A Stochastic Method for Test Case Selection in Software Testing

Nithya T.M, Chitra. S

Abstract*: The quality of the software is a very important aspect in the development of software application. In order to make sure there is the software of good quality, testing is a critical activity of software development. Thus, software testing is the activity which focuses on the computation of an attribute or the ability of either a system or program that decides if user requirements are met. There is a proper strategy for the design of software for which testing has to be adopted. The techniques of test case selection attempt at reduction of the test cases that need to be executed at the same time satisfying the needs of testing that has been denoted by the test criteria. In the time of software testing, and the resource will be the primary constraints at the time of testing since this has been a highly neglected phase in the Software Development Life Cycle (SDLC). The optimizing of a test suite is very critical for the reduction of the testing phase and also the selection of the test cases that eliminate unwanted or redundant data. All work in literature will make use of techniques of single objective optimization that does not have to be efficient as the code coverage will play an important role at the time of selection of test case. As the test case choice is Non-Deterministic, the work also proposes a novel and multi-objective algorithm like the Non-Dominated Sorting Genetic Algorithm II (NSGA II) and the Stochastic Diffusion Search (SDS) algorithm that makes use of the cost of execution and code coverage as its objective function. The results prove a faster level of convergence of the algorithm with better coverage of code in comparison to the NSGA II.

***Index Terms*: Multi- Objective Optimization, Non-dominated Sorting Genetic Algorithm II (NSGA II) Stochastic Diffusion Search (SDS), Software Testing, Test Case Selection.**

I. INTRODUCTION

Both the verification along with the validation of software made by means of dynamic testing which is part of the area of such software engineering in which the progress is towards the automation which was slow. More particularly, for automatic design or also the generation of test data, which is generally a manual activity. Even today, there is software testing which continues to be the primary technique that is used for the purpose of gaining the confidence of customers in this software. This testing process of a software system is identified to be a major task and this is also very time-consuming. Software testing is laborious and about 50% of the development of software system resources [1].

Normally, the primary goal of software testing will be the designing of a new set of such test cases in a way in which it can depict the maximum faults. There are, however, some more benefits which are: a test preparation which is done well in advance and this will have some test runs that are very fast with the confidence of the testing result that may be increased. But, there is also an automation of software testing is not a process and is straightforward. For many years, there were several researchers that have proposed many methods for generation of test data in an automatic manner with different methods to develop the generators of test data. The technique further supports the automation of software testing that may result in a significant level of savings of cost.

A test that is effective gets dependent on a certain specific number of the detailed conditions that were employed in the process of software testing. There were some more test cases that were the parameters of an input along with some more conditions of execution and also their expected results that were used for testing. But, there was a set of another set of these test cases (called the test suite) that is available to the testers that grow in size with the evolving of the needs of the software. For the purpose of this scenario, there had been another execution of the test suite that may be unfeasible. This selection of the test case indicates all approaches that have been aimed at the choice of a new subset of test cases. The approaches were the key to the definition of all testing strategies or their development since they aim at the development since they aim at eliminating unwanted or redundant data and maximizing fault detection [2].

The prioritization of the test cases is the choice of test cases in the order of priority along with execution with the components that will specify the input, the operation and the outcome expected to determine if the properties of the application is working right. The methods of prioritization: the initial ordering, the random ordering, and the reverse ordering were based on the ability of fault detection. In the development of software application, there is a test suite that is less commonly called the suite to check the software validity. This test suite consists of a detailed set of goals or instructions for every test case collection or information on the configuration of the system that is used at the time of testing [3].

The process of these test automation aspects aims at replacing all manual and subjective tests that contribute to the boosting of productivity alongside the optimization of both resources and cost. The primary issue found in this scenario was to create and further improve their test suite as the number of traits can have a major impact on the process of software development. Owing to fact of the complexity and also the diversity that is inherent in test optimization presented by literature and divided into various topics which are distinct and related like the test case generation, test suite minimization and also the choice and prioritization of the test cases [4].

Revised Manuscript Received on December 5, 2019.

*__Mrs. Nithya T.M,__ Assistant Professor, Department of Computer Science and Engineering, K. Ramakrishnan college of Engineering, Trichy, Tamil Nadu, India

Dr. Chitra. S, Principal, Er. Perumal Manimekalai College of Engineering, Hosur, Tamil Nadu, India.

The generators of test cases are the programs receiving the parameters of input, the data structure definitions, test criteria, specifications and so on. The algorithms of test case generation make use of heuristics and some more strategies that aim at the test cases to maximize test coverage. But, generating the realistic test cases meet the needs of tests and with the complexity and the diversity of the parameters of input are important for the real world applications.

The coverage-based test case selection: this is examined widely in code coverage. This coverage found in the test case is measured by the total number of lines that have been executed using the test case. Even though there has been a finer level of coverage can result in potentially more effective information that is unavailable unless there is an execution of the code [5].

The Diversity-based selection of the test case: in recent years, there has been a lot of diversity among test cases which is a critical function in optimization. Speaking intuitively, this diversity among two of the test cases has been a distance function which can ideally measure dissimilarity. Diversity for either one or more than one of the test cases denotes an average and a pairwise diversity for this set. By assuming that every test case was encoded as the binary vector wherein the 1s correspond to program units as the functions within the test case. For this encoding, all functions of diversity in literature are the Hamming distance, the Dice diversity or the Levenshtein.

The problems of optimization which get impacted by several factors are known as multi-objective. It may not be possible at all times to be able to find a single solution to optimize all the objectives at the same time. This is owing to the fact that the objective functions that are connected to the diverse metrics generally have some conflict with a set of ideal solutions that were generated normally following the concepts of Pareto dominance [6]. This Pareto efficient approach can take many objectives like the code coverage, the history of past fault-detection and the cost of execution.
In order to be able to overcome the problem of multi-objective optimization, the NSGA with the SDS algorithm was proposed. The rest of the investigation has been organized thus. The discussion of all related work in literature is made in section 2. The different methods used were explained in section 3. The experimental results were discussed in section 4 and the conclusion was made in Section 5.

II. RELATED WORK

Shin and Lim [8] had made yet another proposal of a method which will automatically generate software along with hardware test cases from the UML model developed using a process of model-based development. In this, languages like the source code were used inside the mode. These expected values that are used for the test case had been generated with a custom parser. These subsequent steps are found in the test case was combined for generating an integration test with a bottom-up approach. After this, all the cases will be converted into their hardware test cases that were used for the approval testing of embedded systems by means of using the XQuery along with tables of hardware mapping. This approach had been able to provide a procedure for automatic testing in the embedded systems that had been developed by the methods that were model-based and also generated the test cases very efficiently. For concluding, the method could help in the reduction of resources needed for test case generation made from the software to the hardware.

Ali et al., [9] had proposed some results of a comprehensive and systematic review aiming at characterizing the manner in which empirical investigations were designed for investigating the Search-Based Software Testing (SBST) and the cost-effectiveness along with empirical evidence that was available regarding the SBST scalability and effectiveness of cost. This could also provide a new framework to drive the process of data collection for a systematic review and was taken to be the starting point for the guidelines and the manner in which these SBST techniques were assessed empirically. The intent was to help researchers in future to conduct empirical investigations in the SBST by means of providing unbiased empirical evidence and also by guiding them in the performance of certain well-designed and well-executed references to empirical investigations.

Wang et al., [10] had made a proposal of a new and practical guide to the SBSE community for being able to choose some quality indicators to assess the search that was Pareto-based in the context of software engineering. This was also a practical guide derived from the complementary empirical and theoretical methods mentioned below: 1) the key theoretical foundations for quality indicators; 2) an evidence from the extended review of literature; and 3) all evidence that had been collected from the extensive experiments that had been conducted for the evaluation of the eight indicators of quality from a total of four categories that had a total of six search algorithms that were Pareto-based by using a total of three industrial problems obtained from two different domains that were diverse.

Wang et al., [11] had proposed the solution which was resource-aware and multi-objective using a fitness function that was defined on the basis of four measures that were cost-effective. In a similar context, there was a new set of software releases that were tested based on a small set of the Video Conferencing Systems (VCSs) based hardware (test resources) found compatible by means of executing a new set of test cases that were cost-effective which were in an optimal order in a certain given test cycle that was constrained by means of a maximum allowed time budget and the available test resources. This was also evaluated in an empirical manner by employing seven different search algorithms and were compared with their current practice (Random Ordering (RO)). Results proved that the solution along with its best search algorithm (Random-Weighted Genetic Algorithm (RWGA)) and this will be able to improve current practice by means of bringing down on an average of about 40.6% of the time that had been used for the allocation of resource and the execution of the test cases with an improved usage of test resource by about 37.9% and a fault detection on an average by about 60%.

Guizzo et al., [12] had further introduced a new Hyper-heuristic for Integration and Test Order (HITO) issues. This included another new set of some well-designed steps that were based on a total of two different selection functions (the Choice Function and the Multi-Armed Bandit) for the purpose of choosing the ideal heuristic (a combination of both mutation and crossover operations) in each mating. In order to perform this type of selections, there was a measure of quality that had been proposed for assessing the performance of heuristics of a low level in the entire process of evolution. The HITO had been implemented by employing the NSGA-II and was further evaluated for solving the integration along with the problem of test order in all the seven systems. This hyper-heuristic was able to obtain the results that were the best for the systems on being compared to that of the traditional algorithms.

Panichella et al., [13] had improved the actual optimality of that of the Multi-Objective Genetic Algorithms (MOGAs) which was improved to a significant extent by means of diversifying solutions (the sub-sets of test suites) which were generated at the time of the search. More specifically, this work brought in the MOGA that was coined as the DIVersity based Genetic Algorithm (DIV-GA). This was based on the orthogonal design and the orthogonal evolution mechanisms that improved diversity. The results of this empirical work made on eleven programs proved that the DIV-GA was able to outperform the greedy algorithms and also the traditional MOGAs from the point of view of optimality. Furthermore, these solutions (of subsets of test suites) given by the DIV-GA could detect some more faults aside from the remaining algorithms at the same time keeping the cost of executing the same.

Saber et al., [14] had made a proposal of yet another novel hybrid algorithm for addressing the problem with three steps: the greedy algorithm which was for identifying good solutions in a quick manner, a Genetic Algorithm (GA) for the purpose of increasing the search space covered with a local search algorithm to refine solutions. This also demonstrated it by means of an empirical evaluation of a large scale making the method more reliable and robust. This was proposed in the scenario using four different objectives with time for default execution that was about 178% better in the hyper-volume compared to all algorithms that were state-of-the-art.

This was in response to a competitive market that had to be kept cost-effective with the software of good quality for which the testing and also the debugging has to be done independently making it quite expensive. For this, they had to explore the test cases for each product. There was a new GA-based framework that was proposed by Li et al., [15] to integrate the techniques of localization of software faults focusing on the specifications of the test being reused. There were case studies that made use of four product lines along with eight techniques of fault localization that had been conducted for demonstrating the framework and its effectiveness. Results proved that these test cases were generated in an easily reusable way (with suitable conversion) among products belonging to the same family that help in overall costs of testing and debugging.

Garousi et al., [16] had been motivated an industrial need for improving the practices of regression-testing which were in the context of the industrial software that was safety-critical in the domain of defence in Turkey. For this, there was an "action-research" that was conducted which was a collaborative project executed between the academia and industry. This selected a Multi-Objective Regression-Test selection framework (MORTO) and had adopted it to the context of the industry thus developing a GA that was custom-built. This was able to provide complete coverage of the needs and bringing about benefit and cost factors such as bringing down the test cases and increasing the detected faults for every test suite.

Agrawal and Kaur [17] had made a comparison of the Hybrid PSO and the Ant Colony Optimization. This was of major relevance in the field of software engineering. There were several experiments that were conducted using the MATLAB, and it was reported that the work had the underlying motivation of creating an awareness of two different aspects: the comparison of the performance of all metaheuristic algorithms and duly demonstrating the test case selection and its significance in the field of software engineering.

III. METHODOLOGY

Most of the problems in optimization have several conflicting objectives. The primary goal of multi-objective optimization was to optimize all these conflicting objectives in a simultaneous manner [18]. Generally, a problem of multi-objective minimization using the M decision variables along with the N objectives are stated as in (1 and 2):

$$\begin{aligned} &Minimize\ f_i(x) = 1,\ldots,N \\ &Where\ x = (x_1,\ldots,x_m) \in X \end{aligned} \quad (1)$$

$$\begin{aligned} &Subject\ to:\ g_i(x) = 0 \quad j = 1,\ldots,M \\ &\qquad h_k(x) \le 0 \quad k = 1,\ldots,K \end{aligned} \quad (2)$$

In this, the f_i denotes an ith objective function, the x denotes a decision vector representing the solution and X the parameter space. The functions of gj and the hk are equality and inequality. It has a desired solution as a "trade-off" which is a compromise among parameters. This type of an optimal solution to trade-offs among objects will constitute the Pareto front. There are several multi-objective deals of optimization that generate the Pareto front as well. The solution is normally supposed to be non-dominated in case it is impossible to be able to improve a single component without making it detrimental to the value of the other component.

The primary goal of this type of multi-objective optimization was to ensure the identification of well-established Pareto front which consists of solutions that are non-dominated. For the same purpose there had been a problem of multi-objective test case selection that was identified with the NSGA-II and SDS algorithm.

A. Multi-Objective Test Case Selection

The problem of this multi-objective test case was to select a new subset which was Pareto efficient and was based on several test criteria. This may be defined as below: Given: the test suite, T, a vector of the M objective functions, f_i, i = 1, 2… M. The Problem: was to find the subset of T, T', so that the T' was a new Pareto optimal set which was in respect to its objective functions, f_i, i = 1, 2… M. These objective functions were the descriptions of this test criteria. The subset t_1 will dominate t_2 at the time the decision vector for the t_1 ({f_1 (t_1) . . . f_M (t_1)}) will dominate the t_2. The subset resulting from the test suite T', as several other benefits with regard to regression testing [19].

For the purpose of this work, this will instantiate two of the objectives and their formulation along with its code coverage which is identified as a measure of test adequacy. Time is the other objective that needs be minimized for certain level of code coverage.

For this type of instantiation of problems, there needs to be a subset for the test suite which is s that has a coverage c_1 and the execution time t_1 on a Pareto frontier, which is:

• T1. None of the other subsets of s can get more coverage than the c_1 without having to spend more time than the t_1.

• T2. None of the other subsets of s are able to finish in a time that is less than t_1 at the same time getting a coverage found to be more or equal to c_1.

This denotes the actual implication of the Pareto optimality. As opposed to getting one single answer for approximating the global optimum within the search space for one single objective, this can get another new set of points in an optimal manner. Every member in the Pareto frontier will be a candidate solution that does not have any improvement.

B. Non-dominated Sorting Genetic Algorithm II (NSGA II)

The NSGA II has been a new and evolutionary algorithm that is multi-objective and based on sorting that is non-dominated. The algorithm makes use of the elitist and non-dominated sorting. There is yet another objective function which is in terms of the variables coded in the algorithm. The members belonging to the Pareto-front are part of a non-dominated set which is obtained on the basis of convergence. The choice is made for the crowded comparison based on ranking (based on the level of non-domination) and also the crowding distance which is obtained on the convergence of this algorithm [20]

There is also a random generation of parent population (or solution) P of a size N. For the purpose of identifying the level of non-domination, every solution is duly compared to yet another solution and it is further checked if the solution is able to satisfy the rules as in (3):

$$Obj.1[i] > Obj.1[j] \; and \; Obj.2[i] \geq Obj.2[j],$$
$$or \; Obj.1[i] \geq Obj.1[j] \; and \; Obj.2[i] > Obj.2[j], \quad (3)$$

Where the i and j are the chromosome numbers.

Now, in case the rules are met, its chosen solution will be marked as being dominated. If not, there is a chosen solution will be marked as non-dominated. In the case of the first sorting, all non-dominated solutions (N1) were assigned to a rank 1. From the rest of the N–N1 dominated solution taken from the first sorting gets sorted to all non-dominated solutions found in second sorting assigned to rank 2. The process will continue until such time all the solutions duly get ranked. Each solution will be given a new fitness which is equal to the non-domination task level (rank 1 is the best, rank to the next and so on). These solutions will belong to a certain rank or a level of non-domination and no solution that is better compared to that of the others. As soon as the rank for the solution was identified as the crowding distance for the solution.

The crowding distance is the average distance of the points that are located on each side of the solution point. For computing the crowded distance, the populations in the non-dominated set that were sorted in ascending order of magnitude ideally in accordance with every objective function. After this, there had been a new boundary solution for every objective function which was the one that had both the largest and the smallest of infinity values. The remainder of these intermediate solutions had been assigned to the value that was equal to the absolute and normalized difference found in the function value for two of the adjacent solutions. In order to solve this problem of optimization employing the GA, there is a need for fitness value. These fitness values were objective function values. Thus, we will have to use a function or an equation that can be related to the decision variable with the objective.

Crossover: In the case of the NSGA II Simulated Binary Crossover (SBX) which is used, the work is completed with two different parent solutions creating two different offspring as below [21]:

Step 1: Select a random number $u_i \in [0, 1]$,

Step 2: Calculate this by using equation (4),

Step 3: Compute the offspring by using equation (5).

The mathematical formulation for this may be given as below:

$$\beta_{q_i} = \begin{cases} (2u_i)^{\frac{1}{\eta_c+1}} & if \; u_i \leq 0.5; \\ \left(\frac{1}{2(1-u_i)}\right)^{\frac{1}{\eta_c+1}} & otherwise. \end{cases} \quad (4)$$

$$x_i^{(1,t+1)} = 0.5\left[\left(1+\beta_{q_i}\right)x_i^{(1,t)} + \left(1-\beta_{q_i}\right)x_i^{(2,t)}\right],$$
$$x_i^{(2,t+1)} = 0.5\left[\left(1-\beta_{q_i}\right)x_i^{(1,t)} + \left(1+\beta_{q_i}\right)x_i^{(2,t)}\right]. \quad (5)$$

Wherein,

u_i: denotes the random number so that $u_i \in [0, 1]$,

η_c : denotes the distribution index (the Non-negative real number),

$x_i^{(1,t)}$ & $x_i^{(2,t)}$: denotes the parent solutions,

$x_i^{(1,t+1)}$ & $x_i^{(2,t+1)}$: denotes the offspring solutions.

Mutation: In the case of the NSGA II the Polynomial mutation is employed to mutate every solution individually. For example, if one parent solution provides an offspring it is only after it is mutated. This is mathematically depicted as (6 and 7):

$$y_i^{(1,t+1)} = x_i^{(1,t+1)} + (x_i^{(U)} - x_i^{(L)})\,\bar{\delta}_i \qquad (6)$$

Wherein,

$$\bar{\delta}_i = \begin{cases} (2r_i)^{1/(\eta_m+1)} - 1 & if\ r_i < 0.5, \\ 1-[2(1-r_i)]^{1/(\eta_m+1)}, & if\ r_i \geq 0.5 \end{cases} \qquad (7)$$

In this,

r_i: denotes the random number so that $u_i \in [0, 1]$,

η_m: denotes the distribution index (the non-negative real number),

$x_i^{(1,t+1)}$: denotes the parent solution,

$x_i^{(U)}$: denotes the upper bound for its parent solution,

$x_i^{(L)}$: denotes the lower bound for its parent solution,

$y_i^{(t+1)}$: denotes the Offspring solution.

The Crowded Tournament Selection: in order to obtain an estimation of the solutions and their density found to be close to a solution i within the population, it can now an average of both solutions of both sides of solution i along every objective. Quantity d_i will denote its crowding distance. The algorithm which follows will be employed for computing its crowding distance for each point that is found as in set Ƒ.

The assignment procedure: Crowding-sort (Ƒ, $<_c$)

Step 1: Call the actual number of the solutions found in Ƒ as $l = |F|$. For every i found within the set, initially assign $d_i = 0$.

Step 2: For every such objective function m =1, 2…M, sort a set in the worse order of f_m. Now find the sorted indices of vector I_m = sort (f_m, >).

Step 3: For m = 1….M, a large distance to edge solutions $d_{I_1^m} = d_{I_1^m} = \infty$ to be assigned and other solutions, j = 2 to (l – 1) is assigned as in (8):

$$d_{I_j^m} = d_{I_j^m} + \frac{f_m^{(I_{j+1}^m)} - f_m^{(I_{j-1}^m)}}{f_m^{\max} - f_m^{\min}} \qquad (8).$$

C. Stochastic Diffusion Search (SDS) Algorithm

There is also the SDS algorithm introduced to be a new probabilistic approach in order to solve the recognition and the matching problems. The SDS was a population-based multi-agent global search algorithm which is employed in a distributed mode of computation using a framework which was strong that described the algorithm and its behavior by means of investigating the allocation of resources, the robustness, the linear complexity of time, the final criteria of minimal convergence and also the convergence to the global optimum [22].

This SDS algorithm will commence the search or the optimization by means of population initialization (such as miners found in the metaphor of the mining game). For any of the other SDS searches, every agent will maintain a new hypothesis which is h that defines any of these solutions to a possible problem. In this mining game analogy, the agent hypothesis also identifies a hill and once the initialization is complete, there will be two phases that follow:

- The Test Phase (such as the testing of availability of gold)
- The Diffusion Phase (such as the information exchange and its congregation)

The SDS algorithm is depicted as below:

Initialisin g agents ()

While (*stopping condition is not met*)

Testing hypotheses ()

Diffusion hypotheses ()

End

In a **test phase**, the SDS will checks if the agent hypothesis has been successful which is not done by performing an evaluation of a partial hypothesis that which returns the Boolean value. Once this is done, in an iteration, a contingent on a strategy of recruitment employed will have successful hypotheses that diffuse across the population wherein the information on solutions will spread through the whole agent population. For a test phase, every agent will perform the partial function evaluation, pFE, and this denotes the function of the agent hypothesis; pFE = f (h). In the case of the mining game, there is a function of partial evaluation which entails the mining of a region on the hill that was chosen randomly and further defined by the hypothesis of the agent (as opposed to the mining of the regions on the same hill).

In the case of the **diffusion phase**, every single agent will recruit one more agent to interact with the communication hypothesis. For a metaphor of the mining game, there was a diffusion performed by means of communicating the hill hypothesis.

Relate: This denotes a phase that is optional and introduced for multiple models. The technique will permit a large degree of agent re-allocation along with the maintenance of the multiple clusters of all active agents that have several good hypotheses. The relate phase further helps in the dynamic search spaces permitting clusters of agents to re-align themselves successfully with the correct hypothesis. There are two modes to the relate phase: the context-free and the context-sensitive [23].

Halting: Once each test with the diffuse iteration (and as a new option the related phase) is complete, this SDS process determines as to where the agent population has been reached with a new state to determine its search. During the time of their initial iterations, there are some mores such active agent populations that are small until the agent is able to reach a hypothesis that is optimal; a population (cluster) which is around this hypothesis will grow at the time there are more agents to be recruited.

On the basis of the search space and model parameters, the cluster around an optimal hypothesis (the hypotheses in the case of a relate phase) will get stabilized. Identified are two different criteria that are applied for the determination during the SDS process and its search which ends: this is the weak halting and the strong halting criteria.

The Weak Halting Criteria: this indicates the SDS to be able to stop during a percentage of agents who are found as active in spite of the hypothesis. This has been taken to be a population of active agents who are steady within a margin of tolerance for a certain number of iterations. Once these criteria are met, the search ceases.

The Strong Halting Criteria: This indicates the state of halt that had been connected to the agent percentage that falls under its largest cluster. This denotes the hypothesis at the time the agents get clustered and have a similar threshold or rule of tolerance having a state of weak halting and when looking at the agent percentage that is active in the largest cluster.

IV. RESULTS AND DISCUSSION

In this section, the reference, NSGA and stochastic diffusion methods are used. Experiments are evaluated using 10000 to 80000 cost. The print tokens and space as shown in tables 1 & 2 and Fig 1 & 2.

Table 1 Print Tokens

Cost	Reference	NSGA	Stochastic Diffusion
10000	0.8	0.76	0.78
20000	0.8	0.77	0.79
30000	0.81	0.78	0.79
40000	0.85	0.8	0.82
50000	0.86	0.81	0.85
60000	0.88	0.84	0.86
70000	0.87	0.85	0.88
80000	0.88	0.87	0.88

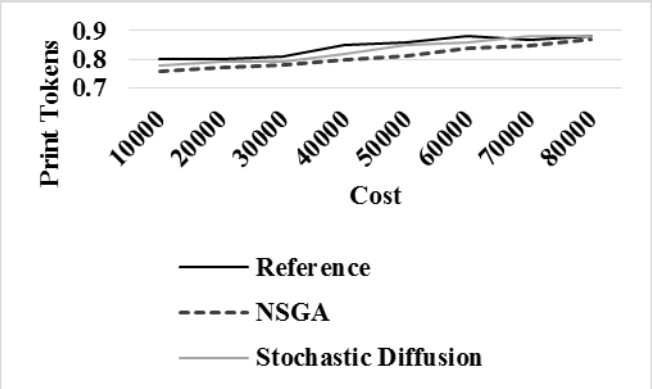

Fig 1 Print Tokens

From the Fig 1, it can be observed that the stochastic diffusion has higher print tokens by 2.59% for 10000 cost, by 1.27% for 30000 cost, by 4.82% for 50000 cost and by 3.47% for 70000 cost when compared with NSGA respectively. The stochastic diffusion has lower print tokens by 2.53% for 10000 cost, by 2.5% for 30000 cost, by 1.17% for 50000 cost and by 1.14% for 70000 cost when compared with reference respectively.

Table 2 Space

Cost	Reference	NSGA	Stochastic Diffusion
10000	0.77	0.74	0.76
20000	0.78	0.75	0.76
30000	0.8	0.76	0.77
40000	0.82	0.78	0.81
50000	0.84	0.79	0.82
60000	0.86	0.81	0.85
70000	0.86	0.83	0.85
80000	0.87	0.85	0.86

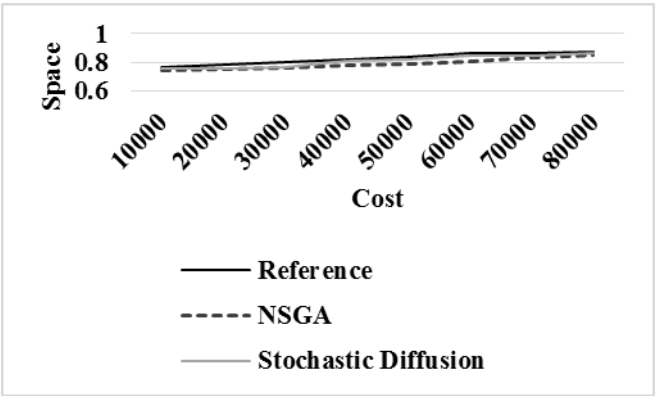

Fig 2 Space

From the Fig 2, it can be observed that the stochastic diffusion has higher space by 2.67% for 10000 cost, by 1.31% for 30000 cost, by 3.73% for 50000 cost and by 2.38% for 70000 cost when compared with NSGA respectively.

The stochastic diffusion has lower space by 1.31% for 10000 cost, by 3.82% for 30000 cost, by 2.41% for 50000 cost and by 1.17% for 70000 cost when compared with reference respectively.

V. CONCLUSION

Software testing indicates the actual process of experimenting of program with the input data for the purpose of observing failure. This testing will be able to identify faults and also remove them to improve software quality. Testing further measures the capacity of achieving the correctness, testability, reusability, maintainability, usability, and reliability. The selection of a test case can be called a classic technique which chooses one more such new subject for the current test cases used for execution because of tight deadlines and limited budgets. This code coverage is the state of practice which is made among the heuristics of the test case selection. In recent literature, there is a 'test case diversity' that has been observed to be a very promising approach. There was another multi-objective selection of test case that aims at optimizing the various objective functions simultaneously.

The NSGA-II denotes the algorithm which is given in order to solve problems in multi-objective optimization. The NSGA-II will employ faster processes of various probabilistic approached for solving the pattern recognition which is best-fit. The SDS has been the algorithm multi-agent global search optimization which is distributed for computation based in interaction of simple agents. Results proved that the stochastic diffusion can have higher print tokens by about 2.59% for the 10000 cost, by about 1.27% for the 30000 cost, by about 4.82% for the 50000 cost and further by about 3.47% for about 70000 cost on being compared with the NSGA respectively. This stochastic diffusion also has some more lower print tokens by about 2.53% for the 10000 cost, by about 2.5% for the 30000 cost, by about 1.17% for the 50000 cost and finally by 1.14% for the 70000 cost on being compared to the reference respectively.

REFERENCES

1. Srivastava, P. R., & Kim, T. H. (2009). Application of genetic algorithm in software testing. International Journal of software Engineering and its Applications, 3(4), 87-96.
2. Sapna, P. G., & Mohanty, H. (2010, September). Clustering test cases to achieve effective test selection. In Proceedings of the 1st Amrita ACM-W Celebration on Women in Computing in India (p. 15). ACM.
3. Pravin, A., & Srinivasan, S. (2013). S. Srinivasan:?Effective Test Case Selection and Prioritization in Regression Testing. In Journal of Computer Science, 9 (5), 654-659.
4. Narciso, E. N., Delamaro, M. E., & Nunes, F. D. L. D. S. (2014). Test case selection: A systematic literature review. International Journal of Software Engineering and Knowledge Engineering, 24(04), 653-676.
5. Mondal, D., Hemmati, H., & Durocher, S. (2015, April). Exploring test suite diversification and code coverage in multi-objective test case selection. In 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST) (pp. 1-10). IEEE.
6. Matnei Filho, R. A., & Vergilio, S. R. (2016). A multi-objective test data generation approach for mutation testing of feature models. Journal of Software Engineering Research and Development, 4(1), 1-29.
7. Kazmi, R., Jawawi, D. N., Mohamad, R., Ghani, I., & Younas, M. (2017). A Test Case Selection Framework and Technique: Weighted Average Scoring Method. Journal of Telecommunication, Electronic and Computer Engineering (JTEC), 9(3-4), 15-22.
8. Shin, K. W., & Lim, D. J. (2018). Model-based automatic test case generation for automotive embedded software testing. International Journal of Automotive Technology, 19(1), 107-119.
9. Ali, S., Briand, L. C., Hemmati, H., & Panesar-Walawege, R. K. (2010). A systematic review of the application and empirical investigation of search-based test case generation. IEEE Transactions on Software Engineering, 36(6), 742-762.
10. Wang, S., Ali, S., Yue, T., Li, Y., & Liaaen, M. (2016, May). A practical guide to select quality indicators for assessing pareto-based search algorithms in search-based software engineering. In 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE) (pp. 631-642). IEEE.
11. Wang, S., Ali, S., Yue, T., Bakkeli, Ø., & Liaaen, M. (2016, May). Enhancing test case prioritization in an industrial setting with resource awareness and multi-objective search. In Proceedings of the 38th International Conference on Software Engineering Companion (pp. 182-191). ACM.
12. Guizzo, G., Fritsche, G. M., Vergilio, S. R., & Pozo, A. T. R. (2015, July). A hyper-heuristic for the multi-objective integration and test order problem. In Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation (pp. 1343-1350). ACM.
13. Panichella, A., Oliveto, R., Di Penta, M., & De Lucia, A. (2015). Improving multi-objective test case selection by injecting diversity in genetic algorithms. IEEE Transactions on Software Engineering, 41(4), 358-383.
14. Saber, T., Delavernhe, F., Papadakis, M., O'Neill, M., & Ventresque, A. (2018, July). A hybrid algorithm for multi-objective test case selection. In 2018 IEEE Congress on Evolutionary Computation (CEC) (pp. 1-8). IEEE.
15. Li, X., Wong, W. E., Gao, R., Hu, L., & Hosono, S. (2018). Genetic algorithm-based test generation for software product line with the integration of fault localization techniques. Empirical Software Engineering, 23(1), 1-51.
16. Garousi, V., Özkan, R., & Betin-Can, A. (2018). Multi-objective regression test selection in practice: An empirical study in the defense software industry. Information and Software Technology, 103, 40-54.
17. Agrawal, A. P., & Kaur, A. (2018). A comprehensive comparison of ant colony and hybrid particle swarm optimization algorithms through test case selection. In Data Engineering and Intelligent Computing (pp. 397-405). Springer, Singapore.
18. Gonsalves, T., & Itoh, K. (2010, March). Multi-objective optimization for software development projects. In International multiconference of engineers and computer scientists (pp. 17-19).
19. Yoo, S., & Harman, M. (2007, July). Pareto efficient multi-objective test case selection. In Proceedings of the 2007 international symposium on Software testing and analysis (pp. 140-150). ACM.
20. Padhee, S., Nayak, N., Panda, S. K., Dhal, P. R., & Mahapatra, S. S. (2012). Multi-objective parametric optimization of powder mixed electro-discharge machining using response surface methodology and non-dominated sorting genetic algorithm. Sadhana, 37(2), 223-240.
21. Golchha, A., & Qureshi, S. G. (2015). Non-dominated sorting genetic algorithm-II–A succinct survey. International Journal of Computer Science and Information Technologies, 6(1), 252-255.
22. Al-Rifaie, M. M., & Bishop, J. M. (2013). Stochastic diffusion search review. Paladyn, Journal of Behavioral Robotics, 4(3), 155-173.

al-Rifaie, M. M., Bishop, J. M., & Blackwell, T. (2012). Information sharing impact of stochastic diffusion search on differential evolution algorithm. Memetic Computing, 4(4), 327-338.