A Survival Study for Software Test Suite Generation using Derived Genetic Algorithm

V. Sangeetha
Department of Computer Science
Periyar University College of Arts & Science
Pappireddipatti, Dharmapuri, Tamilnadu, India

T. Ramasundaram
Department of Computer Science
Sri Vijay Vidyalaya College of Arts & Science
Nallampalli, Dharmapuri, Tamilnadu, India

Abstract- **Software has evolved as an innovative solution for several applications. Unit test suites are mainly used for increasing the software quality using the techniques like search-based software test. Search-based testing generates the unit test suites automatically for object oriented code. Many testing tools like unit testing, integrity testing, is redesigned to check the correctness of software results and to produce the test suites with high coverage. However, performing specific test is impractical due to higher execution time and less coverage capability. In this work, Search based Test Suite Generation using Derived Genetic Algorithm (STSG-DGA) is designed to increase the coverage and to reduce the redundancy for test suite generation. Initially, an initial population of randomly produced candidate solutions is used as search operators. Then, the parent selection is carried out based on fitness function. After that, reproduction is performed by crossover and mutation operation with probabilities. Finally, fitness of population gets increased in DGA and the process gets repeated till the optimal solution is found. Our research work helps to reduce execution time and computational complexity with minimum redundancy for test suite generation.**

Keywords- Software, Coverage, Test Suite Generation, Mutation, Crossover, Fitness, Derived Genetic Algorithm

1. INTRODUCTION

Search-based software engineering has been applied to different tasks in software development. Software testing is one of the most successful one. The key task in software testing where search-based techniques suited is generation of unit tests in automatic manner. In search-based software testing, the testing problem is taken as a search problem. The main process is to generate the group of test cases where the code coverage gets increased. A code coverage criterion explains the structural features of system under test (SUT) by test suite.

High Level Hyper-Heuristic (HHH) strategy (Kamal Z.Zamil, Basem Y. Alkazemi, & Graham Kendall, 2016)applied four low level meta-heuristics with Teaching Learning based Optimization, Global Neighbourhood Algorithm, Particle Swarm Optimization and Cuckoo Search Algorithm to address t-way test suite generation issues. An automatic software test data generation gets trapped in local optimal solution resulting in population aging. Though the strategy reduces GA-based software testing merits, cost gets increased. Memetic Algorithm for Test Suite Optimization extended Genetic Algorithm (Gordon Fraser, Andrea, & Phil Mc Minn, 2015) using many local search operators. These operators were designed to optimize the primitive values that allow the search for test cases to function in effective manner. Memetic Algorithm for Test Suite Optimization was more beneficial for object-oriented software that handled whole test suites. However, test suite generation with high coverage lack knowledge about test suite to optimal values during run time. Parallel Genetic Algorithm based on Spark (PGAS) (Rong-Zhi Qi, CCF, Zhi-Jian Wang, & Shui-Yan Li, 2016) study specific coverage goals for finding better traditional approach. But, PGAS results in the increased execution cost.

A two-phase parallelization algorithm called Spark (Andrea Arcuri, & Gordon Fraser, 2014) based on parallel computing platform solved heavy computation challenge with test suite size. The algorithm employed fitness evaluation and genetic operation for pair wise test suite generation. For providing the validity and

usefulness of whole test suite approach, search-based test generation was not evolved. Regenerate Genetic Algorithm (RGA) (Shunkun Yang, Tianlong Man, Jiaqi Xu, Fuping Zeng , & Ke Li, 2016) defined population aging in software testing with local optimal solution through triggering regeneration. The algorithm proved more efficient with better search efficiency, test coverage and reduces the test case frequency. Evolutionary whole test suite generation (EVOSUITE) (Gordon Fraser, & Andrea Arcuri, 2013)used an evolutionary technique than measuring the test case individually using fitness function. EVOSUITE with randomly generated test suite is selected as an initial population. Genetic Algorithm was applied for optimizing the selected coverage criterion in test suite generation. But, coverage optimization capability of test suite remained unsolved.

Complete controllable test suite for distributed testing (Robert M. Hierons , 2015) performed the mapping of Finite State Machine (FSM) to partial FMS guaranteeing the distributed testing. The method used the state counting to test multiport deterministic FSM for distributed testing. However, restricting testing to controllable test cases remained major concern. Software testing with quality-assurance technique generates effective test suites but computationally expensive. Automated product-line test-suite generation method (Johannes Burdek, Malte Lochau, Stefan Bauregger, Andreas Holzer, Alexander von Rhein, Sven Apel, & Dirk Beyer, 2015) reuses the information for different test goals with multi-goal test coverage assurance. The method used similarity information with the systematic reuse of information between the test cases for ordering the test goals. A new strategy (Bestoun S. Ahmed, Taib Sh. Abdulsamad, & Moayad Y. Potrus, 2015) generate the combinatorial test suite by cuckoo search ideas. Cuckoo Search is employed for the implementation to design the optimized combinatorial sets. However, the coverage is not at required level.

This paper is organized as follows: Section II discusses related works of software test suite generation, Section III discusses existing test suite generation techniques, Section IV explains about Search based Test Suite Generation using Derived Genetic Algorithm, Section V identifies the possible comparison between them, Section VI discusses the existing techniques limitations and future works and Section VII concludes the paper. The key area of research is given to increase the coverage capability and to reduce the redundancy by DGA.

2. RELATED WORKS

A search-based EVOSUITE test generation tool (Gordon Fraser, & Andrea Arcuri, 2015) combines two optimizations. It eliminates the redundant test executions on mutants through observing the state infection conditions. Test suite generation is used to increase the test suites with higher number of mutants. But, the computational complexity is higher. A new fitness function of meta-heuristic algorithms is designed (Le Thi My Hanh, Nguyen Thanh Binh, & Khuat Thanh Tung, 2016) for the test data generation depending on mutation method using Simulink model. The test suites generated efficiency addresses the criterion with counterexample-based test generation and random generation approach (Gregory Gay, Matt Staats, Michael Whalen, & Mats P. E. Heimdahl, 2015). But, the coverage capability is not at required level. An efficient plan is designed (Akram Kalaee, & Vahid Rafe, 2016) using reduced ordered binary decision diagram (ROBDD) for minimum test suite generation with high coverage. Particle swarm optimization (PSO) algorithm is used to choose the optimal test with pair wise combinations. But, the redundancy is not reduced.

A framework for binary search is designed by (Sami Beydeda, & Volker Gruhn, 2003) with path-oriented test case generation. In addition, it also introduced the binary search-based test case generation (BINTEST) algorithm. The lack of knowledge is in cost analysis and factors are linked with the maintenance of GUI-based tests in industrial practice. Visual GUI Testing carried out by (Emil Alegroth, Robert Feldt, & Pirjo Kolstrom, 2016) obtains the data about the maintenance costs and feasibility. Whole test suite generation (Jose Miguel Rojas, Mattia Vivanti, Andrea Arcuri, & Gordon Frase, 2016) manages all the test suites with all objectives simultaneously. The coverage attained in whole test suite generation is higher than targeting individual coverage objectives. But, the execution time is higher.

ACO algorithm is redefined into the discrete version (Chengying Mao, Lichuan Xiao, Xinxin Yu, & Jinfu Chen, 2015) for the test data generation in structural testing. The technical roadmap of ACO algorithm and test process is designed. But, the effectiveness was not increased. A mathematical model for test data generation

is designed for multiple paths coverage. A multi-population genetic algorithm is introduced (Xiangjuan Yao, & Dunwei Gong, 2014) with individual sharing for addressing the established model. An on-the-fly algorithm creates test suite with all feasible coverage items. The algorithm(Anders Hessel, & Paul Pettersson, 2007) presents the traces with path fulfilling items without redundant paths. However, the redundancy is not reduced. A Fuzzy Expert System is designed (Chin-Yu Huang, Chung-Sheng Chen, & Chia-En Lai, 2016) into test suite reduction techniques. But, the execution time is not minimized.

3. EXISTING SOFTWARE TEST SUITE GENERATION TECHNIQUES

Testing is the method of estimating the system functionality to identify the gaps, errors, missing necessities and other features. Testing is the important one for software development process though it is manual and costly. Software development process guarantees sound software operation. A test design method chooses the test cases by sampling mechanism. The process optimizes test cases to attain optimum test suite through removing the time and cost of testing phase in software development. Software testing is the activity of executing the system for identifying the failures.

3.1. Memetic Algorithm for Whole Test Suite Generation

Search-based testing uses optimization methods like Genetic Algorithms for test case generation. The test case generation issues are complicated than procedural code. For generating the tests with all branches in class, the class is instantiated and method call sequence is not generated for placing the object in particular state. The method needs objects as parameters or primitive values like integers and strings. EVOSUITE tool utilizes Genetic Algorithms to generate whole test suite with number of test cases.

A Memetic Algorithm (MA) combines both the global and local search where the individuals of population in global search algorithm have chance for local search development. With Lamarckian-style learning, local development by individuals is determined in genotype and sends to next generation. Local search uses the values in one particular test case of test suite. When local search is used in particular test case, EVOSUITE iterates over statement series from last to first. Every statement uses local search based on statement types like primitive statements, method statements, constructor statements, field statements and array statements. Fitness value evaluation after local search operator needs the partial fitness evaluation. EVOSUITE collects last execution trace with every test case through which the fitness value are computed. When a test case is changed in search process by regular mutation or by local search, the cached execution trace is removed. Fitness calculation for local search needs one test from test suite to execute. With Baldwinian learning, the development is determined with fitness value while genotype was unchanged. Baldwin effect explains the individuals with more potential for improvement during evolution and smoothes the fitness landscape. MAs for test generation were determined for procedural code test generation in different areas like combinatorial testing.

3.2. Tabu Search Hyper-Heuristic Strategy for t-way Test Suite Generation

T-way testing is a method to generate test suite for identifying the interaction faults. The generation of t-way test suite is NP hard problem. The t-way plans are fast and generate optimal solutions with the restrictions on configurations and interaction strength. Computational t-way plans remove the restrictions for arbitrary configurations at cost of non-optimal solution. The hyper-heuristic based strategy called High Level Hyper-Heuristic (HHH) is designed for combinatorial t-way test suite generation. HHH uses Tabu Search (TS) as high level meta-heuristic (HLH) and controls with four low level meta-heuristics (LLH) namely, Comprising Teaching Learning Based Optimization (TLBO), Global Neighbourhood Algorithm (GNA), Particle Swarm Optimization (PSO), and Cuckoo Search Algorithm (CS) as described in the Figure 1.

HHH is hyper-heuristic based strategy used for solving the t-way test suite generation issues. A new hyper-heuristic approach for the meta-heuristic selection and acceptance mechanism are based on three operators (i.e. improvement, diversification and intensification) that integrated into the Tabu Search HLH. The improvement operator checks for the improvements in objective function. Diversification operator measures

how diverse the current and the previously generated solution are against the population of potential candidate solutions. An intensification operator evaluates the current and earlier generated solution against the population.

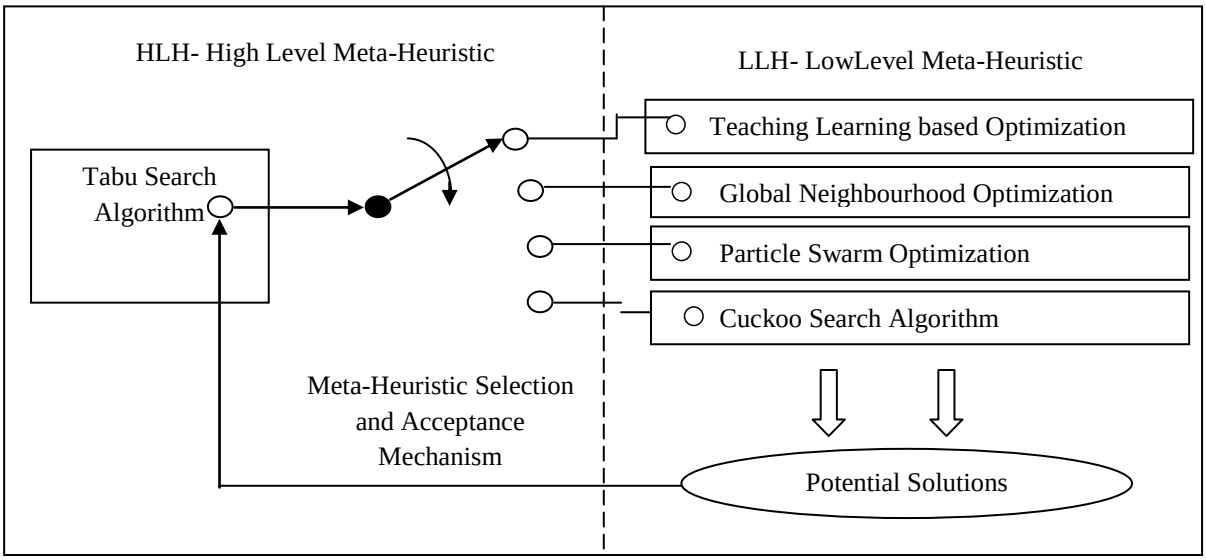

Figure 1. High Level Hyper-Heuristic (HHH) Strategy

3.3. Parallel Genetic Algorithm Based on Spark (PGAS) for Pairwise Test Suite Generation

Pairwise testing reduces the combination explosion exhaustive testing issues. Pairwise testing is an efficient testing plan for systems. A parallel genetic algorithm depending on Spark termed PGAS increases the pairwise test suite generation process. Parallelism increases the performance and quality of solutions. GA is parallelizable as the fitness evaluation and evolution process contains the genetic operation iteration in parallel. Spark is a fast cluster computing platform for managing the GA parallelization.

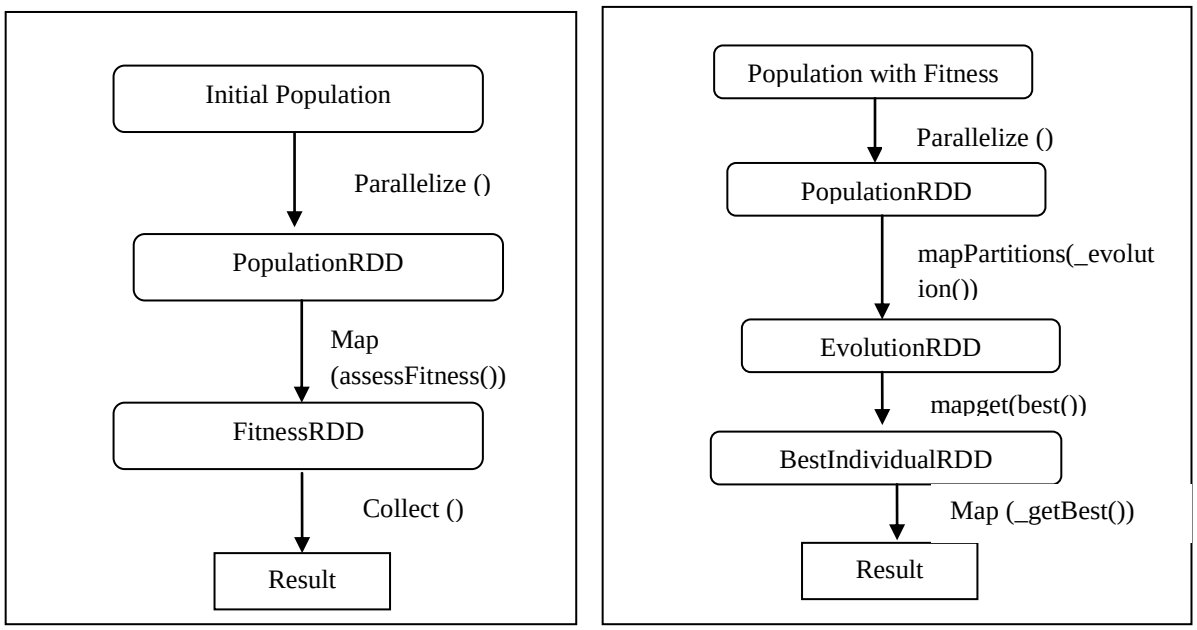

Figure 2. RDD Lineage Graph for Phase 1 and Phase 2

Spark explained about two phase parallelization, namely fitness evaluation parallelization and genetic operation parallelization. The former one estimates the individual's fitness value in parallel. The latter one divides the population into different slices that developed independently.

While generating the test suite with pairwise testing, input SUT is modelled as parameters with one or more assumption values. Pairwise testing selects a subset from complete set of parameter value mixture where all parameter value pairs are in a subset. The selected parameter value mixture generates test case for SUT. Spark depends on master slave distributed computing model suitable for global and distributed model of GA parallelization. Spark is used to parallelize GA and execute two-phase parallelization algorithms, namely fitness evaluation and genetic operation for increasing the GA effectiveness in searching near-minimum test suite.

From Figure 2, fitness evaluation parallelization is used to parallelize the initial population into resilient distributed dataset (RDD) and calculate individual's fitness value on many workers. Initial population is parallelized into population RDD by parallelize() technique of Spark. After that, a map(_assessFitness()) transformation changes the population RDD into fitness RDD with individual and fitness value pairs. The function assessFitness() calculates the individual's fitness value with driver program to run on the cluster. The collect() gathers the pairs and send to the driver.

4. SEARCH BASED TEST SUITE GENERATION USING DERIVED GENETIC ALGORITHM (STSG-DGA)

Search based Test Suite Generation using Derived Genetic Algorithm (STSG-DGA) is designed to increase the coverage and to reduce the redundancy for test suite generation. In STSG-DGA, Derived Genetic algorithm (DGA) is used for finding the optimal solution with minimum redundancy and higher coverage. The derived genetic algorithm is a method for randomized search by addressing the optimization issues. Initially, an initial population of randomly produced candidate solutions is used as search operators in DGA. Then, the parent selection is carried out based on fitness function. After that, reproduction is performed by crossover and mutation operation with probabilities. Finally, fitness of population gets increased in DGA and the process gets repeated till the optimal solution is found. Figure 3 explained the flow process of derived genetic algorithm.

3.1. Population Initialization

In Initialization process, many candidate solutions are created arbitrarily and generation starts with iteration '0'.

3.2. Fitness function

The fitness function is used for increasing the coverage. Fitness function to generate the tests for branch coverage combines the approach-level and the branch distance.

3.3. Crossover and Mutation

The new candidate solutions end the global search through the equivalent crossover operator. Two chromosomes are taken for finding the two offspring. Mutation helps in restoring lost candidate solution in the population. In mutation, a candidate is chosen from the chromosome attained in past generation and gene value is varied for creating the new offspring.

Randomly chosen value

$$< a_1, a_2, a_3, \ldots\ldots a_p \ldots\ldots a_n >$$

If the new offspring is not an optimal solution, it is removed and the iterations get repeated till it obtains the optimal solution. The mutation operation comes out from local optima findings and search for the global optima.

5. COMPARISON OF SOFTWARE TEST SUITE GENERATION USING ACO AND GENETIC ALGORITHM USING DIFFERENT TECHNIQUES

In order to compare the software test suite generation using different techniques, number of test cases is taken to perform this experiment. Various parameters are employed to increase the coverage capability with minimal execution time for software test suite generation.

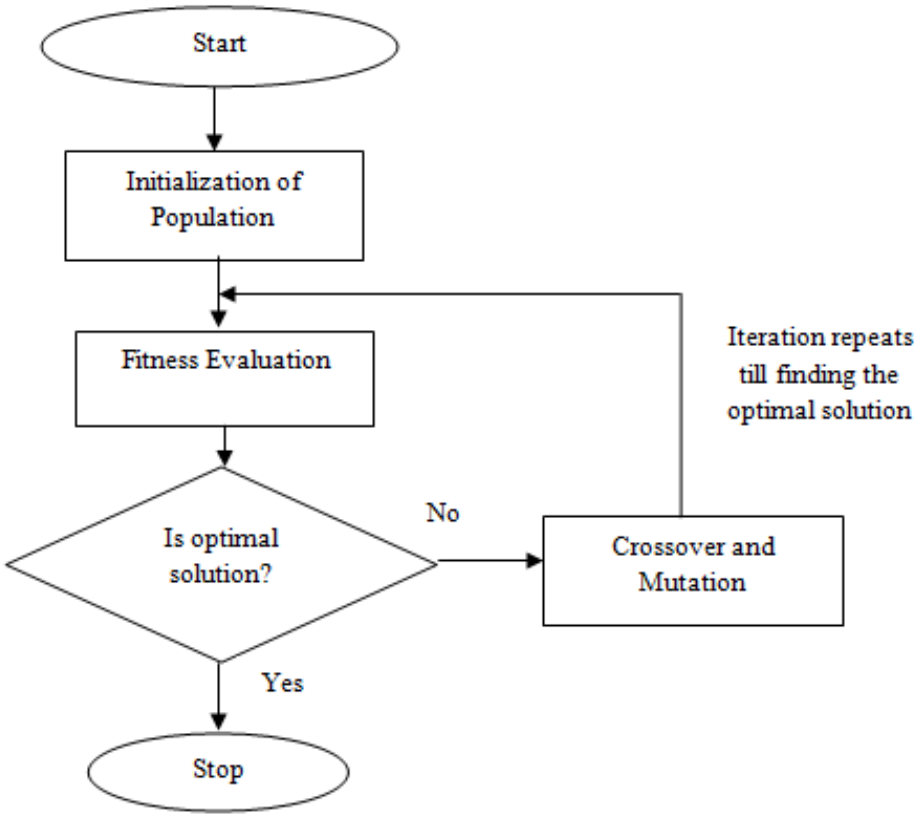

Figure 3. Flowchart for Derived Genetic Algorithm Process

3.1. Coverage

Coverage is a measure of test suite quality that denotes how much of program's behaviour is used by execution sets. Coverage is used in testing process to evaluate test suites and to generate test suites. It is measured in terms of percentage (%).

Table 1 explains the coverage for different number of test cases in the range of 10 to 100.The coverage comparison takes place on existing Memetic Algorithm, HHH strategy, PGAS and proposed STSG-DGA. Figure 4measures the coverage of proposed STSG-DGA and existing techniques. Coverage of proposed STSG-DGA is comparatively higher than that of HHH strategy, Memetic Algorithm and PGAS. Research in proposed STSG-DGA has 8.13% higher coverage than Memetic Algorithm, 20.32% higher coverage than HHH strategy and 34.30%higher coverage than PGAS.

Table : 1 Tabulation of Coverage

Number of test cases (Number)	Coverage (%)			
	STSG-DGA	Memetic Algorithm	HHH Strategy	PGAS
10	80	Memetic Algorithm	65	55
20	81	72	66	57
30	83	75	68	59
40	85	76	69	61
50	86	78	71	64
60	87	80	73	67
70	89	81	75	69
80	91	83	77	71
90	93	85	79	73
100	95	86	81	75

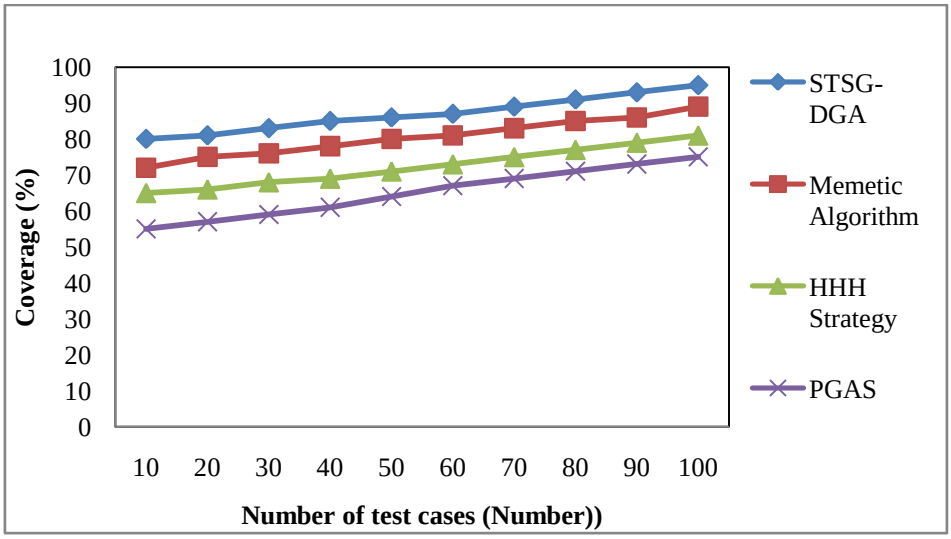

Figure 4. Measurement of Coverage

3.2. Execution time

Execution time is defined as the time taken for the software test suite generation process. Execution time is the difference of ending time and starting time of software test suite generation as formulated in equation 1. It is measured in terms of milliseconds (ms). When lower the execution time, the method is said to be more efficient.

$$Execution\ Tme = Ending\ Time - Starting\ Time\ of\ Software\ Test\ Suite\ Generation \quad (1)$$

Table 2 explains the execution time for different number of test cases in the range of 10 to 100. The execution time comparison takes place on existing Memetic Algorithm, HHH strategy, PGAS and proposed STSG-DGA.Figure5 measures the execution time of proposed STSG-DGA and existing techniques. Execution time of proposed STSG-DGA is comparatively lesser than that of Memetic Algorithm, HHH strategy and PGAS. The current research in proposed STSG-DGA takes 22.10% lesser execution time than Memetic Algorithm, 38.74% lesser execution time than HHH strategy and 44.01% lesser execution time than PGAS.

Table : 2 Tabulation of Execution Time

Number of test cases (Number)	Execution Time (ms)			
	STSG-DGA	Memetic Algorithm	HHH Strategy	PGAS
10	10	15	21	25
20	12	18	24	27
30	15	20	27	30
40	17	22	29	32
50	19	25	32	35
60	22	28	35	38
70	25	31	38	41
80	28	33	41	43
90	31	36	43	46
100	34	39	47	51

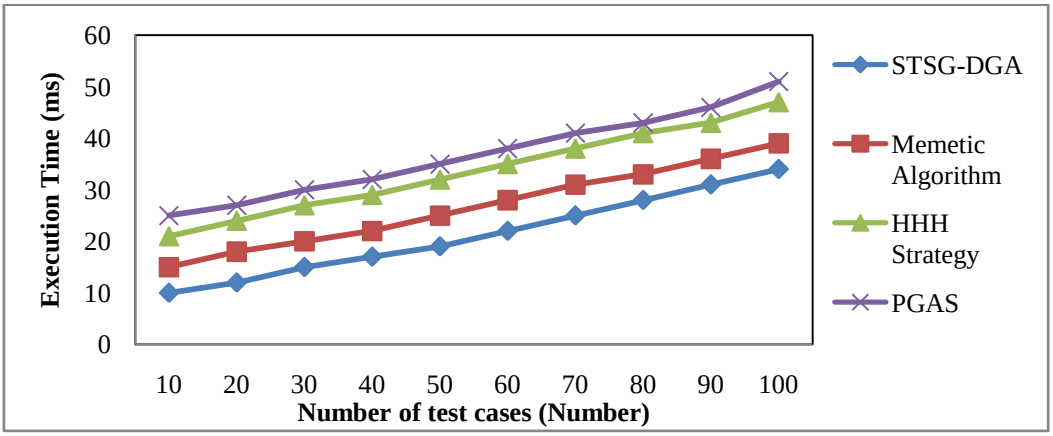

Figure 5. Measurement of Execution Time

3.3. Computational complexity

Computational complexity is defined as the time taken to collect the number of test cases for forming the test suites as given below in the equation 2. It is measured in terms of milliseconds (ms).

$$Computational\ Complexity = Time(collect\ the\ number\ of\ test) \quad (2)$$

Table : 3 Tabulation of Computational Complexity

Number of test cases (Number)	Computational Complexity (ms)			
	STSG-DGA	Memetic Algorithm	HHH Strategy	PGAS
10	9	12	19	23
20	11	15	21	26
30	13	19	24	29
40	16	23	27	31
50	18	26	30	34
60	21	28	31	36
70	23	29	33	38
80	24	32	35	41
90	26	35	37	43
100	29	38	41	45

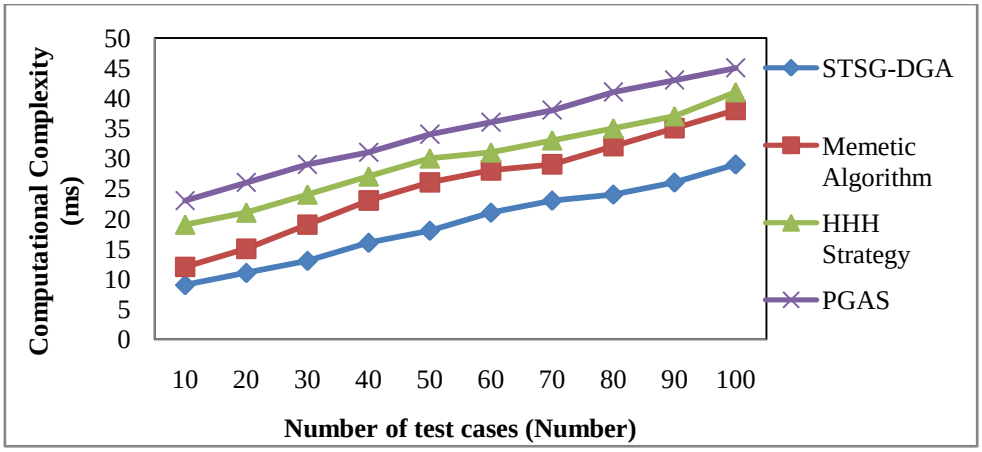

Figure 6. Measurement of Computational complexity

When lower the computational complexity, the method is said to be more efficient. Table 3 explains the computational complexity for different number of test cases in the range of 10 to 100. The computational complexity comparison takes place on existing Memetic Algorithm, HHH strategy, PGAS and proposed STSG-DGA.Figure6 measures the computational complexity of proposed STSG-DGA and existing techniques. Computational complexity of proposed STSG-DGA is comparatively lesser than that of HHH strategy, PGAS and Memetic Algorithm. Research in proposed STSG-DGA has 26.45 % lesser computational complexity than Memetic Algorithm, 37.98% lesser complexity than HHH strategy and 46.68% lesser computational complexity than PGAS.

6. DISCUSSION AND LIMITATION OF SOFTWARE TEST SUITE GENERATION USING DIFFERENT TECHNIQUES

HHH strategy manages the deficiency of each individual algorithm. HHH strategy increases the diversification and intensification of searching process by LLHs depending on their earlier performances. Meta-heuristic based strategies explain the effective solution for attaining good quality solutions.The set of LLHs failed to implement for each problem area. The complete statistical analysis with all the strategies is not performed. An in-depth analysis using PGAS was carried out to learn whether the specific coverage goals. But, redundancy problem in PGAS results in higher execution cost.

Memetic Algorithms combines the local search on test cases and primitive values in global search for test suite. Local search is carried out on numerical inputs to string inputs, arrays, and objects. Memetic algorithm designed the comprehensive approach for object oriented software for managing the test data like strings and arrays. However, local search on test suites is not enhanced and it is unable to achieve higher coverage.

3.1. Future Direction

The future direction of software test suite generation is to further increase the coverage capability and reduce the redundancy using ant colony optimization algorithm.

7. CONCLUSION

The comparison of different techniques for software test suite generation is carried out. The execution time of test suite generation gets minimized using Memetic Algorithm. In Memetic Algorithm, local search on test suites is not enhanced and so it is not able to achieve higher coverage. PGAS studies whether there were specific coverage goals. But, the PGAS results in the increased execution time. A Search based Test Suite Generation using Derived Genetic Algorithm (STSG-DGA) is designed to increase the coverage and to reduce the redundancy for test suite generation. Initial population of randomly produced candidate solutions is used as

search operators and the parent selection is carried out based on fitness function. The reproduction is performed by crossover and mutation operation with probabilities. Fitness of population gets increased and the process gets repeated till the optimal solution is identified. Finally from the result, the research work mainly focus on improving the coverage capability and reducing the redundancy for software test suite generation. The simulation is carried out for different parameters such as computational complexity, coverage and execution time. The results show that STSG-DGA offers better performance with an improvement of coverage by 20 % and reduces the execution time by 34 % compared to existing methods.

REFERENCES

Akram Kalaee, & Vahid Rafe. (2016). An Optimal Solution for Test Case Generation using ROBDD Graph and PSO Algorithm. Wiley Online Publications, 32 (7), 2263–2279.

Anders Hessel, & Paul Pettersson. (2007). A Global Algorithm for Model-Based Test Suite Generation. Electronic Notes in Theoretical Computer Science, Elsevier, 190 (2), 47-59.

Andrea Arcuri, & Gordon Fraser. (2014). On the Effectiveness of Whole Test Suite Generation. Search-Based Software Engineering,Springer, 8636, 1-15.

Bestoun S. Ahmed, Taib Sh. Abdulsamad, & Moayad Y. Potrus. (2015). Achievement of Minimized Combinatorial Test Suite for Configuration-Aware Software Functional Testing Using the Cuckoo Search Algorithm. Information and Software Technology, Elsevier, 66, 13-29.

Chengying Mao, Lichuan Xiao, Xinxin Yu, & Jinfu Chen. (2015). Adapting ant colony optimization to generate test data for software structural testing. Swarm and Evolutionary Computation, Elsevier, 20, 23–36.

Chin-Yu Huang, Chung-Sheng Chen, & Chia-En Lai. (2016). Evaluation and analysis of incorporating Fuzzy Expert System approach into test suite reduction. Information and Software Technology, Elsevier, 79, 79–105.

Emil Alegroth, Robert Feldt, & Pirjo Kolstrom. (2016). Maintenance of automated test suites in industry: An empirical study on Visual GUI Testing. Information and Software Technology, Elsevier, 73, 66–80.

Gordon Fraser, & Andrea Arcuri. (2015). Achieving scalable mutation-based generation of whole test suites. Empirical Software Engineering, Springer, 20 (3), 783–812.

Gordon Fraser, & Andrea Arcuri. (2013). Whole Test Suite Generation. IEEE Transactions on Software Engineering, 39 (2), 276 – 291.

Gordon Fraser, Andrea, & Phil Mc Minn. (2015). A MemeticAlgorithm for whole test suite generation. Journal of Systems and Software, Elsevier, 103, 311-327.

Gregory Gay, Matt Staats, Michael Whalen, & Mats P. E. Heimdahl. (2015). The Risks of Coverage-Directed Test Case Generation. IEEE Transactions on Software Engineering, 41 (8), 803-819.

Johannes Burdek, Malte Lochau, Stefan Bauregger, Andreas Holzer, Alexander von Rhein, Sven Apel, & Dirk Beyer. (2015). Facilitating Reuse in Multi-goal Test-Suite Generation for Software Product Lines. Fundamental Approaches to Software Engineering, 9033, 84-99.

Jose Miguel Rojas, Mattia Vivanti, Andrea Arcuri, & Gordon Frase. (2016). A detailed investigation of the effectiveness of whole test suite generation. Empirical Software Engineering, Springer , 1–42.

Kamal Z.Zamil, Basem Y. Alkazemi, & Graham Kendall. (2016). A Tabu Search hyper-heuristicstrategy for t-way test suite generation. Applied Soft Somputing, Elsevier, 44, 57-74.

Le Thi My Hanh, Nguyen Thanh Binh, & Khuat Thanh Tung. (2016). A Novel Fitness Function of Metaheuristic Algorithms for Test Data Generation for Simulink Models based on Mutation Analysis. Journal of Systems and Software, Elsevier, 120, 17-30.

Robert M. Hierons. (2015). Generating Complete Controllable Test Suites for Distributed Testing. IEEE Transactions on Software Engineering, 41 (3), 279-293.

Rong-Zhi Qi, CCF, Zhi-Jian Wang, & Shui-Yan Li. (2016). A Parallel Genetic Algorithm Based on Spark for Pairwise Test Suite Generation. Journal of Computer Science and Technology, 31 (2), 417–427.

Sami Beydeda, & Volker Gruhn. (2003). Test Case Generation According to the Binary Search Strategy. Computer and Information Sciences (ISCIS), Springer, 2869, 1000-1007.

Shunkun Yang, Tianlong Man, Jiaqi Xu, Fuping Zeng, & Ke Li. (2016). RGA: A lightweight and effective regeneration genetic algorithm for coverage-oriented software test data generation. Information and Software Technology, Elsevier, 76.

Xiangjuan Yao, & Dunwei Gong. (2014). Genetic Algorithm-Based Test Data Generation for Multiple Paths via Individual Sharing. Computational Intelligence and Neuroscience, Hindawi Publishing Corporation, 2014, 1-12.