^2 \ldots X_i^d)$ and the associated velocity is denoted as $V_i = (V_i^1, V_i^2 \ldots V_i^d)$. The personal best position of the i^{th} particle in dimension *d* is denoted as $pbest_i^d$. The position of the best particle of the entire swarm in dimension *d* is denoted as $gbest^d$. The velocity and position of the i^{th} particle in dimension *d* can be updated by Equations 1 and 2 as given below.

$$V_i^d = w \times V_i^d + c_1 \times r_1 \times (pbest_i^d - X_i^d) + c_2 \times r_2 \times (gbest^d - X_i^d) \quad (1)$$

$$X_i^d = X_i^d + V_i^d \quad (2)$$

where, c_1 and c_2 are positive learning constants called cognitive and social scaling parameters chosen in such a way that their sum never exceeds 4, and r_1 and r_2 are two random numbers in the range [0,1]. The inertia weight *w* controls the impact of the previous history on the new velocity of the i^{th} particle. A particle's velocity in each dimension is clamped to a maximum magnitude V_{max}. The position and velocity of each particle in the swarm are continuously updated until an optimal solution is achieved.

4.1 Adaptive inertia weight

In PSO algorithm, a large value of inertia weight facilitates *exploration* (global search) of the input search space and a small value of inertia weight facilitates *exploitation* (local search) of the input search space for the optimal solution. Various inertia weighting strategies used in the literature have been categorized into *constant*, *random*, *time varying* and *adaptive* inertia weight strategies [42]. In constant and random inertia weight strategies, value of inertia weight is either constant or is chosen randomly during the search. In time varying inertia weight strategies, inertia weight is defined as a function of time or iteration number. Here, value of inertia weight is independent of the state of the particles in the search space. In adaptive inertia weight strategies, state of the particles in the search space (feedback mechanism) is used to adjust the value of the inertia weight.

In this study, fitness value of the particles is used to adjust the inertia weight. Ratio α of the particle's fitness to the average fitness of the swarm is calculated as shown in Equation 3 below:

$$\alpha = f_i / f_{max} \quad (3)$$

Here, f_i=fitness of i^{th} particle and f_{max} is the maximum fitness achieved by the particles in the swarm.

The range of α is [0, 1]. For lower values of α, increasing inertia weight can strengthen the particle's search capability. For values of α that are closer to 1, smaller inertia weight should be used. The inertia weight w_i for the i^{th} particle is therefore defined as a linear function of α and is calculated as follows:

$$w_i = 0.5 \times (1-\alpha) + 0.5 \quad (4)$$

The range of the inertia weight is [0.5, 1].

PSO is computationally inexpensive. The ability of PSO to balance between local exploitation and global exploration of the search space enhances searching ability and avoids premature convergence towards the optimal solution.

5 Differential evolution

Differential Evolution (DE) algorithm was given by Storn and Price [43] in 1995. It is a stochastic population-based global optimization algorithm that uses an evolutionary differential operator to create new offspring from parent chromosomes. Unlike GA, DE works upon real-valued chromosomes. The differential operator of DE replaces the classical crossover and mutation operators of GA.

Let's say, the initial population consists of *n* vectors denoted as (p_1, p_2... p_n). Position of the i^{th} vector in the d-dimensional space is denoted as $X_i = (Xi^1, X_i^2 \ldots X_i^d)$. These vectors are referred as chromosomes in DE. To change each chromosome *(target vector)*, a *difference vector* V_i is created. In the literature, there are various mutation schemes to create this vector. In this paper, DE/Rand/1 scheme is used. In this scheme, for each i^{th} member X_i of the current population, three other members (say r_1, r_2 and r_3) are randomly chosen from the current population. Next, the scaled difference (mutation scaling factor F) of any two of the three vectors is added to the third one to obtain the difference vector V_i. The j^{th} component of the difference vector is as given below:

$$v_{i,j} = x_{r1,j} + F \times (x_{r2,j} - x_{r3,j}) \quad (5)$$

To increase the population diversity, a 'crossover scheme' is applied. The difference vector exchanges its components with the target vector X_i to obtain the offspring/trial vector U_i. The most common crossover in DE is 'uniform crossover' as given below:

$$\begin{aligned} u_{i,j} &= v_{i,j} && \text{if } \mathrm{rand}(0,1) < CR \\ &= x_{i,j} && \text{else} \end{aligned} \quad (6)$$

CR is called the crossover constant.

The final step in DE algorithm is the fitness-based selection of either target vector or trial vector in the next generation. F and CR are the control parameters of DE. The performance of DE depends on the manipulation of *target vector* and *difference vector* in order to obtain a *trial vector*.

6 Proposed hybrid algorithm

In the proposed study, an adaptive PSO algorithm is hybridized with the DE algorithm incorporating local neighborhood search strategy. The synergy between PSO and DE algorithms has resulted in a more powerful global search algorithm. The local neighborhood search strategy helps in exploring more promising candidate solutions to overcome the problem of local optima.

In the proposed hybrid (adaptive PSO and DE) algorithm, a differential velocity term inspired by the DE mutation scheme is computed by taking the difference of the position vectors of any two distinct particles randomly chosen from the swarm. A random number r is generated between 0 and 1. If r is less than DE crossover probability, Equation 7 (given below) is used to update the velocity of a particle. In Equation 7, the cognitive term (second term) in Equation 1 is replaced by the differential term scaled by DE mutation scaling factor.

$$V_i^d = w \times V_i^d + F \times (x_j^d - x_k^d) + c_2 \times r_2 \times (gbest^d - X_i^d) \quad (7)$$

Here, x_j and x_k denote the position of particles j and k respectively ($i \neq j \neq k$) that are randomly chosen from the swarm. A survival of the fittest mechanism is also followed by incorporating the greedy selection scheme of DE as given by Equation 6. Therefore, the particle either moves to a better location or remains at its previous position in the input search space. The current position of a particle will always be its best position.

The steps of the proposed hybrid (adaptive PSO and DE) algorithm are given in Figure 5. The flowchart is given in Figure 6. Inputs to the algorithm are an instrumented program, dominator tree of the program, list of def-use paths to be traversed and the killing nodes if any, number of input variables, domain range of each input variable, and the algorithmic parameters: population size, PSO acceleration parameters, PSO maximum velocity, DE mutation scaling factor and DE crossover probability. Adaptive inertia weight is used as given by Equations 3 and 4. For data-flow coverage criterion, the design of fitness function is explained in Section 6.2 below. Initial value of *pbest* and *gbest* is 0. The algorithm is run once for each uncovered def-use path. If the selected path is not covered by any member of the current population, fitness value is computed for each member. Accordingly, for each particle, the personal best position *pbest* and the global best position *gbest* can be updated. During the evolution process, particle's position and velocity is adjusted according to Equations 2 and 7 respectively. If the updated position of the particle is out of input domain range, a local neighbourhood strategy is applied. Then, the greedy selection scheme of DE is used to generate the new population. The evolution process continues until the termination criteria is met. The other uncovered paths are also checked for coverage. The output is an optimal test suite and a list of def-use paths marked as covered or uncovered, if any.

A tool is developed for instrumenting programs and to generate def-use paths. Dominator tree is generated manually. Infeasible paths, if any, are determined by careful analysis of the program.

6.1 Neighbourhood search strategy

Every meta-heuristic search algorithm suffers with the problem of local optima. Another issue related to meta-heuristic search algorithms is boundary constraints. There are no set mechanisms to deal with such problems. Hence, in this study, an effort is also made to handle the problems of local optima and boundary constraints and to improve the exploitation ability of the algorithm. A neighbourhood search strategy (Figure 4) is introduced to sample more promising candidate solutions to overcome these problems. It is summarized as follows:

Step 1: For each particle, Euclidean distance is calculated from the other particles in the input search space using the position of particles. Accordingly, other particles within a threshold Euclidean distance (determined by preliminary study to fine-tune the algorithmic parameters) form the neighbourhood. Euclidean distance between two particles X_i and X_j in the n-dimensional search space is given by the following equation:

$$d_{ij} = \sqrt{\sum_{k=1}^{n} (x_{ik} - x_{jk})^2} \quad (8)$$

Step 2: If a particle's new position is out of range, other particles in the neighbourhood are evaluated.

Step 3: The position of the particle is then replaced with that of the best particle in the neighbourhood instead of a random value.

This helps in exploring more promising candidate solutions.

6.2 Design of Fitness Function

Def-use associations can be represented as node-node fitness functions [28]. Def-use associations specify the node of definition and the node of use for the program variables in the CFG without specifying a concrete path between the nodes. This implies that the first objective to reach is the definition node and then the use node, without however, specifying a path through the CFG. The distance to a node is represented by the standard minimizing metric given below:

$$\text{node distance} = \text{approach level} + v(\text{branch distance}) \quad (9)$$

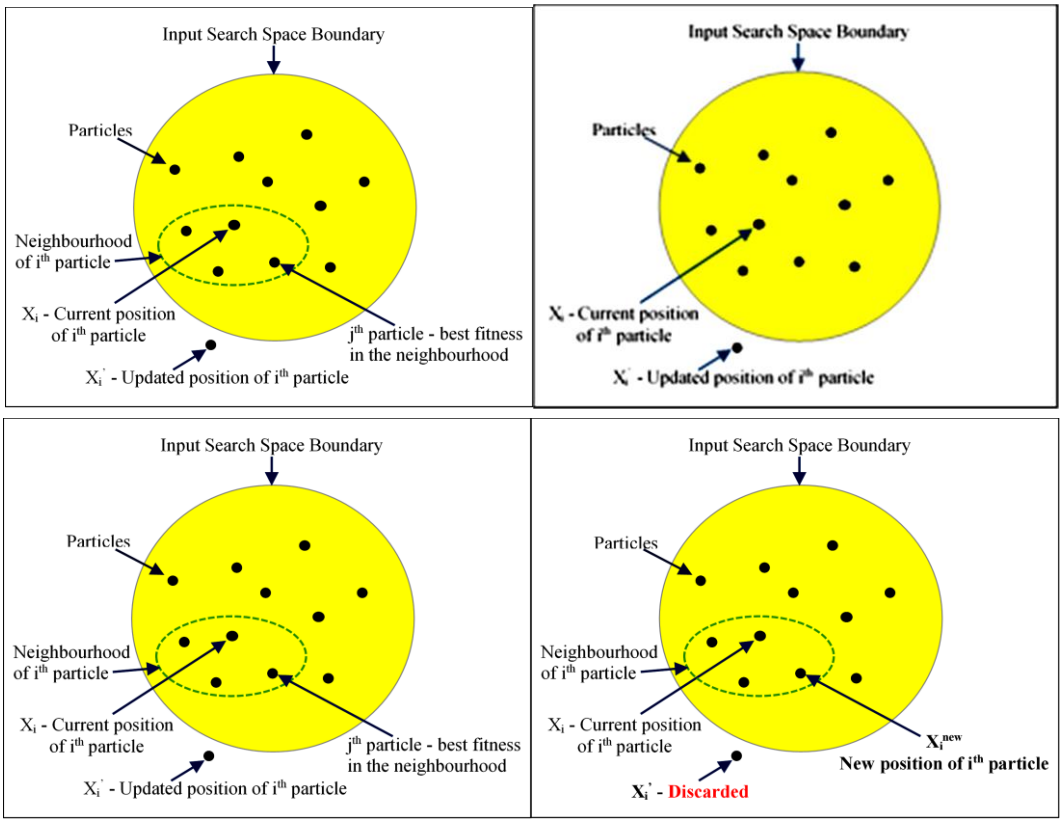


Figure 4: Local Neighbourhood Strategy.

It evaluates to 0 if the target is covered. Approach level is the closest point (a node) of a given execution to the target node. A branch is said to be *critical* if it leads the program execution away from the target node in a path through the program structure [44]; *branch distance* is calculated at that particular predicate node using values of the variables according to the formulae given in Table 4 [3] below.

Table 4: Branch distance measure for relational and logical predicates.

S. No.	Predicate (C)	Branch Distance Formulae: f(C)
1	Boolean	if true then 0 else K
2	x = y	if (x-y)=0 then 0 else abs(x-y)+K
3	x ≠ y	if abs(x-y)≠0 then 0 else K
4	x > y	if (y–x)<0 then 0 else (y-x)+K
5	x ≥ y	If (y–x)≤0 then 0 else (y-x)+K
6	x < y	if (x–y)<0 then 0 else (x-y)+K
7	x ≤ y	if (x–y)≤0 then 0 else (x-y)+K
8	C1 && C2	f(C1) + f(C2)
9	C1 || C2	min(f(C1), f(C2))

K is a failure constant that is added to branch distance if predicate is false

Branch distance provides a measure of how close the program execution was to traverse the alternate edge of the critical branch. Branch distance is normalized in the range [0, 1] using a normalization function *v*, such that the approach level always dominates the branch distance.

In our previous study [22], a novel maximizing fitness function is proposed for data-flow coverage adequacy criterion based on the standard metric (Equation 9) and dominator tree. Dominance relations between the nodes of the CFG are used to obtain path-cover for the nodes of the selected def-use path. The fitness function considers each def-use path as two objectives. For a dcu-path, the first objective is to cover the dominance path of the *definition* node and then to cover the dominance path of the *use* node. For a dpu-path, the first objective is to cover the dominance path of the *definition* node and then to cover the dominance paths of the nodes of the p-use *edge* (u_1, u_2). A dpu-path is formed for both the branches (T/F) of the predicate node. A test case is evaluated with respect to the selected def-use path by executing the program under test with it as an input and recording the nodes that are covered. If a killing node is traversed between the source node and the use node, a fitness value of 0 is assigned to the test case and it is discarded. The fitness value is 1 if all the nodes of the dominance paths of both the objectives are covered; otherwise closeness of the test case to the missed objective (*branch distance*) is computed.

In this work, for fitness maximization, branch distance *bch(x, t_i)* at the critical branch for test case t_i and target node x is the reciprocal of the value returned by an appropriate formula from Table 4 i.e. the closer a test case is to cover the required branch, higher is its fitness value. The fitness function uses control-flow information (dominance relations between the nodes of the CFG) augmented with branch distance if a partial aim is achieved. This provides a smoother landscape/guidance to the search process towards the optimal solution. Branch distance is computed using Equation 10 and the

fitness functions are given by Equations 11 and 12 as explained below.

Branch distance *bch (x, t_i)* for test case t_i *(i=1...p)* and target node *x*, for fitness maximization, is calculated as follows:

$$\text{bch}(x, t_i) = \begin{cases} 1 & \text{if the test case } t_i \text{ leads to the target node x} \\ \frac{1}{f(C)} & \text{otherwise, using an appropriate formula from Table 4 for the predicate C at the critical branch} \end{cases} \quad (10)$$

The fitness function to evaluate the fitness of a test case t_i (i=1...p) w.r.t. a dcu-path (d, u, v), where d is the *definition node* and u is the *c-use node* of a variable v, is given below:

$$\text{ft}(d, u, t_i) = \frac{1}{2} \times \left(\frac{|\text{cdom}(d, t_i)|}{|\text{dom}(d)|} \times \text{bch}(d, t_i) + \frac{|\text{cdom}(u, t_i)|}{|\text{dom}(u)|} \times \text{bch}(u, t_i) \right) \quad (11)$$

Similarly, the fitness function to evaluate the fitness of a test case t_i (i=1...p) w.r.t. a dpu-path (d, (u_1, u_2), v), where d is the *definition node* and (u_1, u_2) is the *p-use edge* of a variable v, is given below:

$$\text{ft}(d, (u_1, u_2), t_i) = \frac{1}{3} \times \left(\frac{|\text{cdom}(d, t_i)|}{|\text{dom}(d)|} \times \text{bch}(d, t_i) + \frac{|\text{cdom}(u_1, t_i)|}{|\text{dom}(u_1)|} \times \text{bch}(u_1, t_i) + \frac{|\text{cdom}(u_2, t_i)|}{|\text{dom}(u_2)|} \times \text{bch}(u_2, t_i) \right) \quad (12)$$

In general,

- dom(x): set of nodes in the dominance path of the target node *x*
- cdom(x, t_i): set of nodes in dom(x) that are covered by test case t_i *(i=1...p)*
- bch(x, t_i): branch distance for test case t_i *(i=1...p)* and target node x using Equation 9

If a killing node is traversed, a fitness value of 0 is assigned to the test case t_i and it is discarded; otherwise Equation 11 or Equation 12 is used to compute the fitness value. Test case t_i is said to be optimal if its fitness value is 1 i.e. the target is covered.

Consider the def-use path# 5 (1, 7, 8) for coverage from Table 2. This is a dpu-path that tests for 'Equilateral triangle' condition. Node 1 (source) and the p-use edge (7, 8) (target) form the two objectives - their dominance paths to be covered by an input test case. There are three cases - if the dominance paths of both the nodes are covered, fitness value of the input test case is 1 and it is optimal. However, if a partial aim is covered (one of the two nodes) or none of the nodes is covered, fitness value of the input test case is computed using Equations 3.2 and 3.4.

From Table 3, the dominance paths of the nodes are as given below:

dom(d) = dom(1) = {1}

dom(u1) = dom(7) = {1, 2, 6, 7}

dom(u2) = dom(8) = {1, 2, 6, 7, 8}

Case 1: Input test case t_1 <2, 2, 2>

Path traversed {1, 2, 3, 4, 5, 6, 7, 8, 12, 15}

Dominance path of the definition node (node 1) is covered.

Dominance path of the first node of the p-use edge (node 7) is covered.

Dominance path of the second node of the p-use edge (node 8) is covered.

As the dominance paths of both the objectives are covered, the fitness value of the input test case using Equation 3.4 is 1; the input test case t_1 is therefore optimal.

Case 2: Input test case t_2 <2, 2, 1>

Path traversed {1, 2, 3, 4, 5, 6, 7, 9, 10, 12, 15}

Dominance path of the definition node (node 1) is covered.

Dominance path of the first node of the p-use edge (node 7) is covered.

Dominance path of the second node of the p-use edge (node 8) is not covered; the critical node is node 7. The branch distance at node 7 using Equation 3.2 is bch (8, t_2) = 0.91

The fitness value of the input test case using Equation 3.4 is ft (1, (7, 8), t_2) = 0.91

Case 3: Input test case t_3 <1, 2, 4>

Path traversed {1, 2, 3, 5, 6, 12, 13, 14, 15}

Dominance path of the definition node (node 1) is covered.

Dominance path of the first node of the p-use edge (node 7) is not covered; the critical node is node 6. The branch distance at node 6 using Equation 3.2 is bch (7, t_3) = 0.91

Dominance path of the second node of the p-use edge (node 8) is not covered; the critical node is node 7. The branch distance at node 6 using Equation 3.2 is bch (8, t_3) = 0.91

The fitness value of the input test case using Equation 3.4 is ft (1, (7, 8), t_3) = 0.74

This case study shows that the input test case t_1 covers the selected def-use path# 5. The input test case t_2 covers the def node and the first node of the selected def-use path# 5 (partial aim). The input test case t_3 does not cover any of the two objectives for the selected def-use path# 5. Accordingly, ft (1, (7, 8), t_1) > ft (1, (7, 8), t_2) > ft (1, (7, 8), t_3). Thus, the input test cases are also ranked according to their fitness values.

7 Experimental setup

In this section, research questions, algorithmic parameters settings, details of the subject programs, and experimental results are provided. DE, PSO, GA and random search techniques are also implemented for comparison with the proposed hybrid (adaptive PSO and DE) algorithm.

7.1 Research questions

The following research questions are formulated to evaluate the performance of the proposed hybrid algorithm:

RQ1: How effective is the proposed hybrid (adaptive PSO and DE) algorithm for optimal test data generation to achieve 100% data-flow coverage of a program?

```
Algorithm ATDG_Hybrid_PSO_DE
Input:
    P            : Instrumented version of the program under test
    arg = (a1,a2,…,ad): Argument list of P encoded into a d-dimension position vector
    DT           : Dominator tree for the program P
    Paths        : List of test requirements i.e. def-use paths
    Popinit      : Initial random population of n particles Xi = [Xi1, Xi2…Xid] and their velocities V = [Vi1, Vi2…Vid] for i=1, 2…n
    c1, c2, Vmax : Algorithmic parameters of Particle Swarm Optimization (PSO) algorithm
    F, CR        : Algorithmic parameters of Differential Evolution (DE) algorithm
Output:
    TestSuite    : Set of optimal test cases
    Pathstat     : List of test requirements marked as 'covered' and 'could not be covered' (if any)
Begin
1.  Popold = Popinit
2.  Popcur = Popinit
3.  while some pathi in Paths is not marked {
4.      while (termination criterion is not met) {      //Either pathi is covered or MaxAttempts
5.          for each particle i of Popcur {
6.              Decode position vector Xi into a test case ti
7.              if pathi is not marked {
8.              Check pathi for coverage w.r.t. ti and calculate fitness value using Eq. 10 or Eq. 11
9.                  if pathi is covered {
10.                     Mark pathi as 'covered' (update Pathstat)
11.                     Add ti to TestSuite
12.                 }
13.             }
14.             for each pathj of TestReq other than pathi that is not marked {
15.                 Check pathj for coverage with respect to ti
16.                 if pathj is covered
17.                     Mark pathj as 'covered' (update Pathstat)
18.             }
19.         }
20.         if pathi is covered
21.             Go to line 3
22.         else {
23.             Update gbesti^j
24.             for each particle i of Popcur {          //Generate a new population Popnew
25.                 Calculate inertia weight w using Equations 3 and 4
26.                 Randomly choose two distinct particles k and l from Popcur (i≠k≠l)
27.                 for each dimension j (1≤j≤d) of particle i{
28.                     Update pbesti^j
29.                     Randomly generate r between 0 and 1
30.                     if r<CR{
31.                         Calculate the difference between the jth components of the position vectors of particle k and particle l
32.                         Update velocity Vi^j of particle i in dimension j using Eq. 7
33.                         Clamp velocity Vi^j within the range [-Vmax, Vmax]
34.                     }
35.                     Update position Xi^j of particle i in dimension j using Eq. 2          //Offspring
36.                     if new position Xi^j of particle i in dimension j is out of range {
37.                         Apply neighbourhood strategy to particle i - according to Euclidean distance (Eq.8)
38.                         New position Xi^j of particle i in dimension j is the position of the best particle in the neighbourhood
39.                     }
40.                 }
41.                 Calculate fitness value of Offspring using Eq. 10 or Eq. 11
42.                 if Offspring is better than the parent Xi
43.                     Include Offspring in new population Popnew
44.                 else
45.                     Include parent Xi in new population Popnew
46.             }
47.             Popold = Popcur
48.             Popcur = Popnew
49.         }
50.     }
51.     if selected pathi could not be covered
52.         Mark pathi as 'could not be covered'
53. }
54. Return TestSuite, Pathstat
End
```

Figure 5: Proposed hybrid (adaptive PSO and DE) test data generation algorithm.

RQ2: How effective is the proposed hybrid (adaptive PSO and DE) algorithm for optimal test data generation with respect to the convergence speed (mean number of generations) at termination?

7.2 Parameters tuning

A preliminary study was carried out to determine the appropriate value of the algorithmic parameters and threshold value for Euclidean distance. Population sizes

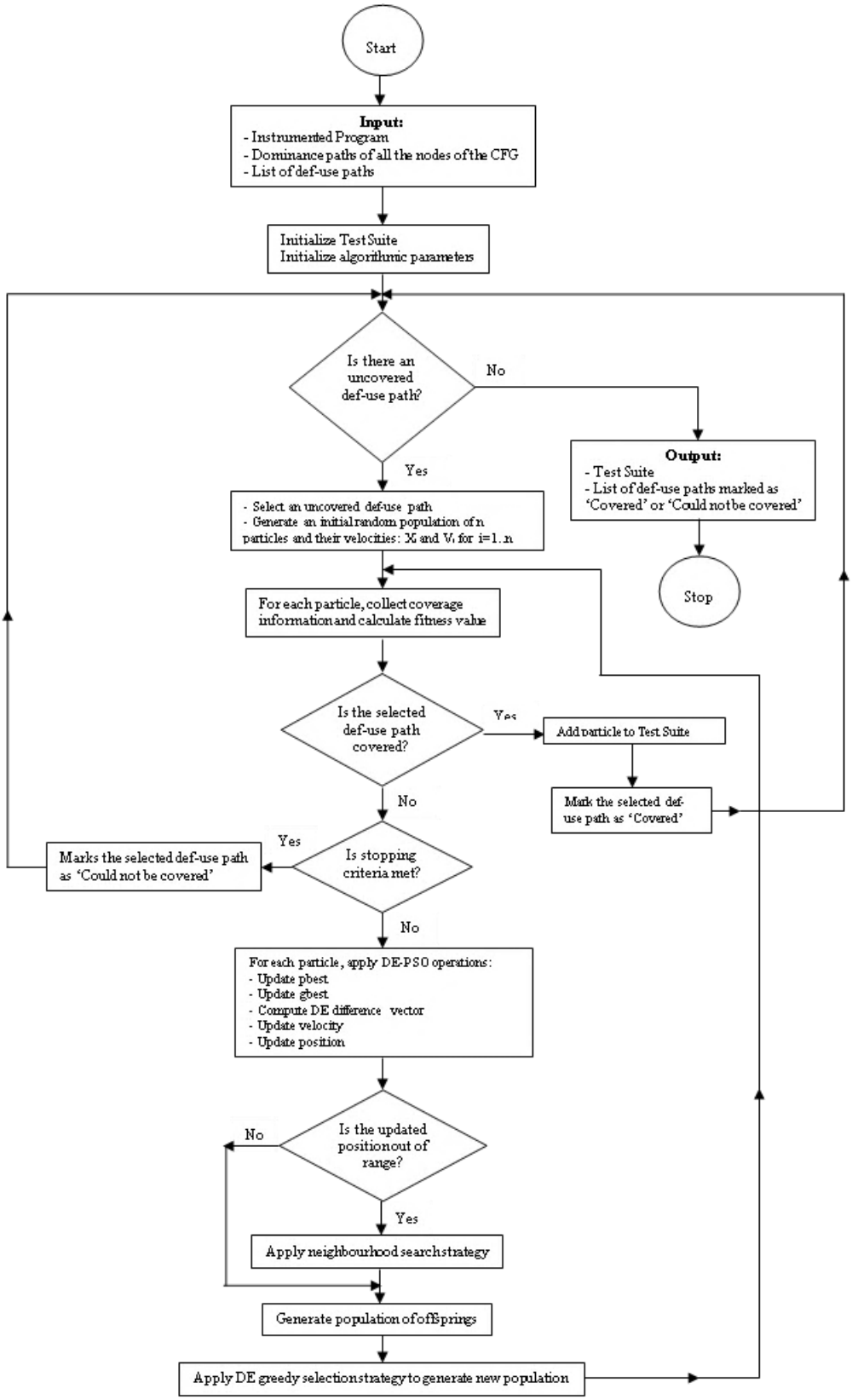


Figure 6: Flowchart of the proposed hybrid (adaptive PSO and DE) test data generation algorithm.

Table 5: Algorithmic parameter settings

Algorithm	Parameters	Value
Common Parameters	Population Size	10, 15, 20, 25
	Maximum number of generations	10^3
	Number of experiments for each program	100
	Fitness Function	As given by Eq. 11 and Eq. 12
	Threshold Euclidean distance	10
DE	Mutation Scaling Factor: F	1
	Crossover Constant: CR	0.9
PSO	Inertia weight	Adaptive as given by Eq. 3 and Eq. 4
	Acceleration constants: c1 and c2	c1=c2=2.0
	Maximum velocity: Vmax	Varies according to the program
GA	Chromosome encoding	Gray encoding
	Parent selection strategy	Roulette Wheel
	Probability of crossover	0.8
	Probability of mutation	0.15

considered are 10, 15, 20 and 25. 'Triangle Classifier' program is used as the pilot benchmark program and 100 experiments were carried out. Accordingly, in the main experiments, the following parameters settings have been used for adaptive PSO, DE and GA:

7.3 Subject programs

For this study, various benchmark programs have been selected from other researchers' work [6, 7, 13, 26] in the area of SBST. Experiments are also performed on programs taken from the SIR repository [45]. Source code of the academic programs is taken from standard reference books [38, 46, 47, 48]. The programs, as given in Table 6 below, have diverse structural elements such as loops, equality conditions, logically connected and nested predicates. A tool has also been developed for the instrumentation of programs and for listing of def-use paths.

7.4 Study results

This section presents the experimental results for various subject programs. For each subject program and each testing approach, 100 experiments were carried out. The measures collected are as follows:

- Mean number of generations: Sum of the number of generations at termination for each experiment over the total number of experiments gives the mean number of generations for a particular subject program.

  Here, termination criteria is either 100% data-flow coverage or 10^3 generations, whichever occurs first. Maximum number of generations is set to 10^3. For more complex programs, the maximum number of generations may be increased. Mean number of generations, however, is not indicative of full data-flow coverage.

Table 6: Subject programs

Program	#def-use Paths	Description	Type
1. Triangle Classifier	12	Finds the type of a triangle	Academic
2. Quadratic Equation	20	Finds the roots of a quadratic equation	Academic
3. Previous Date	66	Finds the previous date of a given date	Academic
4. Day of the Calendar	80	Finds the day on a given date	Academic
5. Marks Processing	19	Finds the final grade and average marks	Academic
6. Banking Transaction System	77	Banking transactions	Industrial
7. Sort	15	Sorting an array	Repository
8. Vector	26	Vector operations	Repository
9. Stack	20	Stack operations	Repository
10. Linked List	35	Linked list operations	Repository

- Mean percentage coverage: Sum of the data-flow coverage achieved for each experiment over the total number of experiments gives the mean percentage coverage achieved for a particular subject program. A def-use path is marked as covered the first time it is traversed and is not checked subsequently. The overall number of fitness evaluations is therefore reduced as stated in Section 2.

  If a path is infeasible, then some c-uses and p-uses that require this path to be traversed might also be infeasible [38]. For each program, infeasible uses, if any, were excluded while measuring data-flow coverage.

7.4.1 Effect of varying population size on the performance of the proposed hybrid (adaptive PSO and DE) algorithm

In this section, the effect of varying population size on the performance of the proposed hybrid algorithm with adaptive inertia weight and neighbourhood search strategy is analyzed. The performance is also compared with other meta-heuristic techniques and random search. The proposed hybrid algorithm, DE, PSO, GA (all guided by the same fitness function) and random search is applied to the various subject programs and experimental results are collected for the different measures. Population sizes that are considered are 10, 15, 20 and 25. Detailed experimental results are presented in Figures 7-16 below.

7.4.2 Overall comparison

In this section, overall performance of the proposed hybrid (adaptive PSO and DE) algorithm is compared with DE, PSO, GA and random search with respect to the measures collected. Tables 7 - 10, as given below, summarize the results of applying the various testing approaches to the set of chosen subject programs for

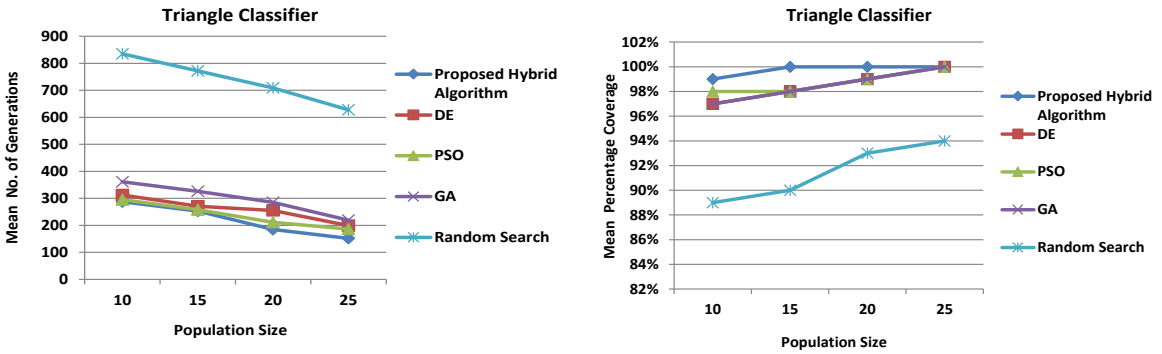


Figure 7: Graphs for 'Triangle Classifier' program.

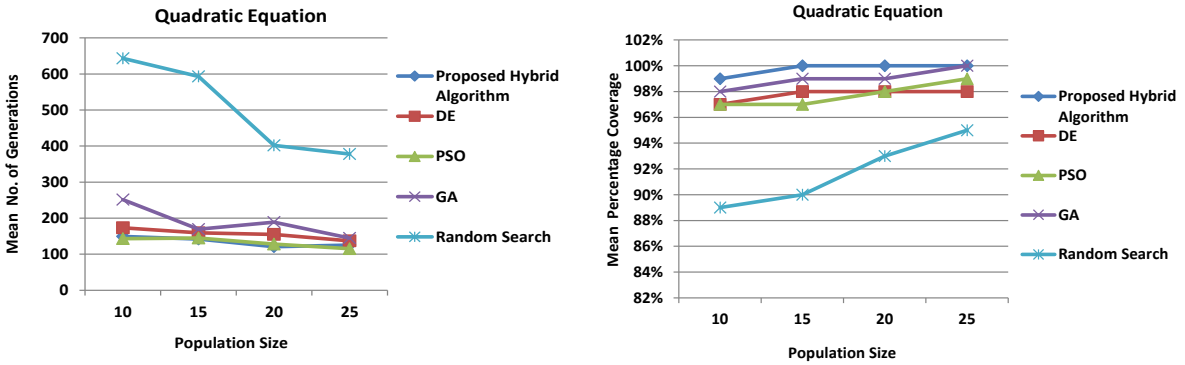


Figure 8: Graphs for 'Quadratic Equation' program.

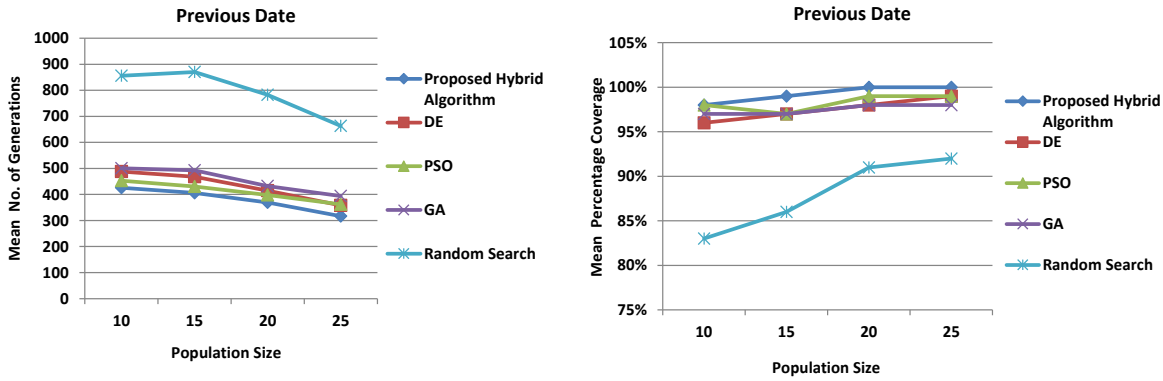


Figure 9: Graphs for 'Previous Date' program.

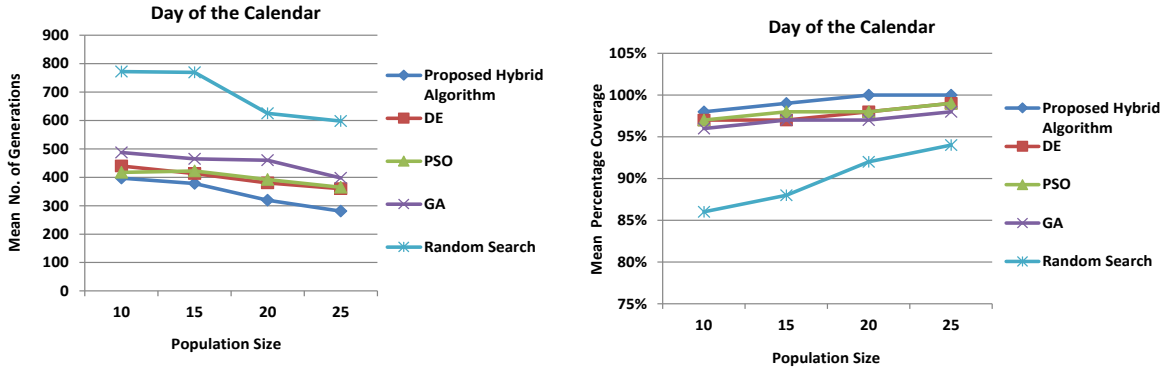


Figure 10: Graphs for 'Day of the Calendar' program.

different population sizes (10, 15, 20, 25). Range of the input integer variables is taken to be 0-100; range is different for variables of Program# 3, 4, and 7 as per the requirement of each program. The results are further discussed in the next section.

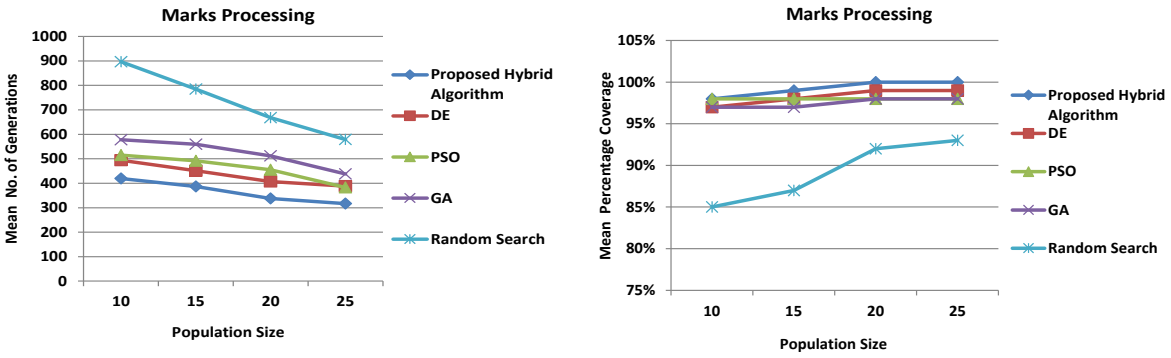


Figure 11: Graphs for 'Marks Processing' program.

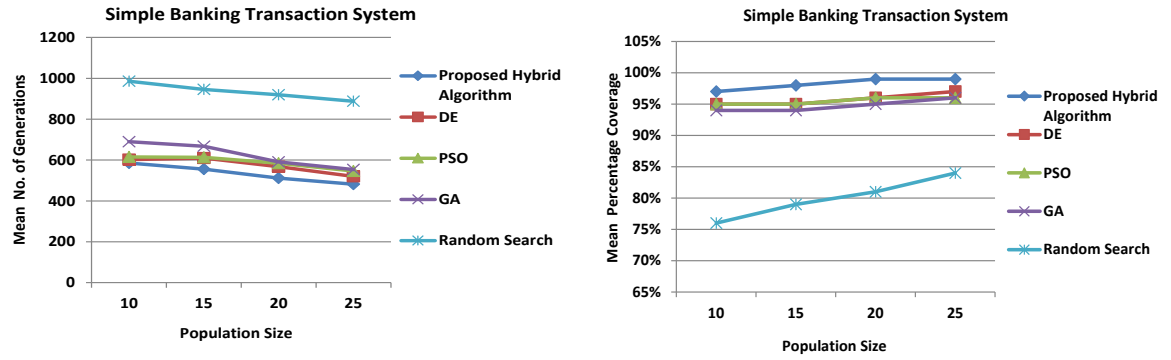


Figure 12: Graphs for 'Simple Banking Transaction System' program.

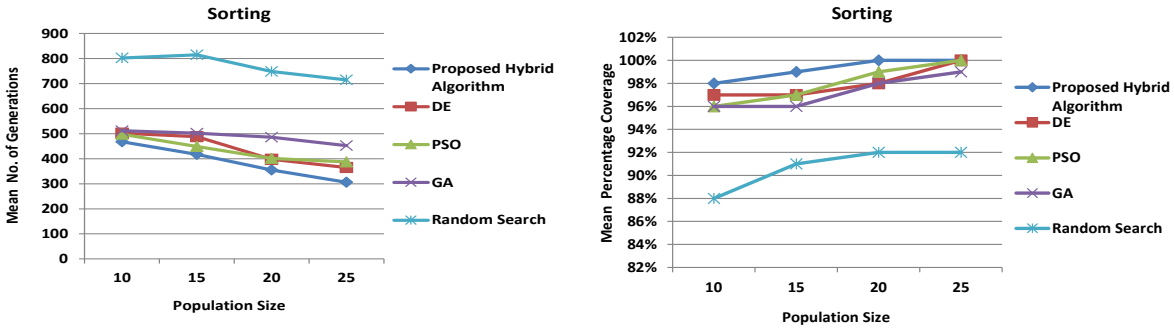


Figure 13: Graphs for 'Sort' program.

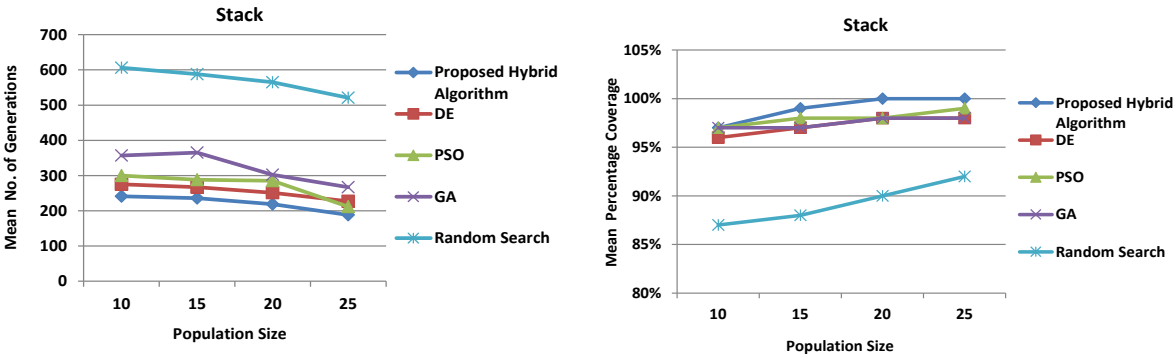


Figure 14: Graphs for 'Stack' program.

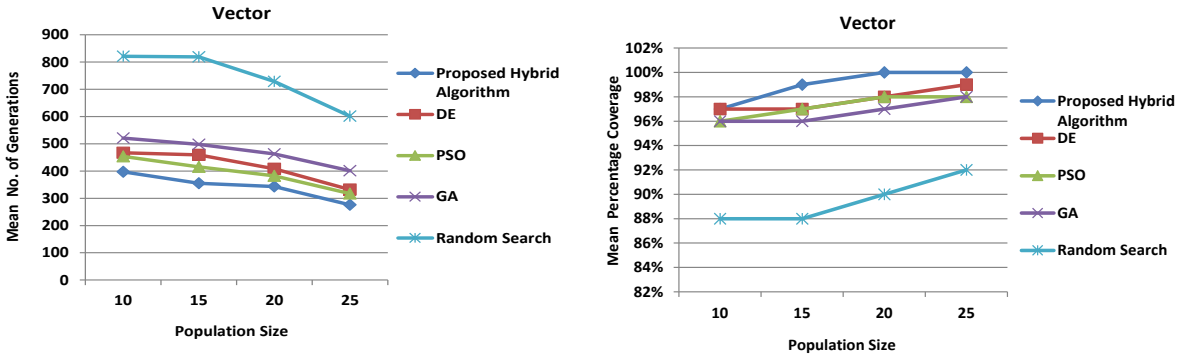


Figure 15: Graphs for 'Vector' program.

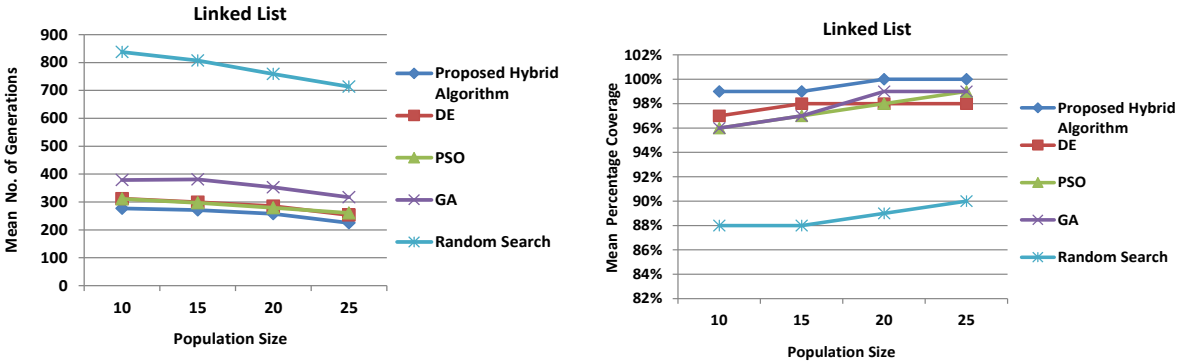


Figure 16: Graphs for 'Linked List' program.

Table 7: Experimental results for **Population Size 10**: Mean number of generations and mean percentage coverage.

Program	Measure									
	Mean Number of Generations					Mean Percentage Coverage				
	Proposed Hybrid Algorithm	DE	PSO	GA	Random Search	Proposed Hybrid Algorithm	DE	PSO	GA	Random Search
Triangle Classifier	287	312	295	361	835	99%	97%	98%	97%	89%
Quadratic Equation	289	320	316	353	743	99%	97%	98%	98%	90%
Previous Date	426	488	453	501	856	98%	97%	98%	97%	85%
Day of the Calendar	397	440	417	487	772	98%	97%	97%	96%	86%
Marks Processing	419	494	515	578	897	98%	97%	98%	97%	85%
Simple Banking Transaction System	585	602	615	690	986	97%	95%	95%	94%	76%
Sort	468	502	498	512	802	98%	97%	96%	96%	88%
Vector	397	467	454	521	821	97%	97%	96%	96%	88%
Stack	241	275	300	357	606	97%	96%	97%	97%	87%
Linked List	277	312	311	379	838	99%	97%	96%	96%	88%

Table 8: Experimental results for **Population Size 15**:
Mean number of generations and mean percentage coverage.

Program	Measure									
	Mean Number of Generations					Mean Percentage Coverage				
	Proposed Hybrid Algorithm	DE	PSO	GA	Random Search	Proposed Hybrid Algorithm	DE	PSO	GA	Random Search
Triangle Classifier	253	271	258	326	772	99%	97%	98%	97%	89%
Quadratic Equation	280	297	288	329	699	99%	98%	99%	99%	91%
Previous Date	406	468	431	493	870	99%	97%	98%	98%	87%
Day of the Calendar	378	413	422	465	769	99%	97%	98%	97%	88%
Marks Processing	387	451	492	559	785	99%	98%	98%	97%	87%
Simple Banking Transaction System	555	610	613	667	945	98%	95%	95%	94%	79%
Sort	417	488	449	502	815	99%	97%	97%	96%	91%
Vector	355	459	415	498	819	99%	97%	97%	96%	88%
Stack	236	267	288	365	588	99%	97%	98%	97%	88%
Linked List	271	299	297	381	807	99%	98%	97%	97%	88%

Table 9: Experimental results for **Population Size 20**:
Mean number of generations and mean percentage coverage.

Program	Measure									
	Mean Number of Generations					Mean Percentage Coverage				
	Proposed Hybrid Algorithm	DE	PSO	GA	Random Search	Proposed Hybrid Algorithm	DE	PSO	GA	Random Search
Triangle Classifier	185	255	211	285	709	100%	99%	99%	99%	93%
Quadratic Equation	246	255	241	289	502	99%	99%	99%	99%	93%
Previous Date	369	415	398	432	783	100%	98%	99%	98%	91%
Day of the Calendar	319	380	392	460	625	100%	98%	98%	97%	92%
Marks Processing	338	407	455	512	668	100%	99%	98%	98%	92%
Simple Banking Transaction System	512	568	584	591	919	99%	96%	96%	95%	81%
Sort	355	398	401	486	748	100%	98%	99%	98%	92%
Vector	343	408	382	463	729	100%	98%	98%	97%	90%
Stack	219	251	285	302	565	100%	98%	98%	98%	92%
Linked List	258	285	279	353	759	100%	98%	99%	99%	90%

Table 10: Experimental results for **Population Size 25**:
Mean number of generations and mean percentage coverage.

Program	Measure									
	Mean Number of Generations					Mean Percentage Coverage				
	Proposed Hybrid Algorithm	DE	PSO	GA	Random Search	Proposed Hybrid Algorithm	DE	PSO	GA	Random Search
Triangle Classifier	152	199	186	219	628	100%	100%	100%	100%	94%
Quadratic Equation	221	262	205	245	478	100%	99%	100%	100%	96%
Previous Date	317	358	362	394	663	100%	99%	99%	99%	93%
Day of the Calendar	281	360	365	398	598	100%	99%	99%	98%	94%
Marks Processing	317	388	383	438	579	100%	99%	98%	98%	93%
Simple Banking Transaction System	482	520	546	554	888	99%	97%	96%	96%	84%
Sort	306	365	387	452	715	100%	100%	100%	99%	92%
Vector	276	332	317	401	601	100%	99%	98%	98%	92%
Stack	188	227	212	267	521	100%	98%	99%	98%	92%
Linked List	225	253	261	317	714	100%	98%	99%	99%	90%

8 Discussion

The experimental results have been presented above in Tables 7-10 and Figures 7-16. In context of the research questions formulated for this study, the experimental results are analysed and discussed in this section.

RQ1: How effective is the proposed hybrid (adaptive PSO and DE) algorithm for optimal test data generation to achieve 100% data-flow coverage of a program?

From the experimental results as shown in Tables 7-10, it can be seen that the proposed hybrid algorithm with adaptive inertia weight and neighbourhood search strategy, achieved highest mean percentage coverage for all the subject programs and for all population sizes that are considered. Only the proposed hybrid algorithm achieved 100% data-flow coverage for all the subject programs for population size 20 (except for Program# 2 and Program# 6) and for population size 25 (except for Program# 6). For population size 10 and 15 also, the mean percentage coverage is 97%-99% with the proposed hybrid algorithm. For each program, infeasible uses, if any, were not considered while measuring data-flow coverage. Infeasible uses, if any, are determined by careful manual analysis as it is not possible to write an algorithm for analyzing a given program to determine if a given element in the coverage domain is feasible or not [38]. This, in addition to the novel fitness function, adaptive inertia weight and neighbourhood search strategy has resulted in full data-flow coverage as the population size is increased from 10 to 25.

For the other meta-heuristic search techniques (DE, PSO and GA), all guided by the same fitness function, mean percentage coverage is between 94%-99% for all the subject programs and for all population sizes that are considered. DE achieved 100% data-flow coverage only for Program# 1 and Program# 7 for population size 25. PSO achieved 100% data-flow coverage only for Program# 1, Program# 2, and Program# 7 for population size 25. GA achieved 100% data-flow coverage only for Program# 1 and Program# 2 for population size 25. However, the proposed hybrid algorithm outperformed DE, PSO and GA with respect to the convergence speed in all the cases. Performance of random search is worst; mean percentage coverage achieved is minimum for all the subject programs for all population sizes that are considered. This provides an explanation for high mean number of generations when percentage coverage is less than 100% as then the algorithm terminates only after 10^3 generations.

RQ2: How effective is the proposed hybrid (adaptive PSO and DE) algorithm for optimal test data generation with respect to the convergence speed (mean number of generations) at termination?

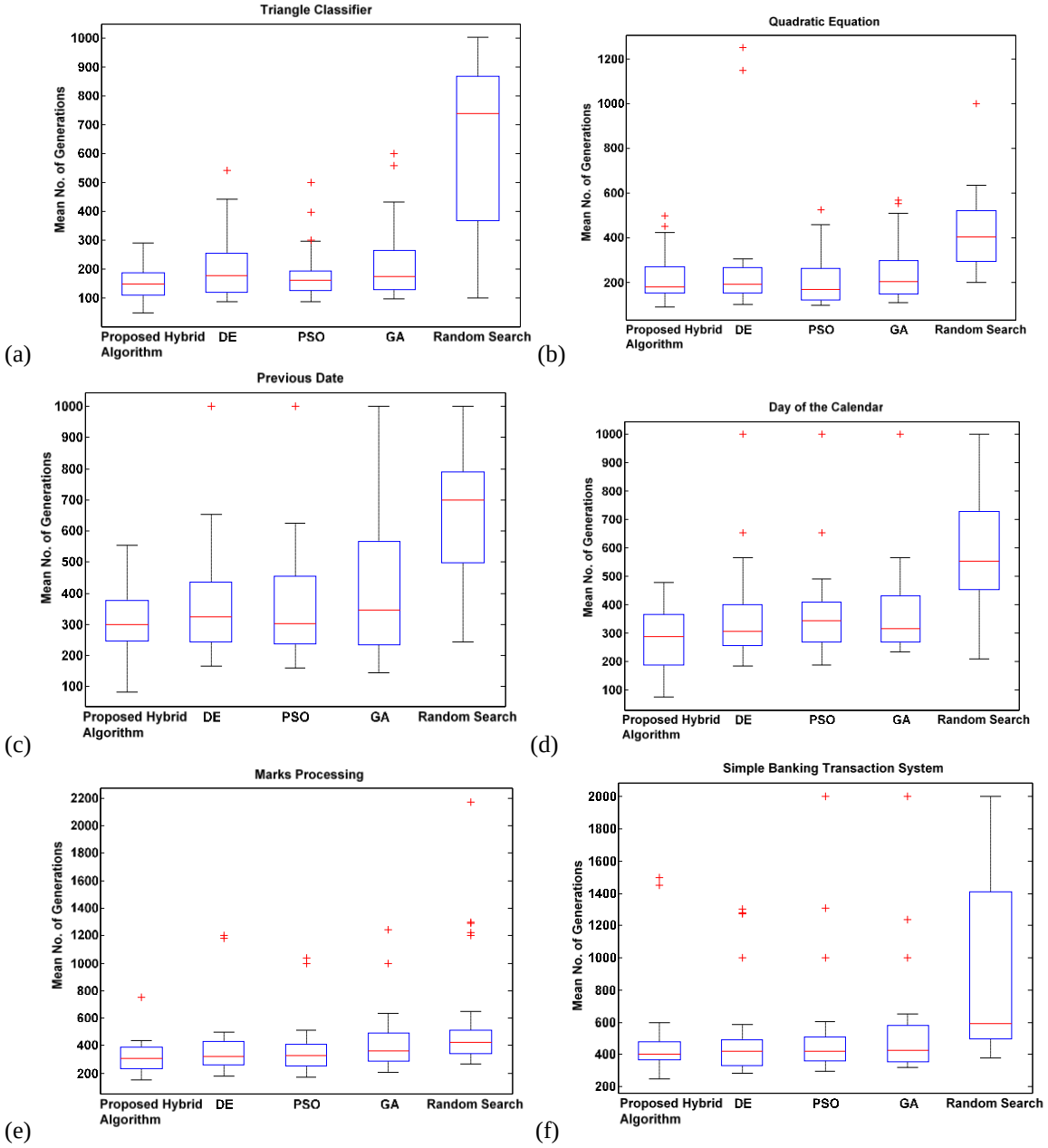
From the experimental results as shown in Tables 7-10, it can be seen that the mean number of generations is least with the proposed hybrid algorithm for all the subject programs and for all population sizes that are considered. There is a substantial reduction in mean number of generations with the proposed hybrid algorithm for benchmark programs such as ‘Triangle Classifier’, ‘Quadratic Equation’, and ‘Previous Date’

Table 11: Statistical results of Friedman Aligned and post hoc test (level of confidence α = 0.05)

Program	Testing Approach	Average Rank	Friedman Aligned Statistic	p-value by Friedman Aligned Test	p-value by applying Post Hoc Methods	Holm's Procedure α/i	Hypothesis
Triangle Classifier	**Proposed Hybrid Algorithm**	**34.5**			-	-	-
	DE	56.23			0.048691	0.05	**Rejected**
	PSO	63.43	24.2877	6.994E-05	0.0099	0.025	**Rejected**
	GA	89.23			0.000001	0.016667	**Rejected**
	Random Search	134.1			0	0.0125	**Rejected**
Quadratic Equation	**Proposed Hybrid Algorithm**	**52.03**			0.799444	0.05	**Not Rejected**
	DE	68.95			0.078049	0.025	**Not Rejected**
	PSO	49.18	24.2141	7.236E-05	-	-	-
	GA	80.03			0.005957	0.016667	**Rejected**
	Random Search	127.3			0	0.0125	**Rejected**
Previous Date	**Proposed Hybrid Algorithm**	**31.78**			-	-	-
	DE	52.57			0.063918	0.05	**Not Rejected**
	PSO	63.02	24.183394	7.339 E-05	0.005364	0.025	**Rejected**
	GA	97			0	0.016667	**Rejected**
	Random Search	133.13			0	0.0125	**Rejected**
Day of the Calendar	**Proposed Hybrid Algorithm**	**35.1**			-	-	-
	DE	58.15			0. 039897	0.05	**Rejected**
	PSO	66.42	24.151826	7.447E-05	0. 005242	0.025	**Rejected**
	GA	85.22			0. 000008	0.016667	**Rejected**
	Random Search	132.62			0	0.0125	**Rejected**
Simple Banking Transaction System	**Proposed Hybrid Algorithm**	**33.2**			-	-	-
	DE	55.08			0.0470	0.05	**Rejected**
	PSO	59.8	23.557598	9.795 E-05	0.017726	0.025	**Rejected**
	GA	96.32			0	0.016667	**Rejected**
	Random Search	133.1			0	0.0125	**Rejected**
Marks Processing	**Proposed Hybrid Algorithm**	**25.68**			-	-	-
	DE	49.28			0.035392	0.05	**Rejected**
	PSO	66.58	23.903359	8.352E-05	0.000266	0.025	**Rejected**
	GA	101.38			0	0.016667	**Rejected**
	Random Search	134.57			0	0.0125	**Rejected**
Sort	**Proposed Hybrid Algorithm**	**34.61**			-	-	-
	DE	57.88			0.038067	0.05	**Rejected**
	PSO	65.55	24.028397	7.883E-05	0.005823	0.025	**Rejected**
	GA	90.38			0.000001	0.016667	**Rejected**
	Random Search	129.07			0	0.0125	**Rejected**
Vector	**Proposed Hybrid Algorithm**	**34.7**			-	-	-
	DE	61.73			0.015956	0.025	**Rejected**
	PSO	56.83	24.235764	7.164E-05	0.048484	0.05	**Rejected**
	GA	99.33			0	0.016667	**Rejected**
	Random Search	124.9			0	0.0125	**Rejected**
Stack	**Proposed Hybrid Algorithm**	**32.52**			-	-	-
	DE	53.25			0.06456	0.05	**Not Rejected**
	PSO	70.13	23.829629	8.641E-05	0.000798	0.025	**Rejected**
	GA	92.77			0	0.016667	**Rejected**
	Random Search	128.83			0	0.0125	**Rejected**
Linked List	**Proposed Hybrid Algorithm**	**34.78**			-	-	-
	DE	60.65			0.021116	0.025	**Rejected**
	PSO	59.48	23.551908	9.821E-05	0.027672	0.05	**Rejected**
	GA	91.53			0	0.016667	**Rejected**
	Random Search	131.05			0	0.0125	**Rejected**

that have multiple and nested conditions along with equality conditions. This is also true for other programs taken from the repository [45] such as 'Sort', 'Stack', 'Vector', and 'Linked List'. As expected, the mean number of generations decreases as the population size increases due to a wider search space.

The performance of random search is worst with respect to the mean number of generations to achieve same data-flow coverage for smaller population sizes and



for programs with multiple and nested conditions. Random search did not achieve full data-flow coverage for any of the subject program. This has resulted in higher values for the measure 'mean number of generations' at termination.

It can be inferred that the proposed hybrid algorithm with adaptive inertia weight and neighbourhood search strategy is the best performing approach for all the subject programs and for all population sizes that are considered with respect to the measures collected. The proposed hybrid algorithm and the other meta-heuristic search techniques (DE, PSO and GA) are all guided by the same novel fitness function; the better performance of the proposed hybrid algorithm can be attributed to the inclusion of adaptive inertia weight and neighbourhood search strategy.

8.1 Statistical analysis on repeated trials

Statistical analysis is performed to validate the effectiveness and efficiency of the proposed hybrid (adaptive PSO and DE) algorithm with adaptive inertia weight and neighbourhood search strategy over other meta-heuristic search techniques (DE, PSO and GA) and random search applied for test data generation in accordance to data-flow coverage criterion. The experiment on each subject program was repeated 100 times. From the experimental results as presented in Section 7.4, it can be seen that the proposed hybrid algorithm as well as the other meta-heuristic search techniques (DE, PSO and GA), all guided by the same fitness function, have comparable results with respect to the measure 'mean percentage coverage' for population size 10 and 15. The proposed hybrid algorithm achieved 100% data-flow coverage for all the subject programs for

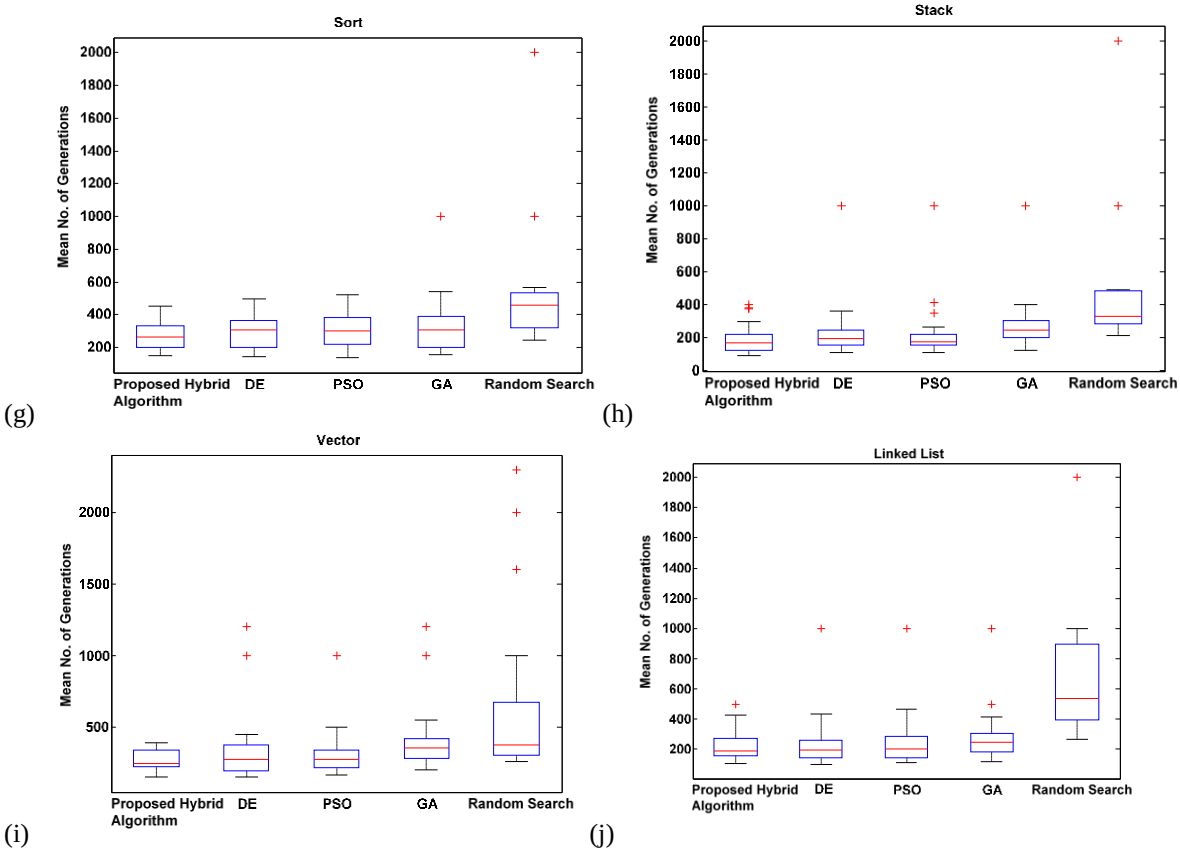


Figure 17: Stability analysis for the measure 'mean number of generations'.

population size 20 (except for Program# 2 and 6) and for population size 25 (except for Program# 6). For population size 25, DE achieved 100% data-flow coverage only for Program# 1 and 7; PSO achieved 100% data-flow coverage only for Program# 1, 2, and 7; GA achieved 100% data-flow coverage only for Program# 1 and 2. Therefore, the convergence speed (mean number of generations) information for population size 25 (best performance for all the approaches) is used for statistical difference test.

In the first step, Friedman Aligned 1xN test, a non-parametric multiple comparison statistical test [49], is applied to check for significant differences between the performance of the proposed hybrid algorithm and the other algorithms. Average rankings of all the algorithms are obtained that provide a fair comparison of the algorithms; a low value indicates higher rank. The unadjusted p-value is also computed through normal approximations; the smaller the p-value, the stronger the evidence against the null hypothesis. The value of α (level of confidence) is set to 0.05. In the second step, if the null hypothesis of equivalence of rankings is rejected, a post hoc test (Holm's procedure) is applied to report adjusted p-values by adjusting the value of α in a step-down manner to compensate for multiple comparisons. Here, the proposed hybrid algorithm acts as the control algorithm and its performance is compared with the rest of the algorithms used for comparison.

Results of the statistical analysis are summarized in Table 11 - average ranking of each algorithm, Friedman Aligned statistic, p-value computed by Friedman Aligned test and p-values obtained in by applying post hoc methods. It can be observed that the rank of the proposed hybrid algorithm is minimum (best performing algorithm) for all the subject programs except for 'Quadratic Equation' program. In case of 'Quadratic Equation' program, PSO is the best performing algorithm; however, PSO did not achieve full data-flow coverage and the proposed hybrid algorithm achieved full data-flow coverage as can be seen in Table 10. Random search gets the worst rank among all the algorithms as expected. The p-values computed by Friedman Aligned test are ≤ α (level of confidence) for all the subject programs, so the null hypothesis of equivalence of rankings can be rejected.

Further, p-values at the level of confidence α are reported by applying Holm's procedure to compensate for multiple comparisons. Holm's procedure rejects those hypotheses that have an unadjusted p-value ≤ α. As can be seen, all the null hypotheses are rejected in all the cases for all the subject programs except for 'Quadratic Equation', 'Previous Date' and 'Stack' programs. The null hypothesis is not rejected for DE in case of 'Quadratic Equation' (for proposed hybrid algorithm also), 'Previous Date' and 'Stack' programs. However, as can be seen from Tables 7 - 10, there is significant difference among the performance of all the algorithms being compared with respect to the measures collected. Thus, it be claimed that there is significant difference between the performances of the proposed hybrid algorithm and the other algorithms being compared.

For further analysis, box plots are drawn as shown in Figure 17 to compare the distribution of the measure *mean number of generations* over 100 trials for all the subject programs (population size 25). It can be observed that the median value of the measure 'mean number of generations' (in 100 trials) for the proposed hybrid algorithm is always less than the corresponding values for DE, PSO, GA and random search for all the subject programs except for 'Quadratic Equation' program. The median value is comparable with that of PSO for the 'Quadratic Equation' program. For all the approaches, the difference between the first quartiles as well as the difference between the third quartiles is quite visible.

It can therefore be concluded that the proposed hybrid (adaptive PSO and DE) algorithm is the best performing algorithm and is significantly different from the other algorithms (DE, PSO, GA and random search) being compared. The proposed hybrid (adaptive PSO and DE) algorithm has stronger ability to generate test data with higher data-flow coverage as well as convergence speed as compared to DE, PSO, GA and random search techniques.

9 Threats to validity and limitations

This section presents the possible validity threats [50] for the proposed study. Threats to internal validity are considered in the context of SBST. The choice of algorithmic parameters such as population size, inertia weight, acceleration constants, maximum velocity, mutation scaling factor, crossover constant affects the performance of the meta-heuristic search algorithms. Preliminary experiments were carried out to determine the appropriate values for the various algorithmic parameters for the proposed hybrid (adaptive PSO and DE) algorithm.

Threats to construct validity may arise from the fact that the performance of the proposed hybrid (adaptive PSO and DE) algorithm is evaluated with respect to the measures 'mean number of generations' and 'mean percentage coverage' for a particular subject program. Other measures such as total number of fitness evaluations and average search time may have also been used for evaluation.

Statistical analysis is performed to establish conclusion validity i.e. to validate the effectiveness and efficiency of the proposed hybrid (adaptive PSO and DE) algorithm over other techniques that have been considered for comparison. It is shown that the proposed hybrid (adaptive PSO and DE) algorithm is significantly different to DE, PSO, GA and random search that are considered for comparison; all except random search have been guided by the same fitness function. Adaptive inertia weight and neighbourhood search strategy have improved the performance of the proposed hybrid (adaptive PSO and DE) algorithm with respect to the measures collected. Threats to conclusion validity may arise from the fact that the infeasible uses / infeasible data-flow paths are identified and eliminated by manual analysis. Also, results for the proposed hybrid algorithm and other techniques have been compiled with respect to the experimental setup used for the present study.

The main external threat to validity is the choice of subject programs that may limit the generalization of results of the proposed study to real and more complex programs. Also, a different population size apart from those considered may produce different coverage results.

However, subject programs that are considered have many of the same programming constructs as large programs. The proposed approach should therefore be able to handle real and more complex programs. The claim is, however, a matter of further investigation.

10 Conclusion

Automated test data generation is still an open problem in spite of decades of research. In the field of SBST, GA has been the algorithm of choice for control-flow coverage criteria. Very recently only, other highly adaptive search-based techniques such as PSO have been employed for structural test data generation. DE is another simple to implement and highly adaptive search-based technique that has been not yet applied for automated test data generation. Among the structural test adequacy criteria, data-flow coverage test adequacy criterion has received relatively little attention. This paper presents a hybrid (adaptive PSO and DE) algorithm with neighbourhood search strategy for optimal test data generation in accordance to the all-uses data-flow coverage test adequacy criterion.

The performance of the proposed hybrid (adaptive PSO and DE) algorithm has been experimentally evaluated and compared with that of DE, PSO, GA and random search for data-flow coverage. It is shown that the proposed hybrid (adaptive PSO and DE) algorithm outperformed DE, PSO, GA and random search with respect to the measure 'mean number of generations' for all the population sizes that are considered. For the measure 'mean percentage coverage', performance of the proposed hybrid (adaptive PSO and DE) algorithm is comparable to that of DE, PSO and GA for smaller population sizes (10 and 15); however, only the proposed hybrid algorithm achieved full data-flow coverage as the population size is increased to 20 and 25 for complex subject programs. Performance of random search is worst. Here, we have explored a promising hybrid optimization algorithm for test data generation. In future, we intend to fine tune the algorithmic parameters and work upon more complex subject programs.

11 References

[1] H. Zhu, P.A.V. Hall, J.H.R. May (1997). Software unit test coverage and adequacy. *Computing Surveys*, ACM, Vol. 29, No. 4, pp. 366-427. https://doi.org/10.1145/267580.267590

[2] R.A. DeMillo and A.J. Offutt (1991). Constraint-based automatic test data generation. *IEEE Transactions on Software Engineering*, Vol. 17, No. 9, pp. 900-910. https://doi.org/10.1109/32.92910

[3] N. Tracey (2010). A Search-Based Automated Test-Data Generation Framework for Safety-Critical Software. *Doctoral Thesis*, University of York.

[4] X.S. Yang (2010). *Engineering Optimization: An Introduction with Metaheuristic Applications*. Wiley, New Jersey.

[5] M.A. Ahmed and I. Hermadi (2007). GA-based multiple paths test data generator. *Computers and Operations Research*, Elsevier, Vol. 35, No. 10, pp. 3107-3124. https://doi.org/10.1016/j.cor.2007.01.012

[6] A.S. Ghiduk, M.J. Harrold, and M.R. Girgis (2007). Using Genetic Algorithms to Aid Test-Data Generation for Data-Flow Coverage. *Proceedings of the 14th Asia-Pacific Software Engineering Conference*, IEEE, pp. 41-48. https://doi.org/10.1109/ASPEC.2007.73

[7] M.R Girgis (2005). Test Data Generation for Data Flow Testing Using a Genetic Algorithm. *Journal of Universal Computer Science*, Vol. 11, No. 6, pp. 898-915. http://dx.doi.org/10.3217/jucs-011-06-0898

[8] P. McMinn (2004). Search-Based Software Test Data Generation: A Survey. *Journal of Software Testing, Verification and Reliability*, Wiley, Vol. 14, No. 2, pp. 105-156. https://doi.org/10.1002/stvr.294

[9] R.P. Pargas, M.J. Harrold, and R. Peck (1999). Test-Data Generation Using Genetic Algorithms. *Journal of Software Testing, Verification and Reliability*, Wiley, Vol. 9, No. 4, pp. 263-282. https://doi.org/10.1002/(SICI)1099-1689(199912)9:4%3c263::AID-STVR190%3e3.0.CO;2-Y

[10] A. Windisch, S. Wappler, and J. Wegener (2007). Applying Particle Swarm Optimization to Software Testing. *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO'07)*, pp. 1121-1128. https://doi.org/10.1145/1276958.1277178

[11] K. Agarwal, A. Pachauri, and Gursaran (2008). Towards Software Test Data Generation using Binary Particle Swarm Optimization. *Proceedings of the XXXII National Systems Conference*, pp. 339-343.

[12] K. Agarwal and G. Srivastava (2010). Towards Software Test Data Generation using Discrete Quantum Particle Swarm Optimization. *Proceedings of the ISEC'10*, pp. 65-68. https://doi.org/10.1145/1730874.1730888

[13] C. Mao (2014). Generating Test Data for Software Structural Testing Based on Particle Swarm Optimization. *Arabian Journal of Science and Engineering*, Springer, Vol. 39, No. 10, pp. 4593-4607. https:/doi.org/10.1007/s13369-014-1074-y

[14] N. Nayak and D.P. Mohapatra (2010). Automatic Test Data Generation for Data Flow Testing Using Particle Swarm Optimization. *Springer-Verlag Heidelberg*, pp. 1-12.

[15] A. Arcuri and X. Yao (2008). Search based software testing of object-oriented containers. *Information Sciences*, Elsevier, Vol. 178, No. 15, pp. 3075-3095. https://doi.org/10.1016/j.ins.2007.11.024

[16] S. Zhang, Y. Zhang, H. Zhou, and Q. He (2010). Automatic path test data generation based on GA-PSO. *Proceedings of the International Conference on Intelligent Computing and Intelligent Systems (ICIS'10)*, IEEE, pp. 142-146. https://doi.org/10.1109/ICICISYS.2010.5658735

[17] K. Li, Z. Zhang, and J. Kou (2010). Breeding Software Test Data with Genetic-Particle Swarm Mixed Algorithm. *Journal of Computers*, Vol. 5, No. 2, pp. 258-265. https://doi.org/10.4304/jcp.5.2.258-265

[18] S. Singla, D. Kumar, H.M. Rai, and P. Singla (2011). A hybrid PSO approach to automate test data generation for data flow coverage with dominance concepts. *International Journal of Advanced Science and Technology*, Vol. 37, pp. 15-26.

[19] A. Kaur and D. Bhatt (2011). Hybrid particle swarm optimization for regression testing. *International Journal on Computer Science and Engineering*, Vol. 3, No. 5, pp. 1815-1824.

[20] M.R. Girgis, A.S. Ghiduk, and E.H. Abd-Elkawy (2015). Automatic Data Flow Test Paths Generation using the Genetical Swarm Optimization Technique. *International Journal of Computer Applications*, Vol. 116, No. 22, pp. 25-33.

[21] P. Chawla, I. Chana, and A. Rana (2015). A novel strategy for automatic test data generation using soft computing technique. *Frontier Computer Science*, Springer, Vol. 9, No. 3, pp. 346-363. https://doi.org/10.1007/s11704-014-3496-9

[22] S. Varshney and M. Mehrotra (2016). Search-based Test Data Generator for Data-Flow Dependencies using Dominance Concepts, Branch Distance and Elitism. *Arabian Journal of Science and Engineering*, Springer, Vol. 41, No. 3, pp. 853-881. https://doi.org/10.1007/s13369-015-1921-5

[23] B. Korel (1990). Automated Software Test Data Generation. *IEEE Transactions on Software Engineering*, Vol. 16, No. 8, pp. 870-879. https://doi.org/10.1109/32.57624

[24] P. G. Frankl and S. N. Weiss (1993). An Experimental Comparison of the Effectiveness of Branch Testing and Data Flow Testing. *IEEE Transactions on Software Engineering*, Vol. 19, No. 8, pp. 774-787. http://doi.ieeecomputersociety.org/10.1109/32.238581

[25] S. Ali, L.C. Briand, H. Hemmati, and R.K.P. Walawege (2010). A Systematic Review of the Application and Empirical Investigation of Search-Based Test Case Generation. *IEEE Transactions on Software Engineering*, Vol. 36, No. 6, pp. 742-762. https://doi.org/10.1109/TSE.2009.52

[26] A. Pachauri and G. Srivastava (2013). Automated test data generation for branch testing using genetic algorithm: An improved approach using branch ordering, memory and elitism. *Journal of Systems*

and Software, Elsevier, Vol. 86, No. 5, pp. 1191-1208. https://doi.org/10.1016/j.jss.2012.11.045

[27] S. Varshney and M. Mehrotra (2013). Search Based Software Test Data Generation for Structural Testing: A Perspective. *ACM SIGSOFT Software Engineering Notes*, Vol. 38, No. 4. https://doi.org/10.1145/2492248.2492277

[28] J. Wegener, A. Baresel, and H. Sthamer (2001). Evolutionary test environment for automatic structural testing. *Information and Software Technology*, Elsevier, Vol. 43, No. 14, pp. 841-854. https://doi.org/10.1016/S0950-5849(01)00190-2

[29] K. Liaskos, M. Roper, and M. Wood (2007). Investigating Data-Flow Coverage of Classes Using Evolutionary Algorithms. *Proceedings of the 9th annual conference on Genetic and Evolutionary Computation (GECCO'07)*, pp. 33-53. https://doi.org/10.1145/1276958.1277183

[30] M. Vivanti, A. Mis, A. Gorla, and G. Fraser (2013). Search-based Data-Flow Test Generation. *International Symposium on Software Reliability Engineering (ISSRE)*, IEEE, pp. 370-379. https://doi.org/10.1109/ISSRE.2013.6698890

[31] A. Ghiduk (2010). A New Software Data-Flow Testing Approach via Ant Colony Algorithms. *Universal Journal of Computer Science and Engineering Technology*, Vol. 1, No. 1, pp. 64-72.

[32] C. Mao, L. Xiao, X. Yu, and J. Chen (2015). Adapting ant colony optimization to generate test data for software structural testing. *Swarm and Evolutionary Computation*, Elsevier, Vol. 20, pp. 23-36. http://dx.doi.org/10.1016/j.swevo.2014.10.003

[33] C. Mao (2014). Harmony search-based test data generation for branch coverage in software structural testing. *Neural Computing and Applications*, Springer, Vol. 25, No. 1, pp. 199-216. https://doi.org/10.1007/s00521-013-1474-z

[34] A. Shamekhi (2013). An Improved Differential Evolution Optimization Algorithm. *International Journal of Research and Reviews in Applied Sciences*, Vol. 15, No. 2, pp. 132-145.

[35] H. Sharma, P. Shrivastava, and J.C. Bansal (2014). Fitness Based Self Adaptive Differential Evolution. *Nature Inspired Cooperative Strategies for Optimization (NICSO 2013)*, pp. 71-84.

[36] S. Das, A. Abraham, and A. Konar (2008). Particle Swarm Optimization and Differential Evolution Algorithms: Technical Analysis, Applications and Hybridization Perspectives. *Studies in Computational Intelligence*, Springer, Vol. 116, pp. 1-38. https://doi.org/10.1007/978-3-540-78297-1_1

[37] R. Malhotra and M. Khari (2013). Heuristic search-based approach for automated test data generation: a survey. *International Journal of Bio-Inspired Computation*, Inderscience, Vol. 5, No. 1. https://doi.org/10.1504/IJBIC.2013.053045

[38] A.P. Mathur (2008). *Foundations of Software Testing*. Pearson.

[39] T. Lengauer and R.E. Tarjan (1979). A fast algorithm for finding dominators in a flowgraph. *ACM Transactions on Programming Languages and Systems*, Vol. 1, No. 1, pp. 121-141. https://doi.org/10.1145/357062.357071

[40] S. Rapps and E.J. Weyuker (1985). Selecting software test data using data flow information. *IEEE Transactions on Software Engineering*, Vol. 11, No. 4, pp. 367-375. https://doi.org/10.1109/TSE.1985.232226

[41] J. Kennedy and R. C. Eberhart (1995). Particle Swam Optimization. *Proceedings of IEEE International Conference on Neural Networks (ICNN'95)*, IEEE, pp. 1942-1948. https://doi.org/10.1109/ICNN.1995.488968

[42] A. Nickabadi, M.M. Ebadzadeh, and R. Safabakhsh (2011). A novel particle swarm optimization algorithm with adaptive inertia weight. *Applied Soft Computing*, Elsevier, Vol. 11, No. 4, pp. 3658-3670. https://doi.org/10.1016/j.asoc.2011.01.037

[43] R. Storn and K.V. Price (1995). Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces. *International Computer Science Institute, USA*, TR-95-012.

[44] R. Ferguson and B. Korel (1996). The chaining approach for software test data generation. *ACM Transactions on Software Engineering and Methodology*, Vol. 5, No. 1, pp. 63-86. https://doi.org/10.1145/226155.226158

[45] SIR: Software-artifact Infrastructure Repository. http://sir.unl.edu/portal/index.php

[46] P.C. Jorgenson (2002). *Software Testing: A Craftsman's Approach*. 2nd ed., CRC Press.

[47] G.J. Myers (2006). The *Art of Software Testing*. 2nd ed., Wiley.

[48] K.K Aggarwal and Y. Singh (2007). *Software Engineering*. 3rd ed., New Age International Publishers.

[49] Garcia, A. Fernandez, J. Luengo, and F. Herrera (2010). Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences*, Elsevier, Vol. 180, pp. 2044-2064. https://doi.org/10.1016/j.ins.2009.12.010

[50] R. Feldt and A. Magazinius (2010). Validity threats in empirical software engineering research-an initial survey. *Proceedings of the 22nd International Conference on Software Engineering and Knowledge*